VIRTUAL UNIVERSITY OF PAKISTAN

Database Management Systems

CS403

Course Handouts

GRADUP

Practice. Pass. Graduate.

thegradup.com

Database Management System

(CS403)

DATABASE MANAGEMENT SYSTEM 1

LECTURE NO. 01 9

Reading Material 9
Overview of Lecture 9
Introduction to the course 9
Database definitions: 10
Importance of the Databases 12
Databases and Traditional File Processing Systems 12
Advantages of Databases 15

LECTURE NO. 02 17

Reading Material 17
Overview of Lecture 17
Difference between Data and Information 17
Further Advantages of Database Systems: 19
Cost Involved: 21
Importance of Data 22
Levels of Data 22
Users of Database Systems: 24

LECTURE NO. 03 31

Reading Material 31
Overview of Lecture 31
Database Architecture: 31
The Architecture: 33
External View (Level, Schema or Model): 35
Conceptual or Logical View: 37

LECTURE NO. 04 40

Reading Material 40
Overview of Lecture 40
Internal or Physical View / Schema 40
Data Independence: 43
Functions of DBMS 45

LECTURE NO. 05 50

Reading Material 50
Overview of Lecture 50
Database Development Process 51
Preliminary Study: 51
Database Development Process: Approach 2 54
Tools Used for Database System Development: 56
Data Flow Diagrams: 56
Types of DFD 60

LECTURE NO. 06 63

Reading Material 63
Overview of Lecture 63
Detailed Data Flow Diagram: 63
Data Dictionary 64
Database Design Phase 67

Data Model 68
Types of Data Models 68
Types of Database Design 69

LECTURE NO. 07 70

Reading Material 70
Overview of Lecture 70
Entity-Relationship Data Model 70
The Entity 71
Classification of entity types 74
Attribute 75
Types of Attributes 77
Summary: 79
Exercises: 79

LECTURE NO. 08 80

Reading Material 80
Overview of Lecture 80
Attributes 80
The Keys 80

LECTURE NO. 09 85

Reading Material 85
Overview of Lecture 85
Relationships 85
Types of Relationships 87

LECTURE NO. 10 91

Reading Material 91
Overview of Lecture 91
Roles in Relationships 95
Dependencies 97
Enhancements in E-R Data Model: 98
Super-type and Subtypes 98
Summary: 99

LECTURE NO. 11 100

Reading Material 100
Overview of Lecture 100
Inheritance Is 100
Super types and Subtypes 101
Specifying Constraints 103
Completeness Constraint 103
Disjointness Constraint 104
Subtype Discriminator 108

LECTURE NO. 12 110

Reading Material 110
Overview of Lecture 110
Steps in the Study of system 110

LECTURE NO. 13 118

Reading Material 118
Overview of Lecture 118

Identification of Entity Types of the Examination System 118
Relationships and Cardinalities in between Entities 120
Conceptual Database Design 122
Logical Database Design 122
Conclusion 123

LECTURE NO. 14 124

Reading Material 124
Overview of Lecture 124
Relational Data Model 125
Introduction to the Relational Data model 126
Mathematical Relations 129
Database Relations 130
Summary 130
Exercise: 131

LECTURE NO. 15 132

Reading Material 132
Overview of Lecture 132
Database and Math Relations 132
Degree of a Relation 133

LECTURE NO. 16 140

Reading Material 140
Overview of Lecture: 140
Mapping Relationships 140
Binary Relationships 140
Unary Relationship 144
Data Manipulation Languages 146
Relational Algebra 147
Exercise: 147

LECTURE NO. 17 148

Reading Material 148
Overview of Lecture: 148
The Project Operator 150

LECTURE NO. 18 157

Reading Material 157
Overview of Lecture: 157
Types of Joins 157
Theta Join: 157
Equi–Join: 159
Natural Join: 159
Outer Join: 161
Semi Join: 161
Relational Calculus 162
Tuple Oriented Relational Calculus: 162
Domain Oriented Relational Calculus: 162
Normalization 162

LECTURE NO. 19 164

Reading Material 164
Overview of Lecture: 164

Functional Dependency 164
Inference Rules 166
Normal Forms 166
Summary 167
Exercise: 168

LECTURE NO. 20 169

Reading Material 169
Overview of Lecture: 169
Second Normal Form 169
Third Normal Form 171
Boyce - Codd Normal Form 173
Higher Normal Forms 175
Summary 175
Exercise: 175

LECTURE NO. 21 176

Reading Material 176
Overview of Lecture: 176
Normalization Summary 176
Normalization Example 177
Physical Database Design 181
Summary 182

LECTURE NO. 22 183

Overview of Lecture 183
The Physical Database Design Considerations and Implementation 183
DESIGNING FIELDS 184
CODING AND COMPRESSION TECHNIQUES: 185

LECTURE NO. 23 187

Reading Material 187
Overview of Lecture 187
Physical Record and De-normalization 187
Partitioning 187
Physical Record and Denormalization 187
Denormalization Situation 1: 188
Partitioning 189

LECTURE NO. 24 191

Reading Material 191
Overview of Lecture 191
Vertical Partitioning 191
Replication 192
Reduced training cost 194
MS SQL Server 194

LECTURE NO. 25 196

Reading Material 196
Overview of Lecture 196
Rules of SQL Format 196
Data Types in SQL Server 197
Summary: 200
Exercise: 200

LECTURE NO. 26 201

Reading Material 201
Overview of Lecture 201
Categories of SQL Commands 201
Summary 205
Exercise: 205

LECTURE NO. 27 206

Reading Material 206
Overview of Lecture 206
Alter Table Statement 206

LECTURE NO. 28 210

Reading Material 210
Select Statement 211
Attribute Allias 213

LECTURE NO. 29 215

Reading Material 215
Overview of Lecture 215
Data Manipulation Language 215

LECTURE NO. 30 220

Reading Material 220
Overview of Lecture 220
ORDER BY Clause 220
Functions in SQL 221
GROUP BY Clause 222
HAVING Clause 223
Cartesian Product 224
Summary 225

LECTURE NO. 31 226

Reading Material 226
Overview of Lecture 226
Inner Join 226
Outer Join 228
Semi Join 230
Self Join 231
Subquery 232
Summary 236
Exercise: 237

LECTURE NO. 32 238

Reading Material 238
Overview of Lecture 238
Application Programs 238
User Interface 239
Forms 240
Tips for User Friendly Interface 243

LECTURE NO. 33 246

Reading Material 246
Overview of Lecture 246

LECTURE NO. 34 255

Reading Material 255
Overview of Lecture 255

LECTURE NO. 35 260

Reading Material 260
Overview of Lecture 260
File Organizations 260

LECTURE NO. 36 265

Reading Material 265
Overview of Lecture 265
Hashing 265
Hash Functions 266
Hashed Access Characteristics 266
Mapping functions 266
Open addressing: 269

LECTURE NO. 37 270

Reading Material 270
Overview of Lecture: 270
Index 270
Index Classification 272
Summary 274

LECTURE NO. 38 275

Reading Material 275
Overview of Lecture 275
Ordered Indices 275
Clustered Indexes 275
Non-clustered Indexes 276
Dense and Sparse Indices 276
Multi-Level Indices 277

LECTURE NO. 39 AND 40 280

Reading Material 280
Overview of Lecture 280
Views 280
To Focus on Specific Data 280
Characteristics /Types of Views: 283
Characteristics of Views 286

LECTURE NO. 41 288

Reading Material 288
Overview of Lecture 288
Updating Multiple Tables 288
Materialized Views 289
Transaction Management 291

LECTURE NO. 42 293

Reading Material 293
Overview of Lecture 293
The Concept of a Transaction 293
Transactions and Schedules 294

© Virtual University of Pakistan

Concurrent Execution of Transactions 295
Serializability 296
Lock-Based Concurrency Control 297
Deadlocks 299

LECTURE NO. 43 302

Reading Material 302
Overview of Lecture 302
Incremental Log with Deferred Updates 302
Incremental Log with Immediate Updates 305
Concurrency Control 307
Summary 308

LECTURE NO. 44 310

Reading Material 310
Overview of Lecture 310
Uncommitted Update Problem 310
Inconsistent Analysis 311
Serial Execution 312
Serializability 315
Locking 315
Summary 316

LECTURE NO. 45 318

Reading Material 318
Overview of Lecture: 318
Locking Idea 319
DeadLock 320
DeadLock Handling 320
Wait – for Graph: 320
Deadlock Resolution 323
Timestamping rules 324

Lecture No. 01

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Chapter 1.

Overview of Lecture

- Introduction to the course
- Database definitions
- Importance of databases
- Introduction to File Processing Systems
- Advantages of the Database Approach

Introduction to the course

This course is first (fundamental) course on database management systems. The course discusses different topics of the databases. We will be covering both the theoretical and practical aspects of databases. As a student to have a better understanding of the subject, it is very necessary that you concentrate on the concepts discussed in the course.

Areas to be covered in this Course:

- **Database design and application development:** How do we represent a real-world system in the form of a database? This is one major topic covered in this course. It comprises of different stages, we will discuss all these stages one by one.

- **Concurrency and robustness:** How does a DBMS allow many users to access data concurrently, and how does it protect against failures?

- **Efficiency and Scalability:** How does the database cope with large amounts of data?

- **Study of tools to manipulate databases:** In order to practically implement, that is, to perform different operations on databases some tools are required. The operations on databases include right from creating them to add, remove and modify data in the database and to access by different ways. The tools that we will be studying are a manipulation language (SQL) and a DBMS (SQL Server).

Database definitions:

Definitions are important, especially in technical subjects because definition describes very comprehensively the purpose and the core idea behind the thing. Databases have been defined differently in literature. We are discussing different definitions here, if we concentrate on these definitions, we find that they support each other and as a result of the understanding of these definitions, we establish a better understanding of use, working and to some extent the components of a database.

Def 1: A shared collection of logically related data, designed to meet the information needs of multiple users in an organization. The term database is often erroneously referred to as a synonym for a "database management system (DBMS)". They are not equivalent and it will be explained in the next section.

Def 2: A collection of data: part numbers, product codes, customer information, etc. It usually refers to data organized and stored on a computer that can be searched and retrieved by a computer program.

Def 3: A data structure that stores metadata, i.e. data about data. More generally we can say an organized collection of information.

Def 4: A collection of information organized and presented to serve a specific purpose. (A telephone book is a common database.) A computerized database is an updated, organized file of machine readable information that is rapidly searched and retrieved by computer.

Def 5: An organized collection of information in computerized format.

Def 6: A collection of related information about a subject organized in a useful manner that provides a base or foundation for procedures such as retrieving information, drawing conclusions, and making decisions.

Def 7: A Computerized representation of any organizations flow of information and storage of data.

Each of the above given definition is correct, and describe database from slightly variant perspectives. From exam point of view, anyone will do. However, within this course, we will be referring first of the above definitions more frequently, and concepts discussed in the definition like, logically related data, shared collection should be clear. Another

important thing that you should be very clear about is the difference between database and the database management system (DBMS). See, the database is the collection of data about anything, could be anything. Like cricket teams, students, busses, movies, personalities, stars, seas, buildings, furniture, lab equipment, hobbies, hotels, pets, countries, and many more anything about which you want to store data. What we mean by data; simply the facts or figures. Following table shows the things and the data that we may want to store about them:

Thing	**Data (Facts or figures)**
Cricket Player	Country, name, date of birth, specialty, matches played, runs etc.
Scholars	Name, data of birth, age, country, field, books published etc.
Movies	Name, director, language (Punjabi is default in case of Pakistan) etc.
Food	Name, ingredients, taste, preferred time, origin, etc.
Vehicle	Registration number, make, owner, type, price, etc.

There could be infinite examples, and please note that the data that is listed about different things in the above table is not the only data that can be defined or stored about these things. As has been explained in the definition one above, there could be so many facts about each thing that we are storing data about; what exactly we will store depends on the perspective of the person or organization who wants to store the data. For example, if you consider food, data required to be stored about the food from the perspective of a cook is different from that of a person eating it. Think of a food, like, Karhahi Ghost, the facts about Karhahi ghosht that a cook will like to store may be, quantity of salt, green and red chilies, garlic, water, time required to cook and like that. Where as the customer is interested in chicken or meat, then black or red chilies, then weight, then price and like that. Well, definitely there are some things common but some are different as well. The thing is that the perspective or point of view creates the difference in what we store; however, the main thing is that the database stores the data.

The database management system (DBMS), on the other hand is the software or tool that is used to manage the database and its users. A DBMS consist of different components or subsystem that we will study about later. Each subsystem or component of the DBMS performs different function(s), so a DBMS is collection of different programs but they all work jointly to manage the data stored in the database and its users. In many books and may be in this course sometimes database and database management system are used interchangeably but there is a clear difference and we should be clear about them. Sometimes another term is used, that is, the database system, again, this term has been used differently by different people, however in this course we use the term database system as a combination of database and the database management system. So database is collection of data, DBMS is tool to manage this data, and both jointly are called database system.

Importance of the Databases

Databases are important; why? Traditionally computer applications are divided into commercial and scientific (or engineering) ones. Scientific applications involve more computations, that is, different type of calculations that vary from simple to very complex. Today such applications exist, like in the fields of space, nuclear, medicine that take hours or days of computations on even computers of the modern age. On the other hand, the applications that are termed as commercial or business applications do not involve much computations, rather minor computation but mainly they perform the input/output operations. That is, these applications mainly store the data in the computer storage, then access and present it to the users in different formats (also termed as data processing) for example, banks, shopping, production, utilities billing, customer services and many others. As is clear from the example systems mentioned, the commercial applications exist in the day to day life and are related directly with the lives of common people. In order to manage the commercial applications more efficiently databases are the ultimate choice because efficient management of data is the sole objective of the databases. So such applications are being managed by databases even in a developing country like Pakistan, yet to talk about the developed countries. This way databases are related directly or indirectly almost every person in society.

Databases are not only being used in the commercial applications rather today many of the scientific/engineering application are also using databases less or more. Databases are concerned of effectively latter form of applications which are more Commercialapplications. The goal of this course is to present an in-depth introduction to databases, with an emphasis on how to organize information in the database and to maintain it and retrieve it efficiently, that is, how to design a database and use it effectively.

Databases and Traditional File Processing Systems

Traditional file processing system or simple file processing system refers to the first computer-based approach of handling the commercial or business applications. That is why it is also called a replacement of the manual file system. Before the use computers, the data in the offices or business was maintained in the files (well in that perspective some offices may still be considered in the pre-computer age). Obviously, it was laborious, time consuming, inefficient, especially in case of large organizations. Computers, initially designed for the engineering purposes were though of as blessing, since they helped efficient management but file processing environment simply transformed manual file work to computers. So processing became very fast and efficient, but as file processing systems were used, their problems were also realized and some of them were very severe as discussed later.

It is not necessary that we understand the working of the file processing environment for the understanding of the database and its working. However, a comparison between the characteristics of the two definitely helps to understand the advantages of the databases

and their working approach. That is why the characteristics of the traditional file processing system environment have been discussed briefly here.

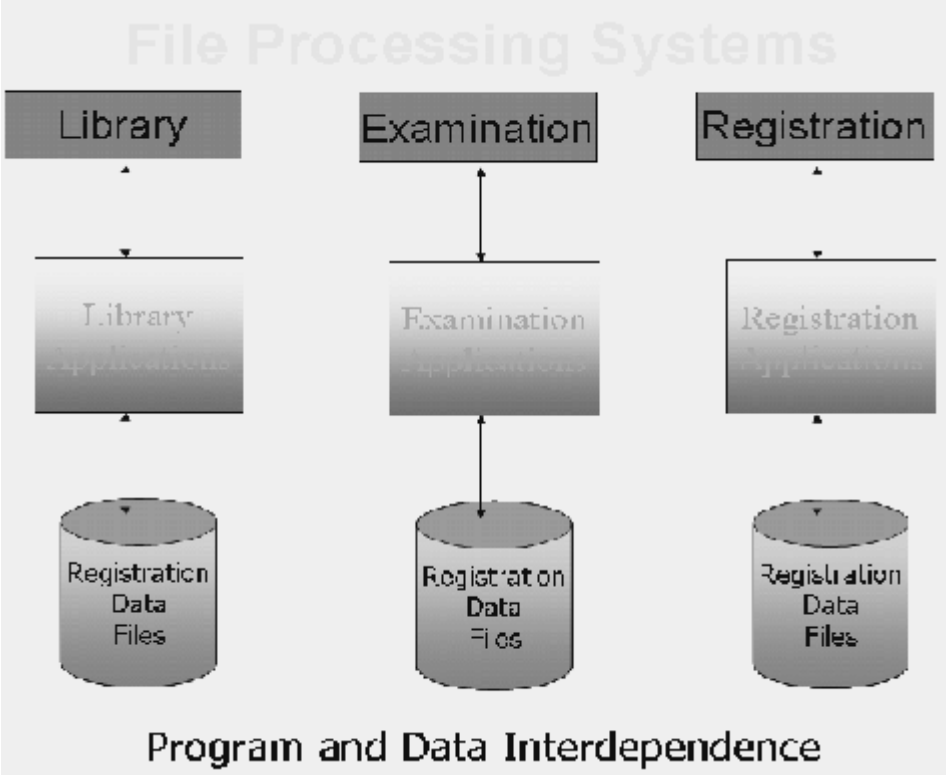


Fig. 1: A typical file processing environment

The diagram presents a typical traditional file processing environment. The main point being highlighted is the **program and data interdependence**, that is, program and data depend on each other, well they depend too much on each other. As a result any change in one affects the other as well. This is something that makes a change very painful or problematic for the designers or developers of the system. What do we mean by change and why do we need to change the system at all. These things are explained in the following.

The systems (even the file processing systems) are created after a very detailed analysis of the requirements of the organizations. But it is not possible to develop a system that does not need a change afterwards. There could be many reasons, mainly being that the users get the real taste of the system when it is established. That is, users tell the analysts or designers their requirements, the designers design and later develop the system based on those requirements, but when system is developed and presented to the users, it is only then they realize the outcome of the effort. Now it could be slightly and (unfortunately) sometimes very different from what they expected or wanted it to be. So the users ask changes, minor or major. Another reason for the change is the change in the requirements. For example, previously the billing was performed in an organization on the monthly

basis, now company has decided to bill the customers after every ten days. Since the bills are being generated from the computer (using file processing system), this change has to be incorporated in the system. Yet another example is that, initially bills did not contain the address of the customer, now the company wants the address to be placed on the bill, so here is change. There could be many more examples, and it is so common that we can say that almost all systems need changes, so system development is always an on-going process.

So we need changes in the system, but due to program-data interdependence these changes in the systems were very hard to make. A change in one will affect the other whether related or not. For example, suppose data about the customer bills is stored in the file, and different programs use this file for different purposes, like adding data into the bills file, to compute the bill and to print the bill. Now the company asks to add the customers' address in the bills, for this we have to change the structure of the bill file and also the program that prints the bill. Well, this was necessary, but the painful thing is that the other programs that are using these bills files but are not concerned with the printing of the bills or the change in the bill will also have to be changed, well; this is needless and causes extra, unnecessary effort.

Another major drawback in the traditional file system environment is the non-sharing of data. It means if different systems of an organization are using some common data then rather than storing it once and sharing it, each system stores data in separate files. This creates the problem of redundancy or wastage of storage and on the other hand the problem on inconsistency. The change in the data in one system sometimes is not reflected in the same data stored in other system. So different systems in organization; store different facts about same thing. This is inconsistency as is shown in figure below.

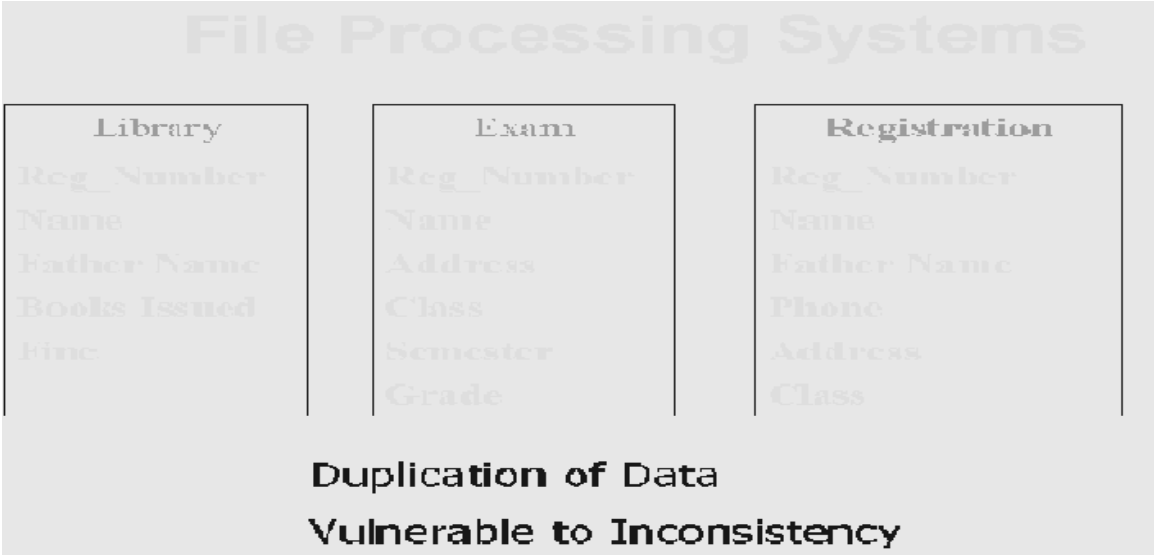


Fig. 2: Some more problems in File System Environment

Previous section highlighted the file processing system environment and major problems found there. The following section presents the benefits of the database systems.

Advantages of Databases

It will be helpful to reiterate our database definition here, that is, database is a shared collection of logically related data, designed to meet the information needs of multiple users in an organization. A typical database system environment is shown in the figure 3 below:

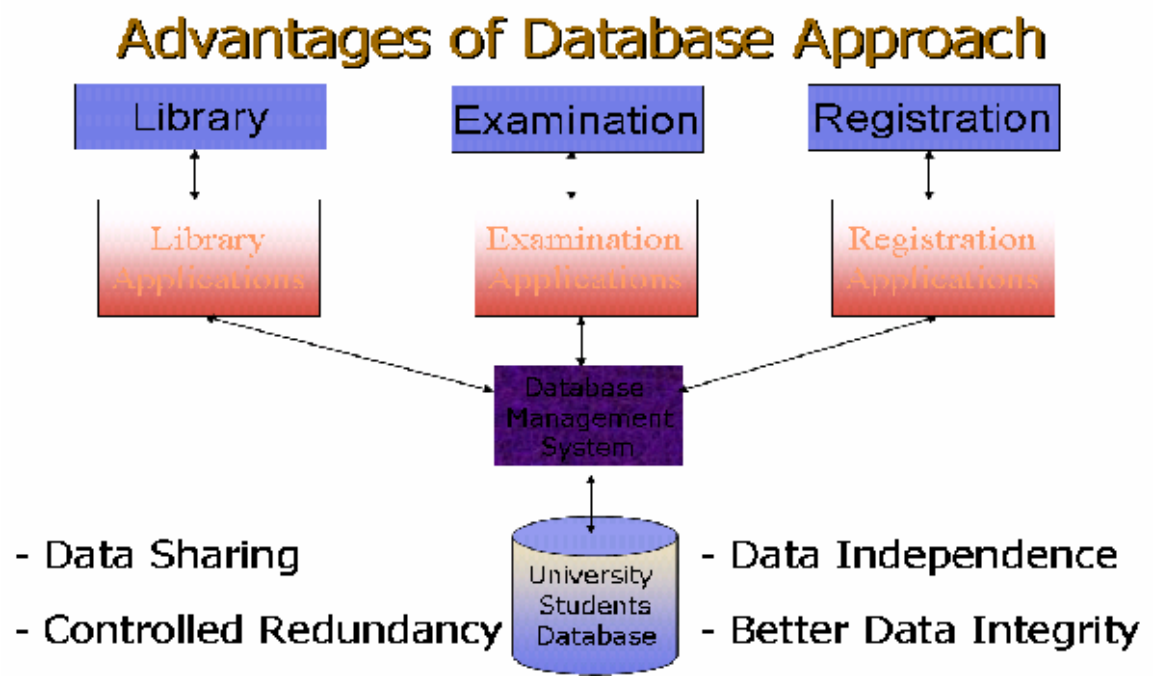


Fig. 3: A typical Database System environment

The figure shows different subsystem or applications in an educational institution, like library system, examination system, and registration system. There are separate, different application programs for every application or subsystem. However, the data for all applications is stored at the same place in the database and all application programs, relevant data and users are being managed by the DBMS. This is a typical database system environment and it introduces the following advantages:

- **Data Sharing**
  The data for different applications or subsystems is placed at the same place. This introduces the major benefit of data sharing. That is, data that is common among different applications need not to be stored repeatedly, as was the case in the file processing environment. For example, all three systems of an educational institution shown in figure 3 need to store the data about students. The example data can be seen

from figure 2. Now the data like registration number, name, address, father name that is common among different applications is being stored repeatedly in the file processing system environment, where as it is being stored just once in database system environment and is being shared by all applications. The interesting thing is that the individual applications do not know that the data is being shared and they do not need to. Each application gets the impression as if the data is being for stored for it. This brings the advantage of saving the storage along with others discussed later.

- **Data Independence**
  Data and programs are independent of each other, so change is once has no or minimum effect on other. Data and its structure is stored in the database where as application programs manipulating this data are stored separately, the change in one does not unnecessarily effect other.

- **Controlled Redundancy**
  Means that we do not need to duplicate data unnecessarily; we do duplicate data in the databases, however, this duplication is deliberate and controlled.

- **Better Data Integrity**
  Very important feature; means the validity of the data being entered in the database. Since the data is being placed at a central place and being managed by the DBMS, so it provides a very conducive to check or ensure that the data being entered into the database is actually valid. Integrity of data is very important, since all the processing and the information produced in return are based on the data. Now if the data entered is not valid, how can we be sure that the processing in the database is correct and the results or the information produced is valid? The businesses make decisions on the basis of information produced from the database and the wrong information leads to wrong decisions, and business collapse. In the database system environment, DBMS provides many features to ensure the data integrity, hence provides more reliable data processing environment.

Dear students, that is all for this lecture. Today we got the introduction of the course, importance of the databases. Then we saw different definitions of database and studied what is data processing then studied different features of the traditional file processing environment and database (DB) system environment. At the end of lecture we were discussing the advantages of the DB approach. There some others to be studied in the next lecture. Suggestions are welcome.

Exercises

- Think about the data that you may want to store about different things around you
- List the changes that may arise during the working of any system, lets say Railway Reservation System

Lecture No. 02

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	

Overview of Lecture

- Some Additional Advantages of Database Systems
- Costs involved in Database systems
- Levels of data
- Database users

Difference between Data and Information

Data is the collection of raw facts collected from any specific environment for a specific purpose. Data in itself does not show anything about its environment, so to get desired types of results from the data we transform it into information by applying certain processing on it. Once we have processed data using different methods data is converted into meaningful form and that form of the Data is called information

Example:

Example:

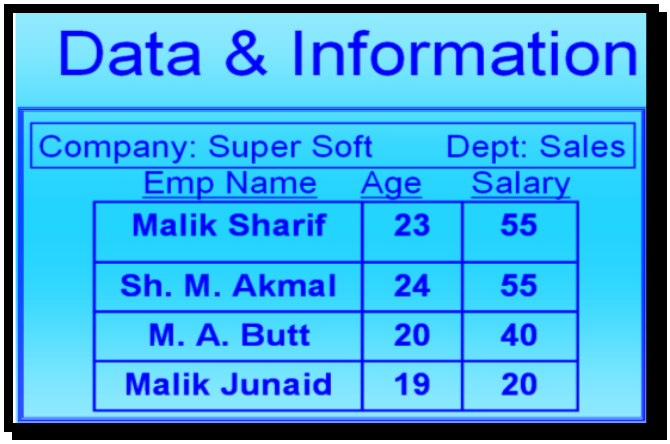


Fig. 1: Data and Information

If we consider the data in the above figure without the titles or the labels associated with the data (EmpName, age, salary) then it is not much useful. However, after attaching these labels it brings some meanings to us, this meaningfulness is further increased when we associate some other labels, like the company name and the department name etc. So this is a very simple example of processing that we can do on the data to make it information.
Once we have clear idea of what data and information is we proceed with another term knows as "schema" Schema is a repository or structure to express the format and other different information about data and database, as we can see from the database definition "Database is a self describing collection of interrelated records." The word self describing means that the data storage and retrieval mechanism and its format is described in the database, Actual place where these definitions and descriptions are performed is database schema.

- **Database Application:**

Database Application is a program or group of programs which is used for performing certain operations on the data stored in the database. These operations may contain insertion of data into a database or extracting some data from the database based on a certain condition, updating data in the database, producing the data as output on any device such as Screen, disk or printer.

- **Database Management Systems:**

Database management system is software of collection of small programs to perform certain operation on data and manage the data.
Two basic operations performed by the DBMS are:

- Management of Data in the Database
- Management of Users associated with the database.

Management of the data means to specify that how data will be stored, structured and accessed in the database.

Management of database users means to manage the users in such a way that they can perform any desired operations on the database. DBMS also ensures that a user can not perform any operation for which he is not allowed. And also an authorized user is not allowed to perform any action which is restricted to that user.

In General DBMS is a collection of Programs performing all necessary actions associated to a database.

Further Advantages of Database Systems:

Database systems are very much beneficent to enterprises and businesses, some of the advantages are listed below:

- Data consistency
- Better data security
- Faster development of new applications
- Economy of scale
- Better concurrency control
- Better backup and recovery procedures

- **Data Consistency:**

Data consistency means that the changes made to different occurrence of data should be controlled and managed in such a way that all the occurrences have same value for any specific data item. Data inconsistency leads to a number of problems, including loss of information and incorrect results. In database approach it is controlled because data is shared and consistency is controlled and maintained.

- **Better Data Security:**

All application programs access data through DBMS, So DBMS can very efficiently check that which user is performing which action and accessing which part of data , So A DBMS is the most effectively control and maintain security of Data stored in a database.

- **Faster Application Development:**

The database environment allows us faster application development because of its many reasons. As we know that database is designed with the factor of future development in mind

So whenever we have to build a new application to meet the growing needs of the computerized environment, it may be easy due to the following reason:

- The data needed for the new application already resides in the database.

- The data might not already reside in the database but it could be derived from the data present in the database

Thus we can say that, to develop a new application for an existing database system less effort is required in terms of the system and database design.

- **Economy of Scale:**

Databases and database systems are designed to share data stored in one location for many different purposes, So it needs not be stored as many number of times in different forms as it is used, for example the data used by Admission Department of any education institution can be used to maintain the attendance record of the students as well as the examination records of the students. So it saves us lots of efforts and finances providing economy of scale.

- **Better Concurrency Control:**

Concurrency means the access of database form as number of points simultaneously. Concurrency control means to access the database in such a way that all the data accesses are completed correctly and transparently. One example of controlled concurrency is the use of ATM Machine for withdrawal of money (cash). All ATM machines of a bank are interconnected to a central database system worldwide, so that a user can access its account from anywhere in the world and can get cash from any ATM terminal. As there are thousands of ATM terminal across the world for a specific bank so as a result thousands of user process and access the bank's database. All this process is managed concurrently using the database systems and is done in such an efficient manner that no two user face any delay in the processing of their requests.

- **Better Backup and Recovery Facility:**

Data is a very important resource and is very much valuable for any organization, loss of such a valuable resource can result in a huge strategic disasters. As Data is stored on today's' storage devices like hard disks etc., It is necessary to take periodic backups of data so that in case a storage device looses the data due to any damage we should be able

to restore the data a nearest point, Database systems offer excellent facilities for taking backup of data and good mechanism of restoring those backups to get back the backed-up data.

It some time happens that a database which was in use and very important transactions were made after the last backup was made, all of a sudden due to any disastrous situation the database crashes (improper shutdown, invalid disk access, etc.) Now in such a situation the database management system should be able to recover the database to a consistent state so that the transactions made after the last backup are not lost.

Cost Involved:

Enjoying all these benefits of the database systems do have some additional costs on any organization which is going to adopt a database environment. These charges may also be known as the disadvantages of the database system. Different types of costs (Financial and Personnel) which an organization faces in adopting a database system are listed below:

- **High Cost:**

Database Systems have a number of inherent charges which are to be born by any organization that is going to adopt it. High Cost is one of these inherent charges, it includes the need for specialized software which is used to run database systems, Additional and specialized hardware and technically qualified staff are the requirements for adopting to the database system, all these requirements need an organization to invest handsome amount of money to have all the requirements of the database systems.

- **Conversion Cost:**

Once an organization has decided to adopt database system for its operations, it is not only the finance and technical man-power which is required for switching on to database system, it further has some conversion charges needed for adopting the database system, this is also a very important stage for making decision about the way the system will be converted to database system.

- **Difficult Recovery Procedures:**

Although the database systems and database management systems provide very efficient ways of data recovery in case of any disaster, still the process of recovering a crashed database is very much technical and needs good professional skills to perform a perfect recovery of the database.

Importance of Data

- **Data as a Resource:**

A resource is anything which is valuable for an organization. There can be a number of resources in any organization, for example, Buildings, Furniture, Vehicle, Technical Staff, Managers, supporting staff and Machinery etc. As all these are resources for organizations and are consumed very much carefully to get full benefit out of them. Data in the same way is a very important resource and needs to considered equally important as other resource are considered.

Why we call data as a resource?

Data is truly considered a resource because for an organization to make proper decisions at proper time it is only the data which can provide correct information and in-turn cause good utilization of other organizational resources. Organizations can not make good and effective decisions if the required data is not available in time or in the correct and desired format, such bad and miscalculated decisions ultimately lead to the failure of organizations or business.

Levels of Data

- **Real World Data**

The real world level of data means that level of data at which entities or objects exist in reality, it means that any object existing in reality have a name and other identifiable attributes through which we can identify that specific object or entity.

Example:

Any Student

- **Meta Data:**

For storage of the data related to any entity or object existing at real world level we define the way the data will be stored in the database. This is called Meta data. Meta data is also known as schema for the real world data. It tells that what type of data will be stored in the database, what will be size of a certain attribute of the real world data, how many and what attributes will be used to store the data about the entity in the database.

Example: Name , Character Type, 25 character size field,
Age, Date type, 8 bytes size
Class, Alpha Numeric, 8 byte size field

- **Existence of Data:**

Existence of the data level shows the actual data regarding the entities as real world level according to the rules define at the Meta Data level.

Example:

According to the definition given in the Meta data level the Actual data or Data occurance for the entity at real world level is shown below:

Name	Age	Class	
Ali	20/8/1979	MCS-I	
Amir	22/3/1978	MCS-II	etc…

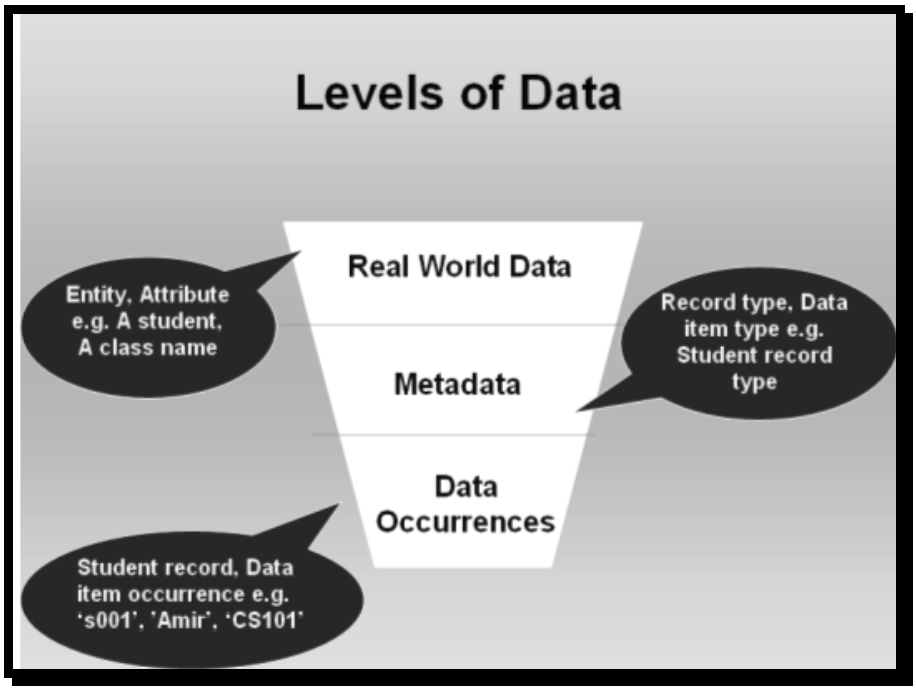


Fig. 2: Levels of Data

Users of Database Systems:

- Application Programmers
- End Users
  - Naïve
  - Sophisticated

- **Application programmers:**

This category of database users contains those people who create different types of database application programs that we have seen earlier. Application programmers design the application according to the needs of the other users of the database in a certain environment. Application programmers are skilled people who have clear idea of the structure of the database and know clearly about the needs of the organizations.

- **End Users:**

Second category of the Database users are the end users, this group of users contains the people who use the database application programs developed by the Application programmers. This category further contains two types of users:

- Naïve Users
- Sophisticated Users

- **Naïve Users**

This category of users is that category who simply use the application database programs created by the programmers. This group has no interaction with other parts of there database and only use the programs meant for them. They have not to worry about the further working of the database.

- **Sophisticated Users:**

This type of users has some additional rights over the Naïve users, which means that they can access the data stored in the database any of their desired way. They can access data using the application programs as well as other ways of accessing data. Although this type of users has more rights to access data, but these users have to take more responsibility and they need to be aware of the database structure. Moreover such users should be skilled enough to be able to get data from database with making and damage or loss to the data in database.

- **Database Administrators (DBA):**

This class of database users is the most technical class of db users. They need to have the knowledge of how to design and manage the database use as well as to manage the data in the database. DBA is a very responsible position in an organization. He is responsible for proper working of the database and DBMS, has the responsibility of making proper database backups and make necessary actions for recovering the database in case of a database crash. To fulfill the requirements of a DBA position a DBA needs vast experience and very elegant technical skills.

- **Duties of the DBA**

A Database administrator has some very precisely defined duties which need be performed by the DBA very religiously. A short account of these jobs is listed below:

- Schema definition
- Granting data access
- Routine Maintenance
  - Backups
  - Monitoring disk space
  - Monitoring jobs running

- **Schema Design**

DBA in some organization is responsible for designing the database schema, which means that DBA is the person who create all the meta data information for the organization on which the database is based. However in some very large scale organizations this job is performed by the database designer, which is hired for the purpose of database design and once the database system is installed and working, it is handed over to the DBA for further operation.

- **Granting Access to Users:**

DBA is also responsible for grant of access rights to the database users. Along with granting and revoking (taking back) the rights the DBA continuously monitors and ensure the legal use of these rights.

- **Monitoring Disk Space :**

When a new database is created it takes a limited space but as a result of daily activity the database acquires more data and grows in size very rapidly. The DBA has to monitor the disk space usage and statistics to ensure that no data over flow occurs at any stage.

- **Monitoring Running Jobs:**

To ensure the secure and proper functioning of the database system a DBA continuously monitors some associated activities also and ensure that all users are using their

authorities legally and different devices attached to the database system are functioning properly.

Typical Components of a Database Environment:

Different typical components of a database environment are shown in the figures below; they describe graphically the role of different types of users.

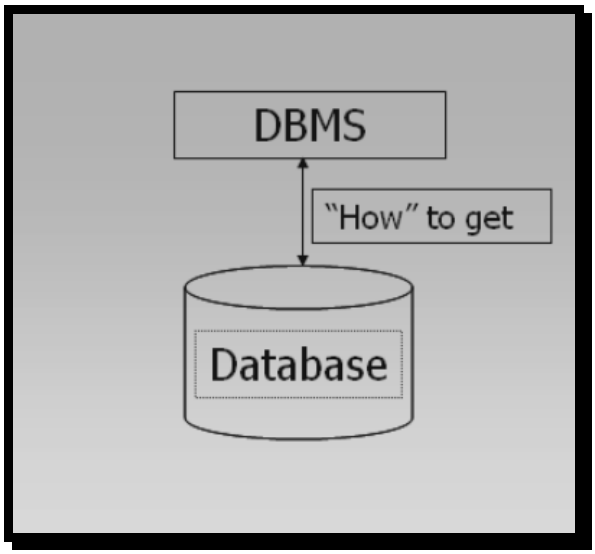


Fig. 3: DBMS and Database

Database is used to store data and DBMS uses mechanisms to get data from the database

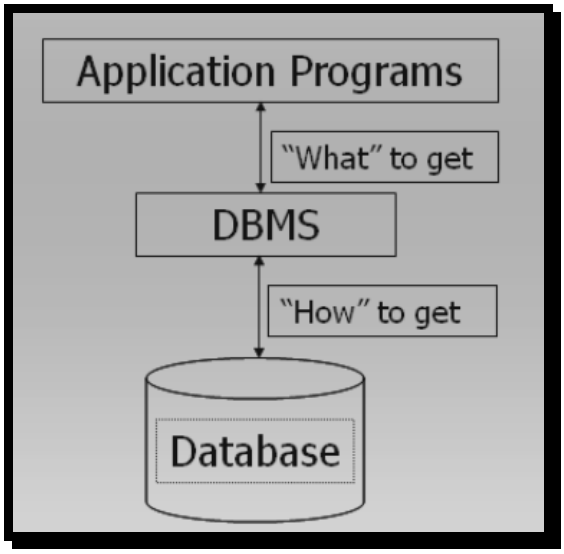


Fig. 4: Application Programs

Application programs talk to DBMS and ask for the data required

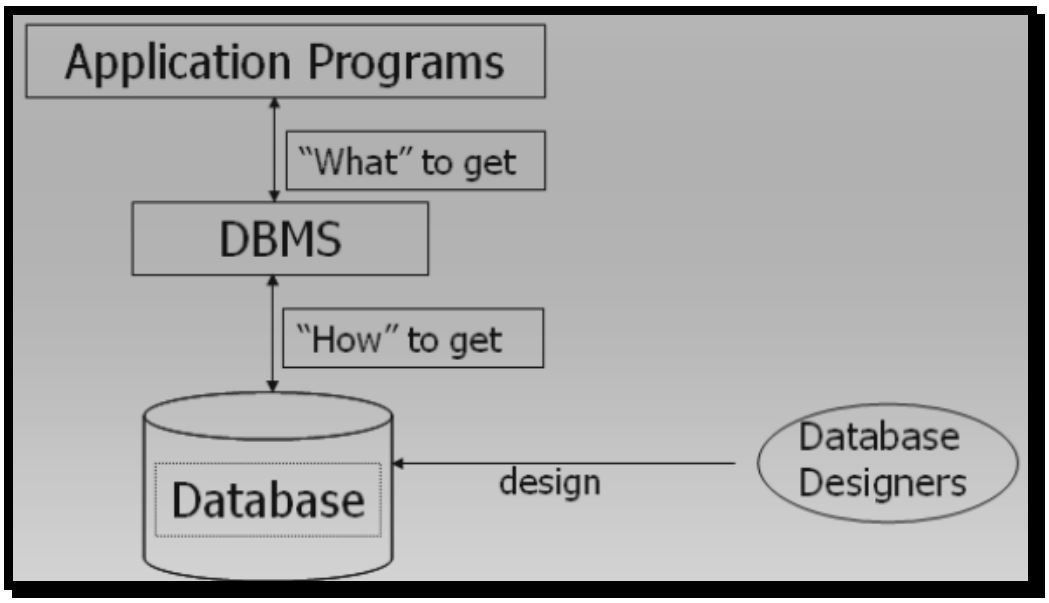


Fig. 5: Database Designers

Database designers design (for large organizations) the database and install the DBMS for use by the users of the database in any specific organization.

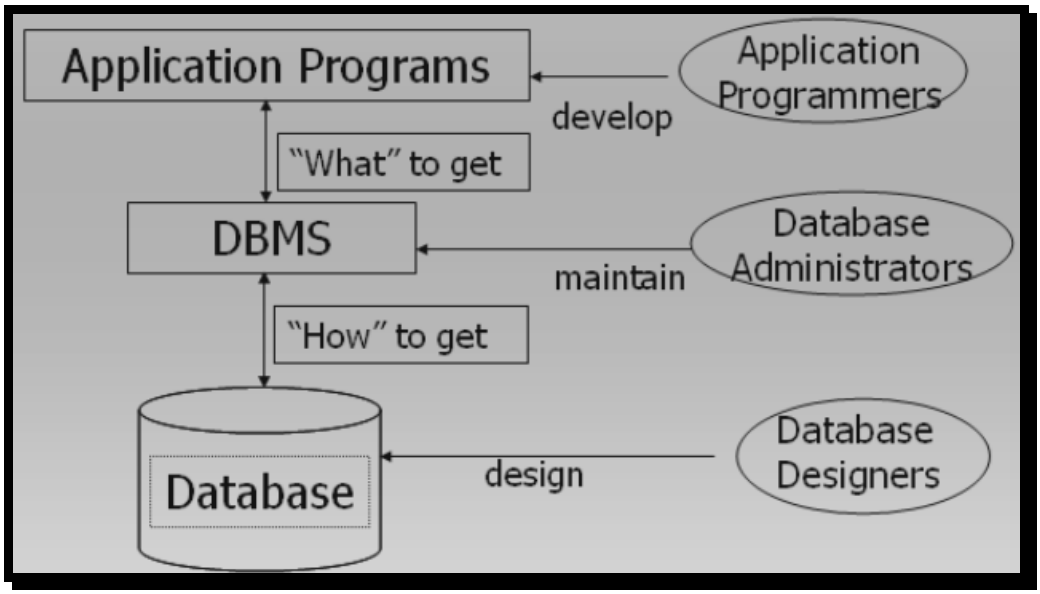


Fig. 6: Database Administrator

Once Database has been installed and is functioning properly in a production environment of an organization the Database Administrator takes over the charge and performs specific DBA related activities including:

- Database maintenance
- Database Backup
- Grant of rights to database users
- Monitoring of Running Jobs
- Managing Print jobs
- Ensuring quality of Service to all users

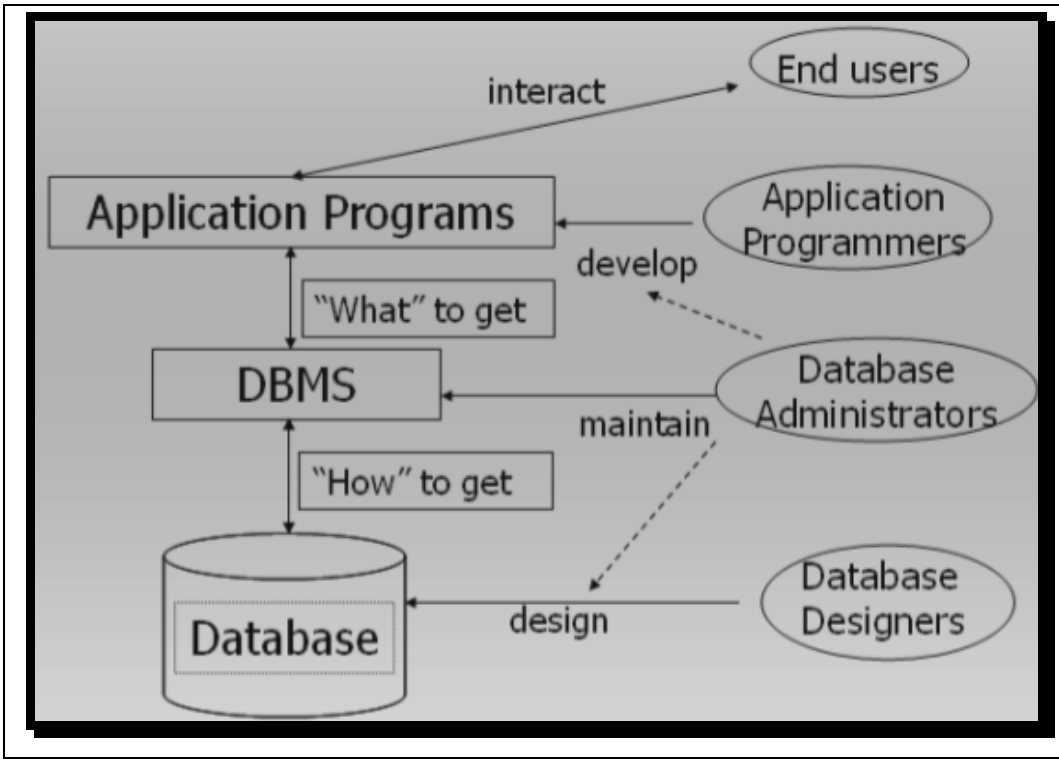


Fig. 7: Database Administration's interaction with other users

- Database administrator can interact with the database designer during database design phase so that he has a clear idea of the database structure for easy reference in future.
- This helps DBA perform different tasks related to the database structure.
- DBA also interacts with the application programmers during the application development process and provides his services for better design of applications.
- End users also interact with the system using application programs and other tools as specified in the description above.

This concludes lecture number 2, in case of any queries, please feel free to contact.

Lecture No. 03

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section: 4.1.1, 4.1.2

Overview of Lecture

- Database Architecture
- External View of the database
- Conceptual view of the database

Database Architecture:

Standardization of database systems is a very beneficent in terms of future growth, because once a system is defined to follow a specific standard, or is built on a specific standard, it provides us the ease of use in a number of aspects.

First if any organization is going to create a new system of the same usage shall create the system according to the standards and it will be easier to develop, because the standards which are already define will be used for developing the system.

Secondly if any organization wants to create and application software that will provide additional support to the system, it will be an easier task for them to develop such system and integrate them into existing database applications.

Users which will be using the system will be comfortable with the system because a system built on predefined standards is easy to understand and use, rather than understanding learning and using an altogether new system which is designed and built without following any standards.

Expansion to systems which are not built on standards is very hard and needs lots of efforts.

Technical staff working on a system built on standard has no problem to learn the use and architecture of the system and whenever there is a need in change of staff new staff members can be hired and put to work without any prior training for the use of system.

Database standard proposed by ANSI SPARK in 1975 is being used worldwide and is the only most popular agreed upon standard for database systems.

The Three Level Schema architecture provides us a number of benefits. For accessing data at different levels we have a number of users because not all users have to access data in database at all the database levels. The 3 levels architecture allows us to separate the physical representation of data from the users' views of data.

In the database, same data is stored in a specific feasible format and is available to different users in different formats as desired by different users. For example, consider we have stored the DOB (Date of Birth) in the database in a particular format, like in the form of dd-mm-yyyy (for example, 28-03-1987). However, the users from different departments may require to view the date of birth in different forms; the examination department may ask it to be displayed as month-day-yyyy (like march-28-1987) the Registrar's office may ask to display date of birth as mm/dd/yyyy, still the Library may need the in the form of dd/mm/yy. The Three Level Schema allows us to access the data in different formats at the external level, which is stored in a specific format at the internal level.

The Three levels architecture is useful for hiding the details of internal systems; it in-fact hides the details of underlying system views from the users at other levels and restricts the access of data and the system from any unauthorized intervention. It is the mechanism which allows us to store the data in the system in such a way that it can be provided to all users in their desired formats and with unveiling other details and information stored in the database. Moreover if there is a change to be done to the data stored in the database subject to the

requirements of a specific user it needs not be changed for that user specifically, we can create a change to the specific external view of that user and the internal details remain unchanged. Also if we want to change the underlying storage mechanism of the data stored on the disk we can do it without affecting the internal and conceptual view at the lowest level in the three levels architecture is the internal view or internal level which is shown below in the diagram and is illustrated in the coming lines.

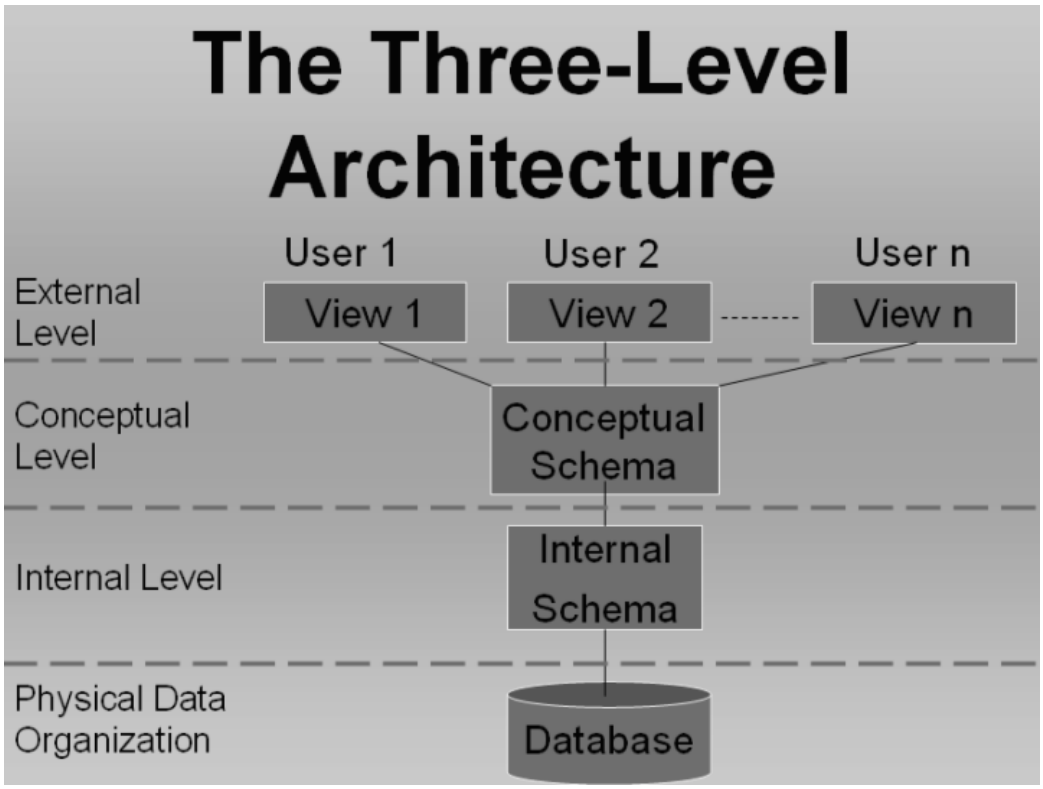


Fig. 1: Three level architecture of database

The Architecture:

The schemas as it has been defined already; is the repository used for storing definitions of the structures used in database, it can be anything from any entity to the whole organization. For this purpose the architecture defines different schemas stored at different levels for isolating the details one level from the other.

Different levels existing at different levels of the database architecture are expressed below with emphasis on the details of all the levels individually.

Core of the database architecture is the internal level of schema which is discussed a bit before getting into the details of each level individually. The internal level implements all the inner details and defines the intentions of the database. Internal schema or view defines the structures of the data and other data related activities in the database. For example it defines that for a student what data will be stored in terms of attributes of the student and it also defines how different values for these attributes will be stored, also it tells that who is allowed to make changes to the database and what changes he can make, etc.

These details give us the internal schema and are called the intention of the database. **Intention** for a database is almost permanent, because while designing the database it is ensured that no information is left behind which is important enough to be stored in the database and what information is important to be stored in the database from the future point of view.

Once the intention of the database has been defined then it is undesirable to change the intention for any reason. Because any small change in the intention of the database may need a lot of changes to be made to the data stored in the database. **Extension** of the database is performed on the bases of a complete intention, i-e once a database has been defined it is populated with the data of the organization for which the database is created. This population of the database is also called as the extension of the database. Extension is always done according to the rules defined in the internal schema design or the intention of the database.

Effects of changes made to different levels of the database architecture:

We can make changes to the different levels of the database but these changes need very serious consideration before they are made, Changes at different levels of database architecture need different levels of users attention for example a change to the data made for the extension of data will effect only a single record whereas when we make a change to the internal level of the

database the change effects all the stored records, similarly an invalid change in the extension of the database is not that fatal as a change in the intention of the database because a change in the extension of the database is not very hard to undo; incase of a mishap whereas a change of the same magnitude to the intention of the database might cause a large number of database errors (inconsistencies and data loss).

External View (Level, Schema or Model):

This level is explicitly an end user level and presents data as desired by the users of the database. As it is known that the database users are classified on two grounds

- Section of the organization
- Nature of Job of the users

The external level of the database caters to the needs of all the database users starting from a user who can view the data only which is of his concern up-to the users who can see all the data in the database and make all type of actions on that data.

External level of the database might contain a large number of user views, each user view providing the desired features and fulfilling requirements for the user or user group for which it is intended. The restriction or liberty a user or user groups get in his rights is the external view of that user groups and is decided very carefully.

External views are also helpful when we want to display the data which is not place in the database or not stored at all. Example of the first case can be a customer Phone number stored in the database. But when contacting the person it might appear that the area code for that specific user is not stored in the database, in that case we can simply pick up the area or city id of the customer and find the area code for that city from the corresponding Area Codes table.

Another situation may arise when we want to get a student enrolled in an institution and want to make sure that the student qualifies for the minimum required age limit, we will look the database, for the students age but if we have

stored only the date of birth of the student then the age of the student needs to be calculated at that very instance; this can be done very easily in the specific user view and age of the student can be calculated, even the user-view itself can tell use whether the student qualifies for the admission or not.

As the user view is the only entity or the interface through which a user will operate the database or use it so it must be designed in such a way that it is easy to use and easy to manage and self descriptive, also it is easy to navigate through. Also it should not allow the user to get or retrieve data which is not allowed to the user, so the user view should both be a facilitator and also a barrier for proper utilization of the database system.

As the system grows it is possible that a user view may change in structure, design and the access it provides to the users. SO External views are designed and create in way that they can be modified at a later stage without making any changes in the logical or internal views.

In the diagram below we can see two different users working as end users having their own external view; we can see that the same data record is displayed in two entirely different ways.

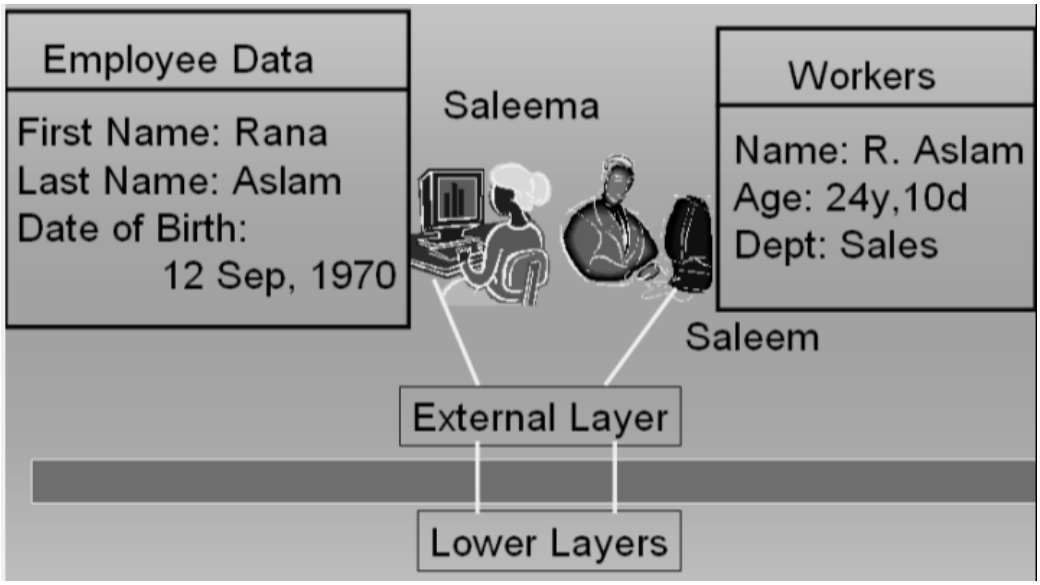


Fig. 2: Mapping between External layer and lower layers

Conceptual or Logical View:

This is the level of database architecture which contains the definition of all the data to be stored in the database and also contains rules and information about that structure and type of that data.

The conceptual view is the complete description of the data stored in the database. It stores the complete data of the organization that is why it is also known as the community view of the database. The conceptual view shows all the entities existing in the organization, attribute or characteristics associated with those entities and the relationships which exist among the entities of the organization.

We can take the example of the customers of a company. Now the conceptual schema will have all the details of the products of the company, retailing stores of the company, products present in the stock, products which are ready to be delivered, salespersons of the company, manager of the company and literally every other thing which is associated with the business of the company in any way.

Now after having all the information we know that the customers buy products from the outlets of the company, thus in such a case a specific customer has a relationship with that specific outlet of the company, or the customer may be represented as having association with the sales person which in-turn has association with the outlet., there may be a number of customers at a certain outlet and also to mange these salespersons there will be one or more managers.

We can see from the above given scenario that all the entities are logically related to each other in way or the other. The conceptual schema actually manages all such relationship and maps these relationships among the member entities. Conceptual schema along-with having all the information which is to be stored in the database stores the definition of the data to be stored. The definition may contain types of data, and constraints on data values etc.

Conceptual schema is also responsible for holding the authorization and authentication information, means that only those people can make use of the database whom we have allowed to make these changes, so therefore it is the task of the DBMS to ensure be checking the conceptual schema that he is authorized to check the data or make any changes to the data.

Conceptual schema as it describes the intention of the database; it is not changed often, because to make a change to the conceptual schema of the database requires lots of consideration and may involve changes to the other views/levels of the database also.

As in the previous example we saw two database users accessing the database and we saw that both of them are having totally different user views. Here when we see in the logical view of the data we can see that the data stored in the database is stored only once and two users get different data from the same copy of data at the underlying conceptual level.

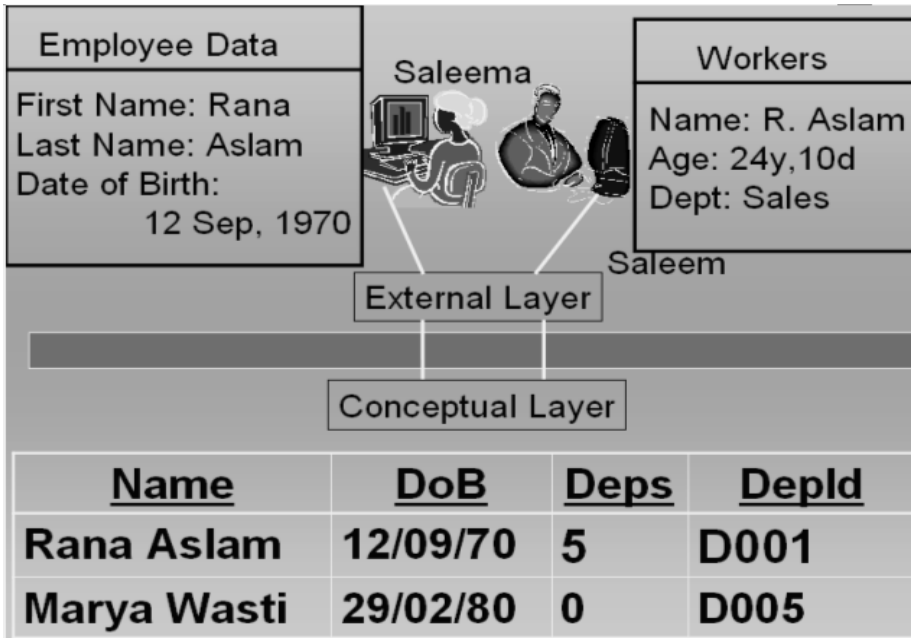


Fig. 3: External and conceptual layers

By summarizing it all we can say that the external view is the view of database system in which user get the data as they need and these database users need

not to worry about the underlying details of the data, all these users have to do is to provide correct requirement information to the DBA or the database designer whoever is designing the database for the system, so that the DBA or the database designer can create the database in such a way that they can fulfill the users requirements using the conceptual schema of the database.

Conceptual view/schema is that view of the database which holds all the information of the database system and provides basis for creating any type of the required user views and can accommodate any user fulfilling his/her requirements.

Exercise:

The data examples that you defined in the exercises of lecture 1, think of the different forms of data at the external and conceptual level. Also try to define mapping between them.

Lecture No. 04

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	4.1.3, 4.1.4
Hoffer	Chapter 2

Overview of Lecture

- Internal Schema of the Database Architecture
- Data Independence
- Different aspects of the DBMS

Internal or Physical View / Schema

This is the level of the database which is responsible for the storage of data on the storage media and places the data in such a format that it is only readable by the DBMS. Although the internal view and the physical view are so close that they are generally referred to a single layer of the DBMS but there lays thin line which actually separated the internal view from the physical view. As we know that data when stored onto a magnetic media is stored in binary format, because this is the only data format which can be represented electronically, No matter what is the actual format of data, either text, images, audio or video. This binary storage mechanism is always implemented by the Operating System of the Computer. DBMS to some extent decides the way data is to be stored on the disk. This decision of the DBMS is based on the requirements specified by the DBA when implementing the database. Moreover the DBMS itself adds information to the data which is to be stored. For example a DBMS has selected a specific File organization for the

storage of data on disk, to implement that specific file system the DBMS needs to create specific indexes. Now whenever the DBMS will attempt to retrieve the data back form the file organization system it will use the same indexes information for data retrieval. This index information is one example of additional information which DBMS places in the data when storing it on the disk. At the same level storage space utilization if performed so that the data can be stored by consuming minimum space, for this purpose the data compression can be performed, this space optimization is achieved in such a way that the performance of retrieval and storage process is not compromised. Another important consideration for the storage of data at the internal level is that the data should be stored in such a way that it is secure and does not involve any security risks. For this purpose different data encryption algorithms may be used. Lines below detail further tidbits of the internal level.

The difference between the internal level and the external level demarcates a boundary between these two layers, now what is that difference, it in fact is based on the access or responsibility of the DBMS for the representation of data. At the internal Level the records are presented in the format that are in match with schema definition of the records, whereas at the physical level the data is not strictly in record format, rather it is in character format., means the rules identified by the schema of the record are not enforced at this level. Once the data has been transported to the physical level it is then managed by the operating system. Operating system at that level uses its own data storage utilities to place the data on disk.

Inter Schema Mapping:

The mechanism through which the records or data at one level is related to the changed format of the same data at another level is known as mapping. When we associate one form of data at the external level with the same data in another form is know as the external/conceptual mapping of the data. (We have seen examples of external/conceptual mapping in the previous lecture) In the same way when data at the conceptual level is correlated with the same data at the internal level, this is called the conceptual/Internal mapping.

Now the question arises that how this mapping is performed. Means how is it possible to have data at one level in date format and at a higher level the same data show us the age. This hidden mechanism, conversion system or the formula which converts the date of birth of an employee into age is performed by the mapping function and

it is defined in the specific ext/con mapping, for example, when the data at the conceptual level is presented as the age of the employee is done by the external schema of that specific user. Now in this scenario the ext/con mapping is performing the mapping with the internal view and is retrieving the data in desire format of the user.
In the same way the mapping between an internal view and conceptual view is performed.
The figure below gives a clear picture of this mapping process and informs where the mapping between different levels of the database is performed.

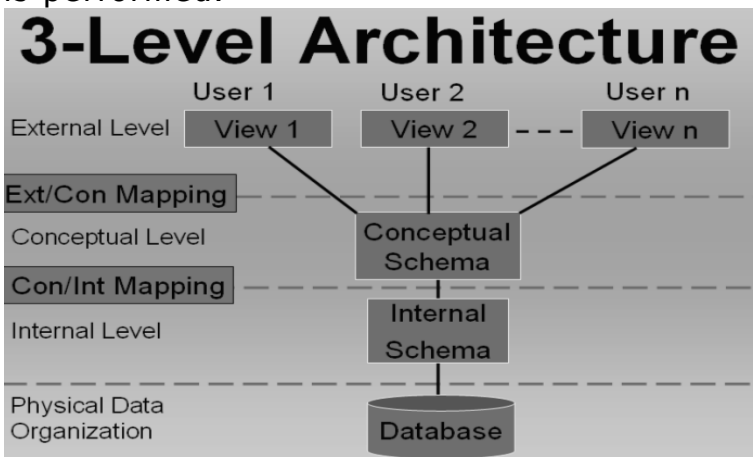


Fig:1: Mapping between External/Conceptual and Conceptual/Internal levels

In Figure-1 we can see clearly where the mapping or connectivity is performed between different levels of the database management system. Figure-1 is showing another very important concept that the internal layer and the physical layers lie separately the Physical layer is explicitly used for data storage on disk and is the responsibility of the Operating system. DBMS has almost no concern with the details of the physical level other than that it passes on the data along-with necessary instructions required to the store that data to the operating system.

Figure-2 on the next page shows how data appears on different levels of the database architecture and also at that of physical level. We can clearly see that the data store on the physical level is in binary format and is separate from the internal view of data in location and format. Separation of the physical level from the internal level is of great use in terms of efficiency of storage and data retrieval.

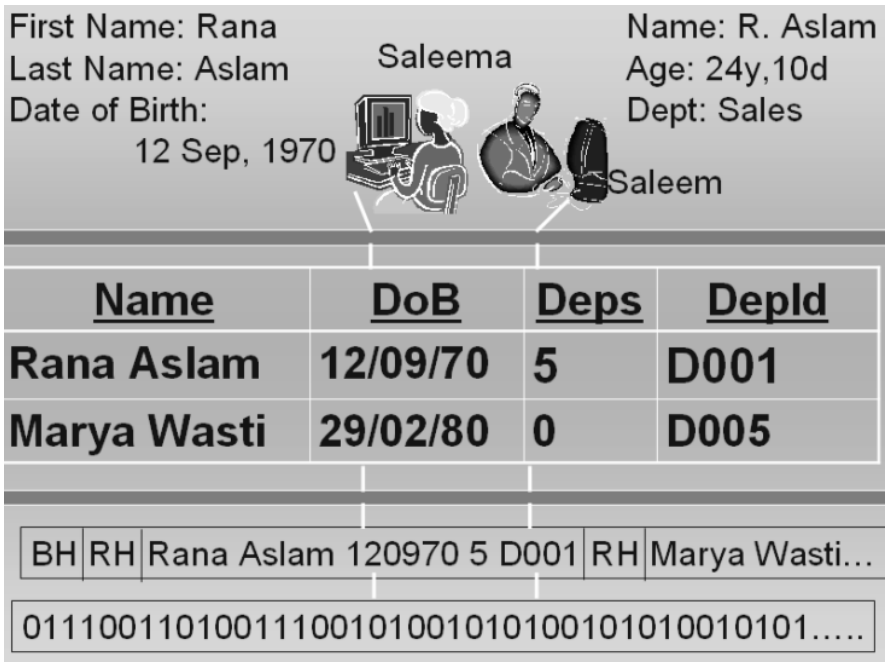


Fig: 2. Representation of data at different levels of data base Architecture and at the physical level at bottom

At the internal level we can see that data is prefixed with Block Header and Record header RH, the Record header is prefixed to every record and the block header is prefixed to a group of records; because the block size is generally larger than the record size, as a result when an application is producing data it is not stored record wise on the disk rather block wise which reduces the number of disk operations and in-turn improves the efficiency of writing process.

Data Independence:

Data Independence is a major feature of the database system and one of the most important advantages of the Three Level Database Architecture. As it has been discussed already that the file processing system makes the application programs and the data dependent on each other, I-e if we want to make a change in the data we will have to make or reflect the corresponding change in the associated applications also.

The Three Level Architecture facilitates us in such a way that data independence is automatically introduced to the system. In other words we can say the data independence is major most objective of the Three Level Architecture. If we do not have data independence then whenever there will be a change made to the internal or

physical level or the data accessing strategy the applications running at the external level will demand to be changed because they will not be able to properly access the changed internal or physical levels any more. As a result these applications will stop working and ultimately the whole system may fail to operate.

The Data independence achieved as a result of the three level architecture proves to be very useful because once we have the data , database and data applications independent of each other we can easily make changes to any of the components of the system, without effecting the functionality and operation of other interrelated components.
Data and program independence is on advantage of the 3-L architecture the other major advantage is that ant change in the lower level of the 3-L architecture does not effect the structure or the functionality on upper levels. I-e we get external/conceptual and conceptual/internal independence by the three levels Architecture.
Data independence can be classified into two type based on the level at which the independence is obtained.

- **Logical Data Independence**
- **Physical Data Independence**

Logical data independence
Logical data independence provides the independence in a way that changes in conceptual model do not affect the external views. Or simply it can be stated at the Immunity of external level from changes at conceptual level.
Although we have data independence at different levels, but we should be careful before making a change to anything in database because not all changes are accepted transparently at different levels. There may be some changes which may cause damage or inconsistency in the database levels. The changes which can be done transparently may include the following:

- Adding a file to the database
- Adding a new field in a file
- Changing the type of a specific field

But a change which may look similar to that of the changes stated above could cause problems in the database; for example: Deleting an attribute from the database structure,

This could be serious because any application which is using this attribute may not be able to run any more. So having data independence available to us we still get problem after a certain change, it means that before making a certain change its impact should also be kept in mind and the changes should be made while remaining in the limits of the data independence.

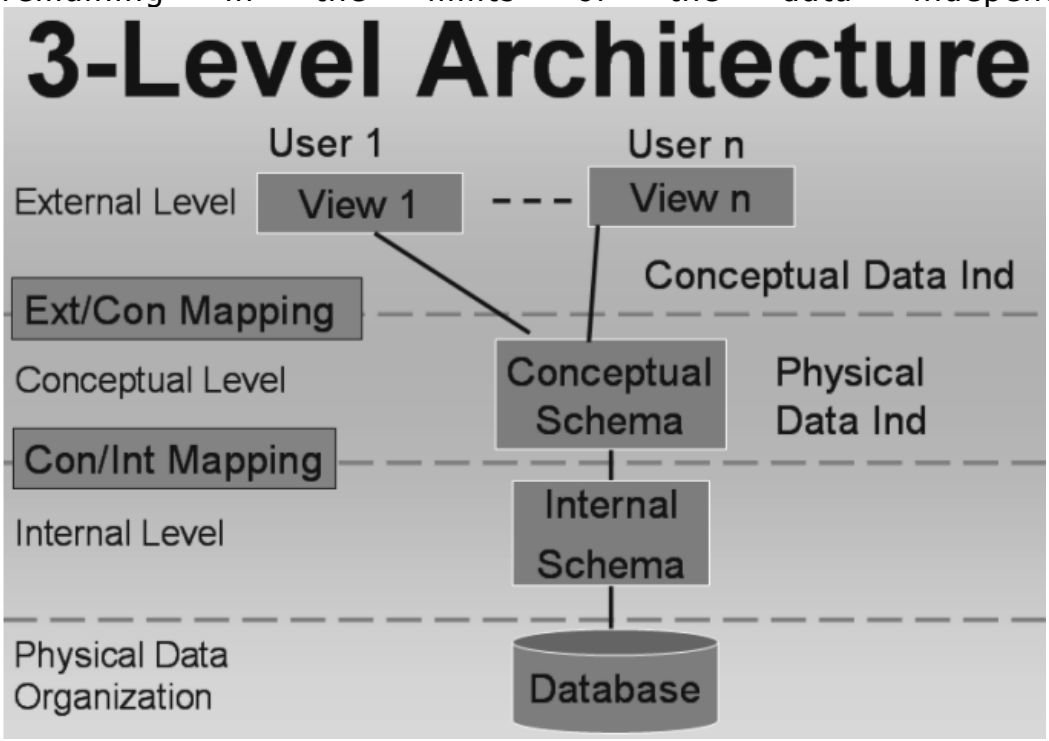


Fig:3. The levels where the Conceptual and Physical data independence are effective

Physical Data Independence

Physical data independence is that type of independence that provides us changes transparency between the conceptual and internal levels. I-e the changes made to internal level shall not affect the conceptual level. Although the independence exist but as we saw in the previous case the changes made should belong to a specific domain and should not exceed the liberty offered by the physical data independence. For example the changes made to the file organization by implementing indexed or sequential or random access at a later stage, changing the storage media, or simply implement a different technique for managing file indexes or hashes.

Functions of DBMS

- **Data Processing**

- **A user accessible Catalog**
- **Transaction Support**
- **Concurrency Control Services**
- **Recovery Services**
- **Authorization Services**
- **Support for Data Communication**
- **Integrity Services**

DBMS lies at the heart of the course; it is the most important component of a database system. To understand the functionality of DBMS it is necessary that we understand the relation of database and the DBMS and the dissection of the set of functions the DBMS performs on the data stored in the database.
Two important functions that the DBMS performs are:

User management
Data Management

The detailed description of the above two major activities of DBMS is given below;

- **Data Processing**

By Data management we mean a number of things it may include certain operations on the data such as: creation of data, Storing of the data in the database, arrangement of the data in the databases and data-stores, providing access to the data in the database, and placing of the data in the appropriate storage devices. These action performed on the data can be classified as data processing.

- **A User Accessible Catalog**

DBMS has another very important task known as access proviso to catalog. Catalog is an object or a place in the DBMS which stores almost all of the information of the database, including schema information, user information right of the users, and many more things about the database. Modern relational DBMS require that the Administrative users of the database should have access to the catalog of the database.

- **Transaction Support**

DBMS is responsible for providing transaction support. Transaction is an action that is used to perform some manipulation on the data stored in the database. DBMS is responsible for supporting all the required operations on the database, and also manages the

transaction execution so that only the authorized and allowed actions are performed.

- **Concurrency Support**

Concurrency support means to support a number of transactions to be executed simultaneously, Concurrency of transactions is managed in such a way that if two or more transactions is making certain processing on the same set of data, in that case the result of all the transactions should be correct and no information should be lost.

- **Recovery Services**

Recovery services mean that in case a database gets an inconsistent state to get corrupted due to any invalid action of someone, the DBMS should be able to recover itself to a consistent state, ensuring that the data loss during the recovery process of the database remains minimum.

- **Authorization Services**

The database is intended to be used by a number of users, who will perform a number of actions on the database and data stored in the database, The DBMS is used to allow or restrict different database users to interact with the database. It is the responsibility of the database to check whether a user intending to get access to database is authorized to do so or not. If the user is an authorized one than what actions can he/she perform on the data?

- **Support for Data Communication**

The DBMS should also have the support for communication of the data indifferent ways. For example if the system is working for such an organization which is spread across the country and it is deployed over a number of offices throughout the country, then the DBMS should be able to communicate to the central database station. Or if the data regarding a product is to be sent to the customers worldwide it should have the facility of sending the data of the product in the form of a report or offer to its valued customers.

- **Integrity Services**

Integrity means to maintain something in its truth or originality. The same concept applies to the integrity in the DBMS environment. Means the DBMS should allow the operation on the database which are real for the specific organization and it should not allow the false information or incorrect facts.

DBMS Environments:

- **Single User**
- **Multi-user**
  - **Teleprocessing**
  - **File Servers**
  - **Client-Server**

- **Single User Database Environment**

This is the database environment which supports only one user accessing the database at a specific time. The DBMS might have a number of users but at a certain time only one user can log into the database system and use it. This type of DBMS systems are also called Desktop Database systems.

- **Multi-User Database systems**

This is the type of DBMS which can support a number of users simultaneously interacting with the database in different ways. A number of environments exist for such DBMS.

- **Teleprocessing**

This type of Multi user database systems processes the user requests at a central computer, all requests are carried to the central computer where the database is residing, transactions are carried out and the results transported back to the terminals (literally dumb terminals). It has become obsolete now.

- **File Servers**

This type of multi-user database environment assumes another approach for sharing of data for different users. A file server is used to maintain a connection between the users of the database system. Each client of the network runs its own copy of the DBMS and the database resides on the file server. Now whenever a user needs data from the file server it makes a request the whole file containing the required data was sent to the client. At this stage it is important to see that the user has requested one or two records from the database but the server sends a complete file, which might contain hundreds of records. Now if the client after making the desired operation on the desired data wants to write back the data on the database he will have to send the whole file back to the server, thus causing a lot of network overhead. The Good thing about this approach is that the server does not have lots of actions to do rather it remains idle for lots of the time in contrast with that of the teleprocessing systems approach.

- **Client-Server**

This type of multi-user environment is the best implementation of the network and DBMS environments. It has a DBMS server machine which runs the DBMS and to this machine are connected the clients having application programs running for each user. Once a users wants to perform a certain operation on data in the database it sends its requests to the DBMS through its machine's application software; the request is forwarded to the DBMS server which performs the required operation on data in the database stored in the dame computer and then passes back the result to the user intending the result. This environment is best suited for large enterprises where bulk of data is processed and requests are very much frequent.

This concludes the topics discusses in the lecture No4.In the next lecture Database application development process will be discussed

Exercises:

- Extend the format of data from the exercise of previous lecture to include the physical and internal levels. Complete your exercise by including data at all three levels
- Think of different nature of changes at all three levels of database architecture and see, which ones will have no effect on the existing applications, which will be adjusted in the inter-schema mapping and which will effect the existing applications.

Lecture No. 05

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	2.3.2, 2.4

Overview of Lecture

- Database Application Development Process
- Preliminary Study of System
- Tools used for Database system Designing
- Data Flow Diagrams
- Different types of Data flow Diagram

Database design and Database Application design are two almost similar concepts, form the course point of view it is worthwhile to mention that the course is mainly concerned with designing databases and it concentrates on the activities which are performed during the design of database and the inner working of the database. The process that will be discussed in this lecture for development of database is although not a very common one, but it specifies all the major steps of database development process very clearly. There exist many ways of system and database development which are not included in the scope of this course. But we will see only those portions of the other processes which are directly related with the design and development of database.

Database Application development Process includes the Following Stages or steps:

- **Database Design**
- **Application Programs**
- **Implementation**

These three steps cannot always be considered as three independent steps performed in a sequence or one after another. Rather, they occur in parallel, which means that from a certain point onward the application programs development may run in parallel with the database design stages, specially the last stages of the database design. Similarly while the design phases of the database are in progress, certain phases of the application programs can also be initiated, for example, the initial study of the screens' format or the reports layout. The database design process that we are going to discuss in this course does not take these steps independently and separately, and since the major concern of this course is the design stages of the database, it concentrate only on those.

- **Database Design:**

This part of the database application development process is most important process with respect to the database application development, because the database is something that will hold the organizations' data, in case the design of the database is not correct or is not correctly reflecting the situations or scenarios of the organization then it will not produce correct result, or even just produce errors in response to certain queries. So this portion of the database design is given great attention when designing a database application.

Database Development Process

The database development process means the same thing that we have mentioned as database application development process. Rather than discussing three stages of database application development separately, the steps given in the database development process include steps that cover all three phases mentioned for the database application development process.

Preliminary Study:

Design of database is carried out in a number of steps; these steps play important role in the design process and need to be given proper attention First Phase of the database development process is the Preliminary Stage, which is based on the proper study of the system. It means that all the parts of the systems, or the section of the subject

organization for which we intend to develop the system must be studied. We should find the relation or interaction of different section of the organization with each other and should understand the way information flows between different sections of the organization. Moreover it should also be made clear that what processing is performed at each stage of the system.

- **Requirement Analysis:**

Once we have investigated the organization for its different sections and the way data flows between those sections. Detailed study of the system is started to find out the requirements of each section. This phase is the detailed study of the system and its functionality decisions made at this stage decide the overall activity of the organization. Requirements of one section of the organization are fulfilled in such a way that all the sections in the organization are supporting each other, for example we can say that the results produced by the processing taking place at one section are used as input for another section. All the users of the systems are interviewed and observed to pinpoint and precisely define the activities taking place in the different section of the organization.

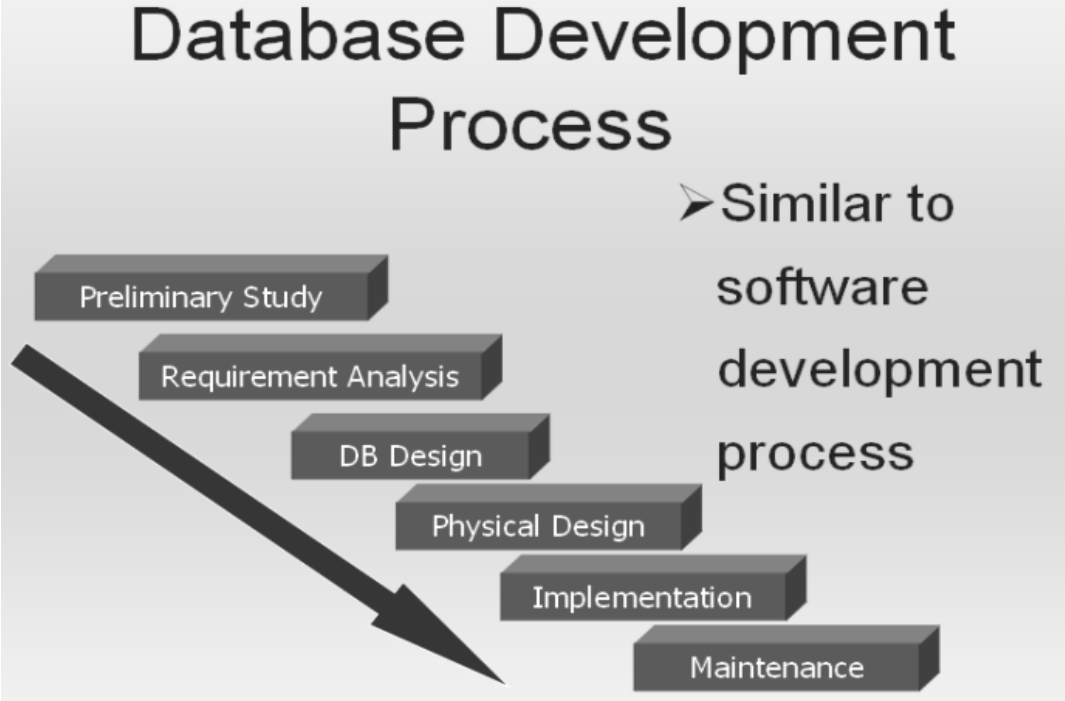


Fig: 1. Database Development process

- **Database Design:**

Third stage in the database development process is the database design; this is a rather technical phase of the process and need handsome skill as a Database Administrator. This is the phase where the logical design of the database is created and different schemas for the database are created logically. Entities are identified and given attributes, relationships are built and different types of entity mappings are performed.

- **Physical Design**

This is the phase where we transform our logical design into a Physical design by implementing the designed database onto a specific DBMS; the choice of the DBMS is made on the basis of requirements and the environment in which the system will operate. Implementing a database on a specific DBMS is very important because it involves the major financial investment of the organization, and can not be reverted in case a selected DBMS in not capable of providing the desired efficiency.

- **Implementation:**

This phase is specific to writing the application programs needed to carry out different activities according to use requirements. Different users may have different requirements of the data in the database, so the number of application programs is not known or fixed for all the organizations, it may vary for different organizations.

- **Maintenance of the Database System:**

Maintenance means to fine tune the system and check that the designed applications systems are fulfilling the purpose for which they are meant. Also this phase may involve designing any new application for the enhancement of the system. Or an already working application may need to be updated or modified to remove any errors or to add some functionality in the system. The phases involved in the development of the database application are expressed graphically in Figure-1.

All these stages are necessary and must be given the necessary attention at each level to get properly working and good system design and a better working environment.

Database Development Process: Approach 2

There are other development processes also with some of the stages or steps modified as compared to the model we have just studied. Such and alternative is given in the Figure-2 below. In this design process we see some of the design stages which existed in the previous designing steps but some of the stages are modified or merged with others to get more precise result or to distinguish different separate design phases. In this process of designing; the following steps exist:

- Analyze User Environment
- Develop Conceptual Model
- Map Conceptual Model to Logical
- Choose DBMS
- Develop Physical Design
- Implement System
- Test System
- Operational Maintenance

- **Analyze User Environment**

This is same step as we discussed while discussing the previous designing process

- **Develop Conceptual Model**

Next stage in this process model is the development of conceptual model or schema Here we actually transform the studied and analyzed information into the conceptual design of the database, this stage may also be connected with the requirement analysis phase, as expressed in the diagram by showing an arrow from this stage back to the first stage.

- **Map Conceptual Model to Logical Model**

Third stage is the mapping of the developed conceptual model to the logical model of the database, means at this stage the schema rules are defined and identified for general database structures.

- **Choose DBMS**

Once the mapping of the conceptual and logical model is done, the decision for the use of DBMS is made; again we refer to the previous model for selecting of the DBMS and will take care of all the necessary requirements of the environment before making a decision.

- **Develop Physical Design**

Once we have selected a DBMS, the logical design is then transformed into physical design. This also includes considering many other decisions, like, data type allocation, indexes to be created, file organizations, etc. Physical database design is achieved by using the DBMS specific rules for schema definition and all the facilities provided by the DBMS,

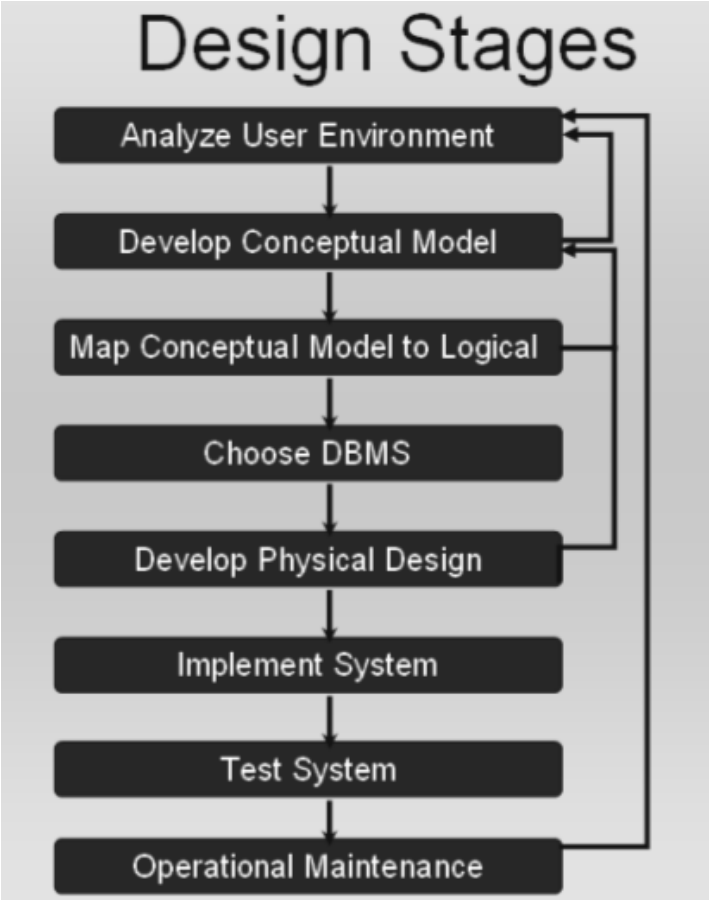


Fig: 3. Database Development Stages. (Second Approach)

- **Implement System**

This stage is also similar to the one described earlier, i.e., designing the application for different users and user groups of the organization.

- **Test System**

Testing is important in the sense that an application may be producing incorrect results, and this incorrectness may lead to the inconsistency of the system. So when a system

design is complete, once it is implemented it must be tested for proper operation and all the modules must be checked for their correctness. Whether the system modules are important or not because the result of the system is mostly dependent on the proper the functionality of all database applications and modules.

- **Operational Maintenance:**

Maintenance means to check that all parts of the system are working and once the testing of the system is completed the periodic maintenance measure are performed on the system to keep the system in working order.

Tools Used for Database System Development:

Why tools are used?

Tools are used for describing the design process in standard ways. If there is no standardized tool available for designing a specific systems; Then everyone will have to use its own design notation, and a notation used by one designer may not be understandable to the another one. This misunderstanding can be more drastic if both the designers are working for the development of the same system. Tools can also help the designer and the user to mutually agree on a specific design.

Data Flow Diagrams:

The most common tool used for deigning database systems is Data Flow Diagram. It is used to design systems graphically and expresses different system detail in different DFD levels.

DFDs show the flow of data between different processes o a specific system.

DFDs are simple, and hide complexities.

DFDs are Descriptive and links between processes describe the information flow.

- **Limitation of DFDs**

They do not provide us a way of expressing decision points.

DFDs are focused on flow of information only.

- **Symbols used in DFD:**

There are a limited number of symbols which are used for design process in DFDs.

- **DATAFLOW:**

The purpose of the dataflow in a DFD is to express the flow of information from one entity to another entity in the system

Data flows are pipelines through which packets of information flow.

Arrows are labeled with name of the data that moves through them. Figure-4 below show the Dataflow diagram



Fig: 4. Dataflow Symbol

- **DATA STORE:**

Data store is a repository for the storage of the data. When in a system the data is to be permanently stored somewhere for future reference or use the DATASTORE is used for this purpose. It is express with a rectangle open on right width and left width of the rectangle drawn with double lines.

Data in the DATASTORE is held sometimes for processing purposes also i-e it may not be a permanent data store.. Name of the DATASTORE is a noun which tells the storing location in the system. Or identifies the entity for which data is stored. Figure-5 shows a data store.



Fig: 5. Data store

- **Processes:**

Processes are expressed with ovals or rounded rectangles. Processes are used to express the transformation of incoming dataflow into outgoing dataflow. Process symbols are used for whatever is the action taking place and whatever is the magnitude or complexity of the action. Simply stating when data is transformed from one form into another the process symbol is used. Figure-6a and Figure-6b show two different shapes used for presenting process in DFD.

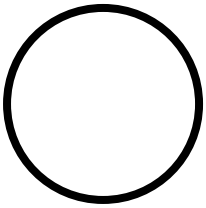


Fig: 6a Process

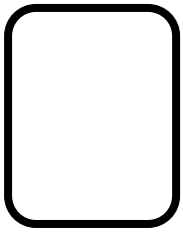
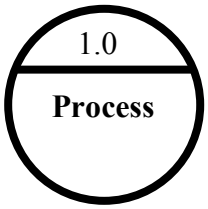


Fig-6b

- **DFD-Process:**

In DFD processes are numbered for expressing their existence at a certain level in the system.



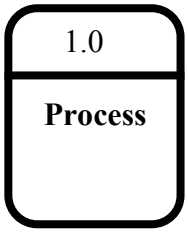


Fig: 7. Numbered DFD Processes

- **External Entities:**

These are the entities interacting with the system in any of two different ways. They may be either receiving the data from the system, or may be producing the data for the system to consume.

Shape used to express external entities is rectangle. The shape for external entity is shown in Figure-8.



Fig: 8. External Entity

- **Collector:**

This DFD shape is used to express several dataflow connections terminating at a single location. Collector is used to show the convergence of data to a single point. Fig 9a shows the Collector symbol and Fig 9b show a collector symbol acting as a sink for multiple data flows.

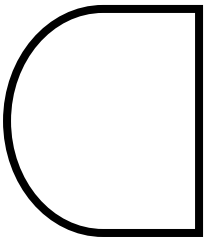


Fig: 9a Collector

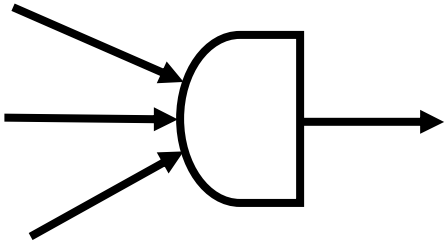


Fig 9b. Collector with Multiple Dataflow

- **Separator:**

The dataflow symbol which is used for separating data from a single source to multiple sinks is known as a separator.

Figure 10a show the presentation of separator and the figure 10b shows the separator as it may appear in a DFD.

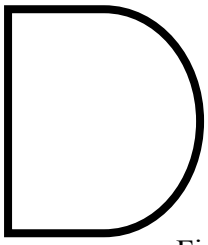


Fig: 10a Separator

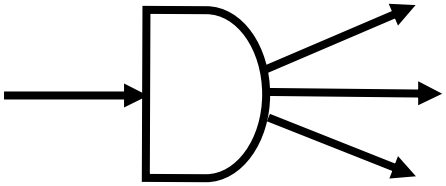


Fig 10b. Separator with Multiple Dataflow

- **Ring Sum Operator:**

This operator is used when data from a source process can flow to one of the mentioned sinks. For this purpose the symbol used is displayed in Figure: 11a and its presentation in a DFD is expressed in Figure-11b.

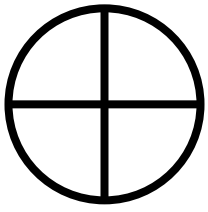


Fig: 11a Ring sum operator

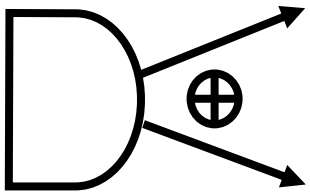


Fig 11b. Separator with Ring sum operator

- **AND Operator:**

This operator is used when data from a source process must flow to all the connected sinks. For this purpose the symbol used is displayed in Figure: 12a and its presentation in a DFD is expressed in Figure-12b.

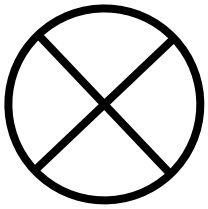


Fig: 12a AND operator

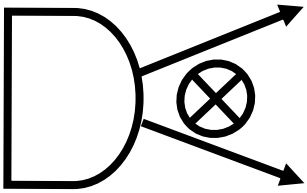


Fig 12b. Separator with AND operator

Types of DFD

- Context diagram
- Level 0 diagram
- Detailed diagram

- **Context Diagram:**

This is the level of DFD which provides the least amount of details about the working of the system. Context DFDs have the following properties:

They always consist of single process and describe the single system. The only process displayed in the CDFDs is the process/system being analyzed. Name of the CDFDs is generally a Noun Phrase.

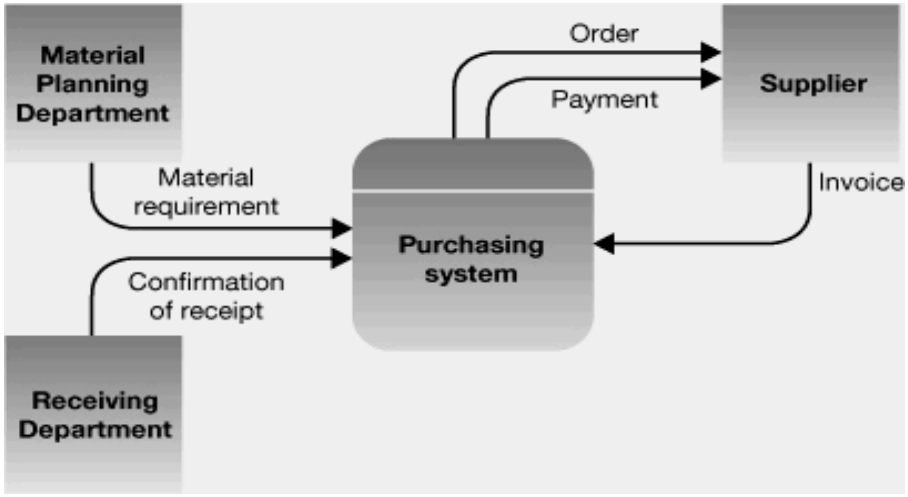


Fig: 13a. Example Context DFD Diagram

No System details are shown in the Contexts DFDs just context is shown. Input and output from and to the process are shown and interactions are shown only with the external entities. An example DFD at context level is shown in Figure: 13a and 13b.

In the context level DFDs no data stores are created. Ant dataflow from external entities are only directed toward the purported system and vice versa, no communication is show between external entities themselves.

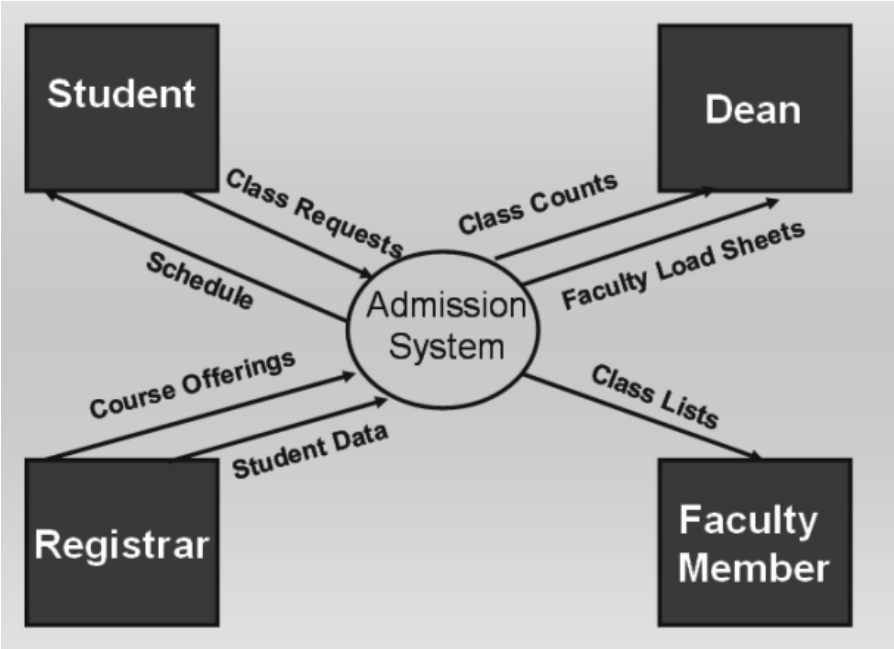


Fig: 13b. Example Context DFD Diagram

- **Level 0 Data Flow Diagrams:**

The level 0 Diagram in the DFD is used to describe the working of the whole system. Once a context DFD has been created the level zero diagram or level 'not' diagram is created. The level zero diagram contains all the apparent details of the system. It shows the interaction between a numbers of processes and may include a large number of external entities. At this level it is the duty of the designer to keep a balance in describing the system using the level 0 diagram. Balance means that he should give proper depth to the level 0 diagram processes. Because placing too much details and showing all of the miniature processes in the level 0 diagrams makes it too much complex. On the other hand it is also not recommended to just ignore even larger processes of the system, because in such a case although the level 0 DFD will become simple but now we will have to create large number of detail DFDs. So a balance in describing the system should be kept so that the depth of the Level 0 DFD is manageable.

- **Steps in creating the level 0 DFD**

1. Identify distinct modules of the system for which to create the DFD

2. Create DFDs for all the modules one by one to show the internal functionality of the system.
3. Once DFD for the distinct modules of the system have been created, establish link between different DFDs where required by either connecting the entities of the system, processes of the system or the data stores in different DFDs.
4. Now comes to the stage of placing the numbers on processes.
   As we know that the level 0 diagram encompasses a large number of smaller systems, ant is a combination of a number of context DFDs. In level 0 diagram a process when it has a lot of details, it is not explained further in the level 0, and rather it is postponed for the detailed diagram.
   In the detailed Data Flow and is given a number. Numbering processes is based on a specific notation, in the level 0 diagrams only left half or the portion before the decimal point is valid but in the detailed diagram when a complex process is expressed further its sub processes are number like 1.0, 1.1, and 1.2 and so on.

Lecture No. 06

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section 2.4

Overview of Lecture

- Detailed DFD Diagrams:
- Database Design Phase
- Data Models
- Types of Data Models
- Types of Database Designs

Detailed Data Flow Diagram:

This Type of the Data flow diagrams is used when we have to further explain the functionality of the processes that we showed briefly in the Level 0 Diagram. It means that generally detailed DFDS are expressed as the successive details of those processes for which we do not or could not provide enough details.

The symbols and other rules regarding the detailed DFD are same as are in other types of DFDs. The special features associated with this diagram are that, one, it is optional, that is, it is created for only those processes from the level 0 diagram for which we want to show the details. For a small sized system we may not need to develop even a single detailed DFD, since the level 0 diagram might be covering it sufficiently. Second specific characteristic of the detailed DFD is its processes' numbering. Numbering of processes in the detailed DFD is done on the basis of numbering of the particular process in level 0 diagrams whose sub-processes are being included in the detailed DFD. For example, a specific process which was numbered in the level 0 diagram as 1.0 or 1 may have a number of sub-processes since we did not represent the process 1.0 in detail in level 0 diagrams. So in the detailed dataflow diagram we create sub-processes of that process and then number all the sub processes of that specific process as the sublets of the process.

Numbering of such sub processes is done as 1.1, 1.2, and 1.3… for first second and third sub-processes of the process 1.0 respectively. The phenomenon of creating sub-processes does not end at creating a few sub-processes for a specific process shown at level 0 diagrams. Rather it may continue deeper if there is requirement for further explanation of the any process or sub-processes. In such a case when we create sub-process of a sub-process 1.2 then the numbering is done in further extension of that specific sub processes number and example of such a numbering process is 1.2.1, 1.2.2, 1.2.3,…

Another point that is worth mentioning here is that we call processes in the detailed DFDs as sub-processes, but they are sub-processes only in reference to the process whose details they are explaining otherwise they are just like processes; transforming some input data into some form of output. The sub-processes may be performing relatively small amount of operations, still they are processes.

Maximum Number of Process in a DFD should not be very huge. Having a moderate number for a detailed DFD is also recommended because it adds clarity to our detailed data flow diagram. For clarity propose it is good to have a maximum of 7 or 9 processes in one detailed DFD. Moreover all the processes, sub processes, data stores, entities data flows and all other components of the DFD must be named properly, so that anyone who is using this DFD should be able to understand the DFD easily.

In all the levels of DFD it must be considered that all the processes have data inputs as well as data outputs. Data being sent to one process should be processed so that it changes its form and transforms from one form to another.

When creating a detailed diagram the data inputs and data outputs must be in coincidence, mean in both the diagrams the data input to a process and data output in the form of data flows must be same.

Data Dictionary

A database that containing data about all the databases in the database system. Data dictionaries store all the various schema and file specifications and their locations. They also contain information about which programs use which data and which users are interested in which reports.

Types of Data Dictionaries:

- **Integrated**

There are basically two types of data dictionaries which are available for use by a DBMS, with respect to their existence.

The first type of data dictionary in this context is the integrated data dictionary. Such a data dictionary is place embedded into the database system, and is created by the DBMS for its usage under the directions and requirements provided by the DBA

As the DBMS needs to talk with the "three level architecture" of database and mapping information along with all the database design information lies in the database schema. The DBMS uses the data dictionary to access the database at each layer or model, for this purpose the data dictionary of any type can be used but the integrated data dictionary is far more efficient than any free standing data dictionary because an integrated data dictionary is created by the DBMS itself and uses the same data accessing techniques etc.

- **Free Standing**

Second type of data dictionary is free standing data dictionary create by any CASE tool and then attached to the database management systems. A number of case tools are available for this purpose and help user designing the database and the database applications as well in some modern forms of the CASE tools.

Cross Reference Matrix

This is a tool available in the data dictionary and helps us in finding entities of the database and their associations. CRM is developed at the designing stage of the database; we can say that at the time of creation of the user views of reports for certain users we identify the material required by the users. In the cross reference matrix, on the Y axis we specify the accessible components of the database such as transitions, reports, or database objects and on the x axis we specify the attributes that will be accessed in the corresponding accessed object.

Now the matrix gets a shape of two dimensional arrays on which we have accessible objects of the database and on the other hand we have the elements which are available for access through those objects. Then whichever data item is accessible through a certain object we place a tick on the intersection of that row and column and thus we can easily identify the deferent items accessed in different reports.

	Transcript	SemResCard	AttendSheet	ClassResSub	cClassResult
courseNam	✓	✓		✓	
cumulative	✓	✓			✓
date	✓	✓		✓	✓
fatherNam	✓	✓			
finalMarks				✓	
grade				✓	✓
grdPoint	✓	✓		✓	✓
marks	✓	✓			
midTerm				✓	
programNa	✓	✓	✓	✓	✓
semesterG	✓	✓			✓
semesterN	✓	✓	✓	✓	✓
semName			✓	✓	✓
session	✓	✓			
sessMarks				✓	
stName	✓	✓			
stNames			✓	✓	
stRegistrat	✓	✓			

Table 1: An example cross reference matrix

The cross reference matrix shown in table 1 lists different attributes against different reports required by different user groups of an exam system. Rows in this matrix contain different attributes and the columns contain different reports. Now the tick mark in the cells represents the use or presence of attributes in different reports. This matrix represents, on one side, the relative importance or use of different attributes. On the other hand it also helps to identify different entity types and their defining attributes. The attributes that are represented collectively on one or more reports are candidates of combining into a single entity type. Although it is necessary that attributes appearing together should be grouped into same entity type, but still they are candidates for combining into the one.

Data Dictionary in not very necessary for using such a cross reference matrix, instead for relatively small systems it can be created manually.

Outcome of the Analysis Phase

In the preliminary study phase, database designers collect information about the existing system from the users of the system. For this purpose they may interview different users or concerned persons, or they may distribute questionnaires among different users and ask them to fill them in and later may use these questionnaires in the analysis phase. Designers represent their understanding of the working of existing system in the form of DFDs and discuss it with the users to make it sure that they have understood all details of the existing system and the requirements of different users groups.

The DFDs are input to the analysis phase, where designers analyze the requirements of the users and establish the procedure to meet those requirements. From the database perspective, in the analysis phase designers have to identify the facts or data that is required to be stored in order to fulfill the users' requirements. For this purpose they may use some CASE tools, like cross reference matrix. Generally, in the analysis phase, designers prepare a draft or initial database design that they ultimately finalize in the next phase, that is, the database design phase. So in short we can say, that DFDs are the output of the preliminary phase and are input to the analysis phase. The initial design or a draft form of design (generally in entity-relationship data model) is the output of the analysis phase and input to the design phase. In the design phase, then you finalize the design.

The sequence of the activities mentioned above is not much important, however, the activities mentioned are important and must be performed in order to have a correct database or database application design. In the following lectures, we are going to study different tools that are used in the design phase, that is, the data models. We will be studying, both, the data models and their implementation in the database design phase.

Database Design Phase

Database design phase follows the analysis phase. Before starting the discussion on the design activity, it will be wise if we clearly understand some basic concepts that are frequently used in this phase.

- **Database Design /Database Model**

These terms can be used interchangeably for the logical structure of the database. The database design/model stores the structure of the data and the links/relationships between data that should be stored to meet the users' requirements. Database design is stored in the database schema, which is in turn stored in the data dictionary.

- **Database Modeling**

The process of creating the logical structure of the database is called database modeling. It is a very important process because the designing of the application provides us the basis for running our database system. If the database is not designed properly the implementation of the system can not be done properly. Generally the design of the database is represented graphically because it provides an ease in design and adds flexibility for the understanding of the system easily.

Data Model

Data model is a set or collection of construct used for creating a database and producing designs for the databases. There are a few components of a data model:

- **Structure:**

What structures can be used to store the data is identified by the structures provided by the data model structures.

- **Manipulation Language**

For using a certain model certain data manipulations are performed using a specific language. This specific language is called data manipulation language.

- **Integrity Constraints**

These are the rules which ensure the correctness of data in the database and maintain the database in usable state so that correct information is portrayed in designing the database.
Generally these components are not explicitly defined in data models, they may be available in some of the modern DBMSs but in traditional and general model, these may not be available.

Significance of the Data Model

Data model is very important tool because it is something which is sued for designing the database for a DBMS and no DBMS can exist independent of any data model, now if we use a specific DBMS but are not sure about the data model it uses for data abase usage, we can not create a proper database.
As a specific DBMS is base on the use of a specific data model so when using a DBMS it is of great use to know that what structures, manipulation languages and integrity constraints are implemented by a specific DBMS. As it is the only way to know the facilities and functionalities offered by the DBMS.
This is the reason whenever we get a specific DBMS, it is explicitly mentioned with that DBMS, that which data model this DBMS uses.

Types of Data Models

- **Semantic Data Model**

These are the data models which provide us better flexibility in implementing constraints, better language utilities and better data structure constructs. As a result actions performed using proper data and structure tools gives us better data designing and manipulation facilities. A better data model provides better opportunities to express multiple situations in the database design and as a result get better output from the tool or model in the form of a better database design.

- ER- Data Model
- Object oriented data model

- Record Based Data Model

This is the second type of data models available to use and has three basic types

- Hierarchical Data Model
- Network Data model
- Relational Data model

These models are records based and are not in similarity with those of semantic data models. These models handle the data at almost all the three level of the three layers of the database architecture. Semantic data models are generally used for designing the logical or conceptual model of the database system, once very common example of the semantic data model is ER-Data Model and is very much popular for designing databases. No DBMS is based on ER Data model because it is purely used for designing whereas a number of DBMS are available based on OO data model, network data model, relational data model l and hierarchical data model.

Types of Database Design

Conceptual database design
This design is implemented using a semantic data model, for example for creating a design for an organization database we can use and we do use the ER-Data model.

Logical Database design
This design is performed using a data model for which we have a DBMS available and we are planning to run our database system that DBMS.

Physical Database Design:
The Logical design created using a specific data model and created after the analysis of the organization, it needs to be implemented in a physical DBMS software so the Physical database design is performed and the design created so far in the logical form are implemented on that very DBMS.

By separating the three design levels we get the benefit of abstraction on one hand whereas on the other hand we can create our logical and conceptual designs using better design tools, which would have not been possible if we are using the same design-tool for al the three levels. Moreover if in future there is a need to make a change in the physical implementation of the data we will have to make no changes in the logical or conceptual level of the database design , rather the change can be achieved by only using the existing conceptual model and implementing it again on Physical model using a separate DBMS.

Lecture No. 07

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section: 5.2 5.3
Hoffer	Page: 85 - 95

Overview of Lecture

- Entity
- Different types of Entities
- Attribute and its different types
- In the previous lecture we discussed the importance and need of data models. From this lecture we are going to start detailed discussion on a data model, which is the entity relationship data model also known as E-R data model.

Entity-Relationship Data Model

It is a semantic data model that is used for the graphical representation of the conceptual database design. We have discussed in the previous lecture that semantic data models provide more constructs that is why a database design in a semantic data model can contain/represent more details. With a semantic data model, it becomes easier to design the database, at the first place, and secondly it is easier to understand later. We also know that conceptual database is our first comprehensive design. It is independent of any particular implementation of the database, that is, the conceptual database design expressed in E-R data model can be implemented using any DBMS. For that we will have to transform the conceptual database design from E-R data model to the data model of the particular DBMS. There is no DBMS based on the E-R data model, so we have to transform the conceptual database design anyway.

A question arises from the discussion in the previous paragraph, can we avoid this transformation process by designing our database directly using the data model of our selected DBMS. The answer is, yes we can but we do not do it, because most commercial DBMS are based on the record-based data models, like Hierarchical, Network or Relational. These data models do not provide too much constructs, so a database design

in these data models is not so expressive. Conceptual database design acts as a reference for many different purposes. Developing it in a semantic data model makes it much more expressive and easier to understand, that is why we first develop our conceptual database design in E-R data model and then later transform it into the data model of our DBMS.

Constructs in E-R Data Model
The E-R data model supports following major constructs:

- Entity
- Attribute
- Relationship

We are going to discuss each one of them in detail.

The Entity

Entity is basic building block of the E-R data model. The term entity is used in three different meanings or for three different terms and that are:

- Entity type
- Entity instance
- Entity set

In this course we will be using the precise term most of the time. However after knowing the meanings of these three terms it will not be difficult to judge from the context which particular meaning the term entity is being used in.

Entity Type
The entity type can be defined as a name/label assigned to items/objects that exist in an environment and that have similar properties. It could be person, place, event or even concept, that is, an entity type can be defined for physical as well as not-physical things. An entity type is distinguishable from other entity types on the basis of properties and the same thing provides the basis for the identification of an entity type. We analyze the things existing in any environment or place. We can identify or associate certain properties with each of the existing in that environment. Now the things that have common or similar properties are candidates of belonging to same group, if we assign a name to that group then we say that we have identified an entity type.

Generally, the entity types and their distinguishing properties are established by nature, by very existence of the things. For example, a bulb is an electric accessory, a cricket bat is a sports item, a computer is an electronic device, a shirt is a clothing item etc. So identification of entity types is guided by very nature of the things and then items having properties associated with an entity type are considered to be belonging to that entity type or instances of that entity type. However, many times the grouping of things in an environment is dictated by the specific interest of the organization or system that may supersede the natural classification of entity types. For example, in an organization, entity

types may be identified as donated items, purchased items, manufactured items; then the items of varying nature may belong to these entity types, like air conditioners, tables, frying pan, shoes, car; all these items are quite different from each other by their respective nature, still they may be considered the instances of the same entity type since they are all donated or purchased or manufactured.

What particular properties of an entity type should be considered or which particular properties jointly form an entity type? The answer to this question we have discussed in detail in our very first lecture, where we were discussing the definition of database. That is, the perspective or point of view of the organization and the system for which we are developing the database is going to guide us about the properties of interest for a particular group of things. For example, if you have a look around you in your bedroom, you might see tube light, a bulb, fan, air conditioner, carpet, bed, chair and other things. Now fan is an item that exists in your room, what properties of the fan we are interest in, because there could be so many different properties of the fan. If we are developing the database for a manufacturer, then we may be interested in type of material used for wings, then the thickness of the copper wire in the coil, is it locally manufactured or bought ready made, what individual item costs, what is the labor cost, what is the total cost, overhead, profit margin, net price etc. But if we are working for a shopkeeper he might be interested in the name of the company, dealer price, retail price, weight, color of fan etc. From the user perspective; company name, color, price, warranty, name of the dealer, purchase date and alike. So the perspective helps/guides the designer to associate or identify properties of things in an environment.

The process of identifying entity types, their properties and relationships between them is called abstraction. The abstraction process is also supported by the requirements gathered during initial study phase. For example, the external entities that we use in the DFDs provide us a platform to identify/locate the entity types from. Similarly, if we have created different cross reference matrices, they help us to identify different properties of the things that are of interest in this particular system and that we should the data about. Anyway, entity types are identified through abstraction process, then the items possessing the properties associated with a particular entity type are said to be belonging to that entity type or instances of that entity type.

While designing a system, you will find that most of the entity types are same as are the external entities that you identified for the DFDs. Sometimes they may be exactly the same. Technically, there is a minor difference between the two and that is evident from their definitions. Anything that receives or generates data from or to the system is an external entity, where as entity type is name assigned to a collection of properties of different things existing in an environment. Anything that receives or generates data is considered as external entity and is represented in the DFD, even if it is a single thing. On the other hand, things with a single instance are assumed to be on hand in the environment and they are not explicitly identified as entity type, so they are not represented in the E-R diagram. For example, a librarian is a single instance in a library system, (s)he plays certain role in the library system and at many places data is generated

from or to the librarian, so it will be represented at relevant places in the DFDs. But the librarian will not be explicitly represented in the E-R diagram of the library system and its existence or role is assumed to be there and generally it is hard-coded in the application programs.

Entity Instance
A particular object belonging to a particular entity type and how does an item becomes an instance of or belongs to an entity type? By possessing the defining properties associated with an entity type. For example, following table lists the entity types and their defining properties:

Entity Types	Properties	Instances
EMPLOYEE	Human being, has name, has father name, has a registration number, has qualification, designation	M. Sharif, Sh. Akmal and many others
FURNITURE	Used to sit or work on, different material, having legs, cost, purchased	Chair, table etc.
ELECTRIC APPLIANCES	Need electricity to work, purchased	Bulb, fan, AC
OFFICE EQUIPMENT	Used for office work, consumable or non-consumable,	Papers, pencil, paper weight etc.

Table 1: Entity types, their properties and instances

Each entity instance possesses certain values against the properties with the entity type to which it belongs. For example, in the above table we have identified that entity type EMPLOYEE has name, father name, registration number, qualification, designation. Now an instance of this entity type will have values against each of these properties, like (M. Sajjad, Abdul Rehman, EN-14289, BCS, and Programmer) may be one instance of entity type EMPLOYEE. There could be many others.

Entity Set
A group of entity instances of a particular entity type is called an entity set. For example, all employees of an organization form an entity set. Like all students, all courses, all of them form entity set of different entity types

As has been mentioned before that the term entity is used for all of the three terms mentioned above, and it is not wrong. Most of the time it is used to mention an entity type, next it is used for an entity instance and least times for entity set. We will be precise most of the time, but if otherwise you can judge the particular meaning from the context.

Classification of entity types

Entity types (ETs) can be classified into regular ETs or weak ETs. Regular ETs are also called strong or independent ETs, whereas weak ETs are also called dependent ETs. In the following we discuss them in detail.

Weak Entity Types
An entity type whose instances cannot exist without being linked with instances of some other entity type, i.e., they cannot exist independently. For example, in an organization we want to maintain data about the vehicles owned by the employees. Now a particular vehicle can exist in this organization only if the owner already exists there as employee. Similarly, if employee leaves the job and the organization decides to delete the record of the employee then the record of the vehicle will also be deleted since it cannot exist without being linked to an instance of employee.

Strong Entity Type
An entity type whose instances can exist independently, that is, without being linked to the instances of any other entity type is called strong entity type. A major property of the strong entity types is that they have their own identification, which is not always the case with weak entity types. For example, employee in the previous example is an independent or strong entity type, since its instances can exist independently.

Naming Entity Types
Following are some recommendations for naming entity types. But they are just recommendations; practices considered good in general. If one, some or all of them are ignored in a design, the design will still be valid if it satisfies the requirements otherwise, but good designs usually follow these practices:

- Singular noun recommended, but still plurals can also be used
- Organization specific names, like customer, client, gahak anything will work
- Write in capitals, yes, this is something that is generally followed, otherwise will also work.
- Abbreviations can be used, be consistent. Avoid using confusing abbreviations, if they are confusing for others today, tomorrow they will confuse you too.

Symbols for Entity Types
A rectangle is used to represent an entity type in E-R data model. For strong entity types rectangle with a single line is used whereas double lined rectangle is drawn to represent a weak entity type as is shown below:

© Virtual University of Pakistan

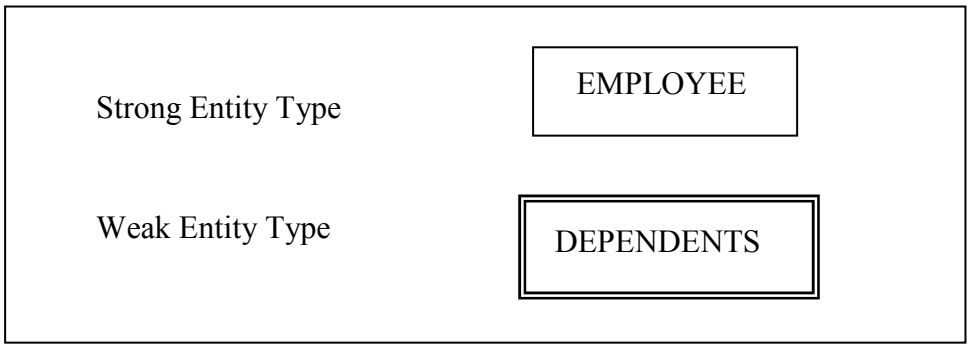


Figure 1: Symbols used for entity types

We have discussed different types of entity types; in the next section we are going to discuss another component or E-R data model, that is, the attribute.

Attribute

An attribute of an entity type is a defining property or quality of the instances of that entity type. Entity instances of same entity type have the same attributes. (E.g. Student Identification, Student Name). However, values of these attributes may be same or different. For example, all instances of the entity type STUDENT may have the attributes name, father name, age; but the values against each of these attributes for each instance may be different. Like, one instance may have the values (M. Hafeez, Noor Muhammad, 37) other may have others. Remember one thing, that the values of the attributes may be same among different entity instances. The thing to remember at this stage is that attributes are associated with an entity type and those attributes then become applicable /valid for all the instances of that entity type and instances have values against these attributes.

An attribute is identified by a name allocated to it and that has to be unique with respect to that entity type. It means one entity type cannot have two attributes with the same name. However, different entity types may have attributes with the same name. The guidelines for naming an attribute are similar to those of entity types. However, one difference is regarding writing the names of attributes. The notation that has been adopted in this course is that attribute name generally consists of two parts. The name is started in lower case, and usually consists of abbreviation of the entity types to which the attribute belongs. Second part of the attribute name describes the purpose of attribute and only first letter is capitalized. For example empName means name attribute of entity type EMPLOYEE, stAdrs means address attribute of the entity type STUDENT and alike. Others follow other notations, there is no restriction as such, and you can follow anyone that you feel convenient with. BUT be consistent.

Domain of an Attribute

We have discussed in the previous section that every attribute has got a name. Next thing is that a domain is also associated with an attribute. These two things, name and the domain, are part of the definitions of an attribute and we must provide them. Domain is the set of possible values that an attribute can have, that is, we specify a set of values either in the form of a range or some discrete values, and then attribute can have value out of those values. Domain is a form of a check or a constraint on attribute that it cannot have a value outside this set.

Associating domain with an attribute helps in maintaining the integrity of the database, since only legal values could be assigned to an attribute. Legal values mean the values that an attribute can have in an environment or system. For example, if we define a salary attribute of EMPLOYEE entity type to hold the salary of employees, the value assigned to this attribute should be numeric, it should not be assigned a value like 'Reema', or '10/10/2004', why, because they are not legal salary values[1]. It should be numeric. Further, even if we have declared it as numeric it will have numeric values, but about a value like 10000000000. This is a numeric value, but is it a legal salary value within an organization? You have to ask them. It means not only you will specify that the value of salary will be numeric but also associate a range, a lower and upper limit. It reduces the chances of mistake.

Domain is normally defined in form of data type and some additional constraints like the range constraint. Data type is defined as a set of values along with the operations that can be performed on those values. Some common data types are Integer, Float, Varchar, Char, String, etc. So domain associates certain possible values with an attribute and certain operations that can be performed on the values of the attribute. Another important thing that needs to be mentioned here is that once we associate a domain to an attribute, all the attributes in all entity instances of that entity type will have the values from the same domain. For example, it is not possible that in one entity instance the attribute salary has a value 15325.45 and in another instance the same attribute has a value 'Reema'. No. All attribute will have values from same domain, values may be different or same, whatever, but the domain will be the same.

[1] Sometimes when some coding has been adopted, then such strange values may be legal but here we are discussing the general conditions

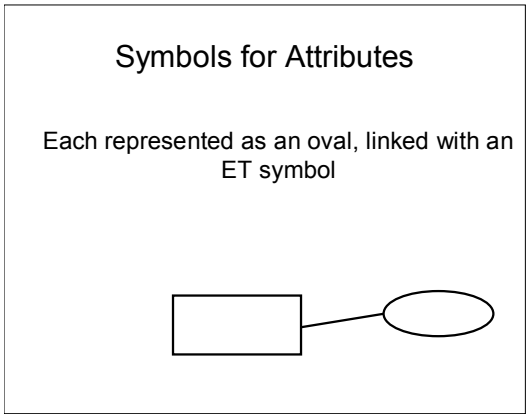


Figure 2: Symbol used for attribute in E-R diagram

Types of Attributes

Attributes may be of different types. They may be:

- Simple or Composite
- Single valued or multi-valued
- Stored or Derived

Simple or Composite Attributes:
An attribute that is a single whole is a simple attribute. The value of a simple attribute is considered as a whole, not as comprising of other attributes or components. For example, attributes stName, stFatherName, stDateOfBorth of an entity type STUDENT are example of simple attributes. On the other hand if an attribute consists of collection of other simple or composite attributes then it is called a composite attributes. For example, stAdres attribute may comprise of houseNo, streetNo, areaCode, city etc. In this case stAdres will be a composite attribute.

Single valued or multi-valued Attributes:
Some attribute have single value at a time, whereas some others may have multiple values. For example, hobby attribute of STUDENT or skills attribute of EMPLOYEE, since a student may have multiple hobbies, likewise an employee may have multiple skills so they are multi-valued attributes. On the other hand, name, father name, designation are generally single valued attributes.

Stored or Derived Attributes:
Normally attributes are stored attributes, that is, their values are stored and accessed as such from the database. However, sometimes attributes' values are not stored as such, rather they are computed or derived based on some other value. This other value may be stored in the database or obtained some other way. For example, we may store the name, father name, address of employees, but age can be computed from date of birth. The

advantage of declaring age as derived attribute is that whenever we will access the age, we will get the accurate, current age of employee since it will be computed right at the time when it is being accessed.

How a particular attribute is stored or defined, it is decided first by the environment and then it has to be designer's decision; your decision. Because, the organization or system will not object rather they will not even know the form in which you have defined an attribute. You have to make sure that the system works properly, it fulfills the requirement; after that you do it as per your convenience and in an efficient way.

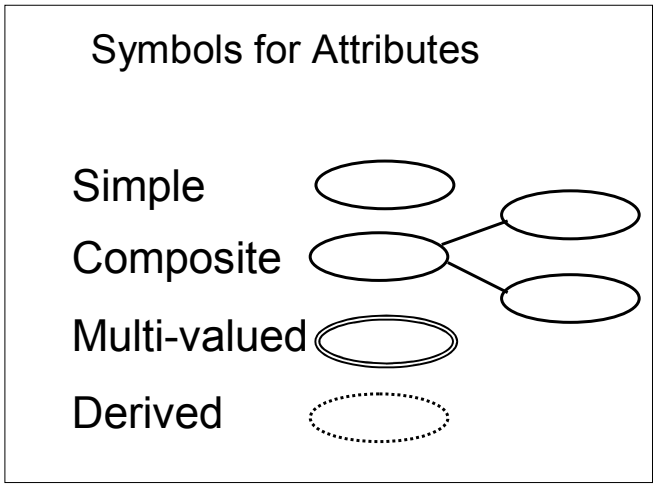


Figure 3: Symbol used for different types of attributes in E-R diagram

An example diagram representing all types of attributes is given below:

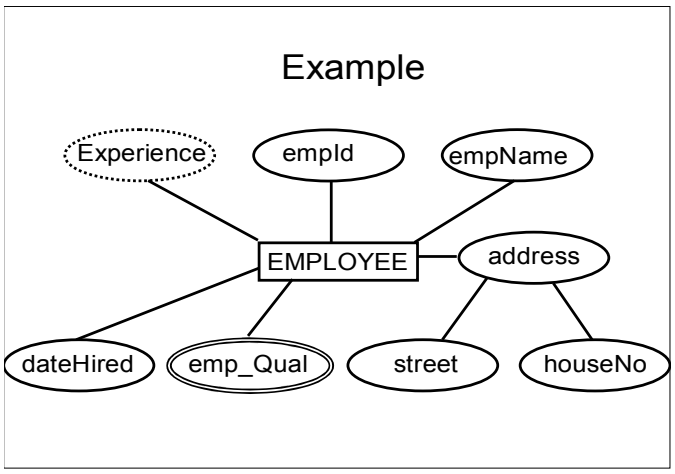


Figure 4: Example entity type with attributes of different types
This concludes this lecture.

Summary:

In this lecture we have discussed entity and attribute. We discussed that there are three different notions for which the term entity is used and we looked into these three terms in detail. They are entity type, entity instance and entity set. An entity type is name or label assigned to items or objects existing in an environment and having same or similar property. An entity instance is a particular item or instance that belongs to a particular entity type and a collection of entity instances is called an entity set. We also discussed in this lecture the attribute component of the E-R data model and its different types. The third component the E-R data model, that is, the relationship will be discussed in the next lecture.

Exercises:

- Take a look into the system where you work or study or live, identify different entity types in that environment. Associate different types of attributes with these entity types.
- Look at the same environment from different possible perspectives and realize the difference that the change of perspective makes in the abstraction process that results in establishing different entity types or/and their different properties.

Lecture No. 08

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section 5.4

Overview of Lecture

- Concept of Key and its importance
- Different types of keys

Attributes

Def 1:

An attribute is any detail that serves to identify, qualify, classify, quantify, or otherwise express the state of an entity occurrence or a relationship.

Def 2:

Attributes are data objects that either identify or describe entities.

Identifying entity type and then assigning attributes or other way round; it's an "egg or hen" first problem. It works both ways; differently for different people. It is possible that we first identify an entity type, and then we describe it in real terms, or through its attributes keeping in view the requirements of different users' groups. Or, it could be other way round; we enlist the attribute included in different users' requirements and then group different attributes to establish entity types. Attributes are specific pieces of information, which need to be known or held. An attribute is either required or optional. When it's required, we must have a value for it, a value must be known for each entity occurrence. When it's optional, we could have a value for it, a value may be known for each entity occurrence.

The Keys

Attributes act as differentiating agents among different entity types, that is, the differences between entity types must be expressed in terms of attributes. An entity type can have many instances; each instance has got a certain value against each attribute defined as part of that particular entity type. A *key* is a set of attributes that can be used to

identify or access a particular entity instance of an entity type (or out of an entity set). The concept of key is beautiful and very useful; why and how. An entity type may have many instances, from a few to several thousands and even more. Now out of many instances, when and if we want to pick a particular/single instance, and many times we do need it, then key is the solution. For example, think of whole population of Pakistan, the data of all Pakistanis lying at one place, say with NADRA people. Now if at sometime we need to identify a particular person out of all this data, how can we do that? Can we use name for that, well think of any name, like Mirza Zahir Iman Afroz, now we may find many people with this name in Pakistan. Another option is the combination of name and father name, then again, Amjad Malik s/o Mirza Zahir Iman Afroz, there could be so many such pairs. There could be many such examples. However, if you think about National ID Card number, then no matter whatever is the population of Pakistan, you will always be able to pick precisely a single person. That is the key. While defining an entity type we also generally define the key of that entity type. How do we select the key, from the study of the real-world system; key attribute(s) already exist there, sometimes they don't then the designer has to define one. A key can be simple, that is, consisting of single attribute, or it could be composite which consists of two or more attributes. Following are the major types of keys:

- Super Key
- Candidate Key
- Primary Key
- Alternate Key
- Secondary Key
- Foreign Key

The last one will be discussed later, remaining 5 are discussed in the following:

- **Super key**
  A **super key** is a set of one or more attributes which taken collectively, allow us to identify uniquely an entity instance in the entity set. This definition is same as of a key, it means that the super key is the most general type of key. For example, consider the entity type STUDENT with attributes registration number, name, father name, address, phone, class, admission date. Now which attribute can we use that can uniquely identify any instance of STUDENT entity type. Of course, none of the name, father name, address, phone number, class, admission date can be used for this purpose. Why? Because if we consider name as super key, and situation arises that we need to contact the parents of a particular student. Now if we say to our registration department that give us the phone number of the student whose name is Ilyas Hussain, the registration department conducts a search and comes up with 10 different Ilyas Hussain, could be anyone. So the value of the name attribute cannot be used to pick a particular instance. Same happens with other attributes. However, if we use the registration number, then it is 100% sure that with a particular value of registration number we will always find exactly a single unique entity instance. Once you identified the instance, you have all its attributes available, name, father name,

everything. The entity type STUDENT and its attributes are shown graphically in the figure 1 below, with its super key "regNo" underlined.

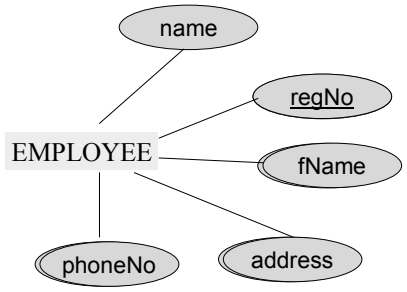


Fig. 1: An entity type, its defining attributes and super key (underlined)

Once specific characteristic with super key is that, as per its definition any combination of attributes with the super key is also a super key. Like, in the example just discussed where we have identified regNo as super key, now if we consider any combination of regNo with any other attribute of STUDENT entity type, the combination will also be a super key. For example, "regNo, name", "regNo, fName, address", "name, fName, regNo" and many others, all are super keys.

- **Candidate key**

  A super key for which no subset is a super key is called a candidate key, or the minimal super key is the candidate key. It means that there are two conditions for the candidate key, one, it identifies the entity instances uniquely, as is required in case of super key, second, it should be minimum, that is, no proper subset of candidate key is a key. So if we have a simple super key, that is, that consists of single attribute, it is definitely a candidate key, 100%. However, if we have a composite super key and if we take any attribute out of it and remaining part is not a super key anymore then that composite super key is also a candidate key since it is minimal super key. For example, one of the super keys that we identified from the entity type STUDENT of figure 1 is "regNo, name", this super key is not a candidate key, since if we remove the regNo attribute from this combination, name attribute alone is not able to identify the entity instances uniquely, so it does not satisfy the first condition of candidate key. On the other hand if we remove the attribute name from this composite key then the regNo alone is sufficient to identify the instances uniquely, so "regNo, name" does have a proper subset (regNo) that can act as a super key; violation of second condition. So the composite key "regNo, name" is a super key but it is not a candidate key. From here we can also establish a fact that every candidate key is a super key but not the other way round.

- **Primary Key**

A candidate key chosen by the database designer to act as key is the primary key. An entity type may have more than one candidate keys, in that case the database designer has

to designate one of them as primary key, since there is always only a single primary key in an entity type. If there is just one candidate key then obviously the same will be declared as primary key. The primary key can also be defined as the successful candidate key. Figure 2 below contains the entity type STUDENT of figure 1 but with an additional attribute nIdNumber (national ID card Number).

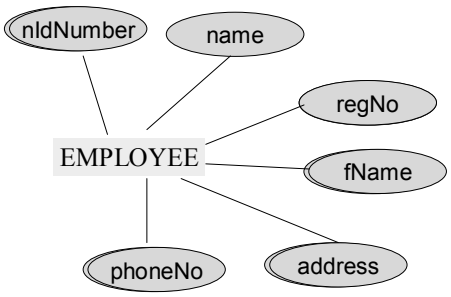


Fig. 2: An entity type, its defining attributes and two candidate keys

In figure 2, we can identify two different attributes that can individually identify the entity instances of STUDENT and they are regNo and nIdNumber, both are minimal super keys so both are candidate keys. Now in this situation we have got two candidate keys. The one that we choose will be declared as primary key, other will be the alternate key. Any of the candidate keys can be selected as primary key, it mainly depends on the database designer which choice he/she makes. There are certain things that are generally considered while making this decision, like the candidate key that is shorter, easier to remember, to type and is more meaningful is selected as primary key. These are general recommendations in this regard, but finally it is the decision of the designer and he/she may have his/her own reasons for a particular selection that may be entirely different from those mentioned above. The relation that holds between super and candidate keys also holds between candidate and primary keys, that is, every primary key (PK) is a candidate key and every candidate key is a super key.

A certain value that may be associated with any attribute is NULL, that means "not given" or "not defined". A major characteristic of the PK is that it cannot have the NULL value. If PK is a composite, then none of the attributes included in the PK can have the NULL, for example, if we are using "name, fName" as PK of entity type STUDENT, then none of the instances may have NULL value in either of the name or fName or both.

- **Alternate Keys**
  Candidate keys which are not chosen as the primary key are known as alternate keys. For example, we have two candidate keys of STUDENT in figure 2, *regNo* and *nIdNumber*, if we select *regNo* as PK then the *nIdNumber* will be alternate key.
- **Secondary Key**

Many times we need to access certain instances of an entity type using the value(s) of one or more attributes other than the PK. The difference in accessing instances using the

value of a key or non-key attribute is that the search on the value of PK will always return a single instance (if it exists), where as uniqueness is not guaranteed in case of non-key attribute. Such attributes on which we need to access the instances of an entity type that may not necessarily return unique instance is called the secondary key. For example, we want to see how many of our students belong to Multan, in that case we will access those instances of the STUDENT entity type that contain "Multan" in their address. In this case address will be called secondary key, since we are accessing instances on the basis of its value, and there is no compulsion that we will get a single instance. Keep one thing in mind here, that a particular access on the value of a secondary key MAY return a single instance, but that will be considered as chance or due to that particular state of entity set. There is not the compulsion or it is not necessary for secondary key to return unique instance, where as in case of super, candidate, primary and alternate keys it is compulsion that they will always return unique instance against a particular value.

Summary

Keys are fundamental to the concept almost any data model including the E-R data model because they enable the unique identity of an entity instance. There are different type of keys that may exist in an entity type.

Exercises:

- Define attributes of the entity types CAR, BOOK, MOVIE; draw them graphically
- Identify different types of keys in each one of them

Lecture No. 09

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	

Overview of Lecture

- Relationships in E-R Data Model
- Types of Relationships

Relationships

After two or more entities are identified and defined with attributes, the participants determine if a *relationship* exists between the entities. A relationship is any association, linkage, or connection between the entities of interest to the business; it is a two-directional, significant association between two entities, or between an entity and itself. Each relationship has a name, an optionality (*optional* or *mandatory*), and a degree (how many). A relationship is described in real terms.

Assigning a name, optionality, and a degree to a relationship helps confirm the validity of that relationship. If you cannot give a relationship all these things, then perhaps there really is no relationship at all.

Relationship represents an association between two or more entities. An example of a relationship would be:

- Employees are assigned to projects
- Projects have subtasks
- Departments manage one or more projects

Relationships are the *connections and interactions* between the entities instances e.g. DEPT_EMP associates Department and Employee.

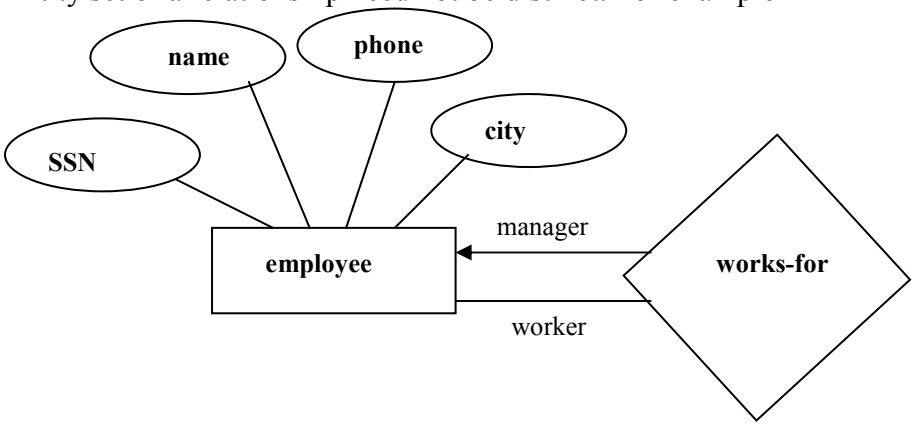
- A ***relationship type*** is an abstraction of a relationship i.e. a set of relationships instances sharing common attributes.

- Entities enrolled in a relationship are called its ***participants***.

The participation of an entity in a relationship is ***total*** when all entities of that set might be participant in the relationship otherwise it is ***partial*** e.g. if every ***Part*** is supplied by a ***Supplier*** then the SUPP_PART relationship is total. If certain parts are available without a supplier than it is partial.

Naming Relationships:
If there is no proper name of the association in the system then participants' names of abbreviations are used. STUDENT and CLASS have ENROLL relationship. However, it can also be named as STD_CLS.

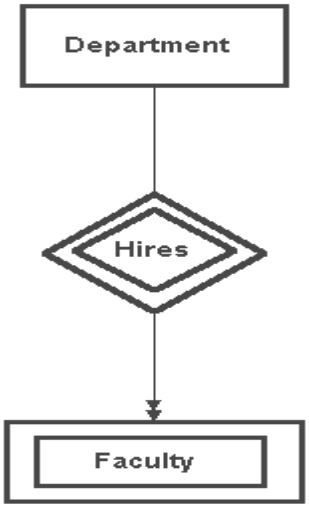
Roles:
Entity set of a relationship need not be distinct. For example



The labels "manager" and "worker" are called "roles". They specify how employee entities interact via the "works-for" relationship set. Roles are indicated in ER diagrams by labeling the lines that connect diamonds to rectangles. Roles are optional. They clarify semantics of a relationship.

Symbol for Relationships:

- Shown as a Diamond
- Diamond is doubled if one of the participant is dependent on the other
- Participants are connected by continuous lines, labeled to indicate cardinality.
- In partial relationships roles (if identifiable) are written on the line connecting the partially participating entity rectangle to the relationship diamond.
- Total participation is indicated by double lines

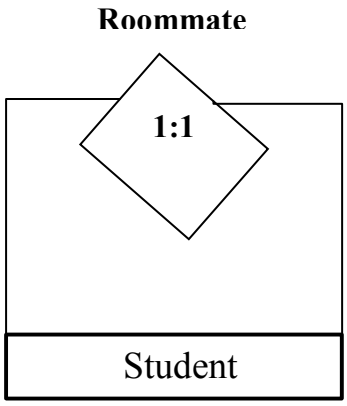


Types of Relationships

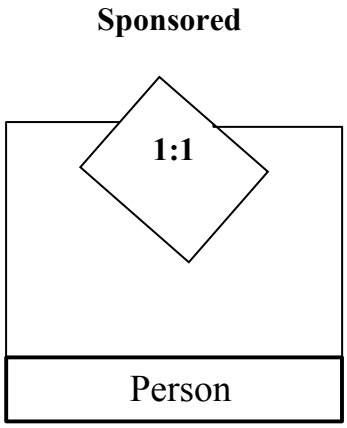
- **Unary Relationship**

  An ENTITY TYPE linked with itself, also called recursive relationship. Example Roommate, where STUDENT is linked with STUDENT

  Example 1:



Example 2:



- **Binary relationship**

  A **Binary** relationship is the one that links two entities sets e.g. **STUDENT-CLASS.** Relationships can be formally described in an ordered pair form.

  Enroll = {(S1001, ART103A), (S1020, CS201A), (S1002, CSC201A)}

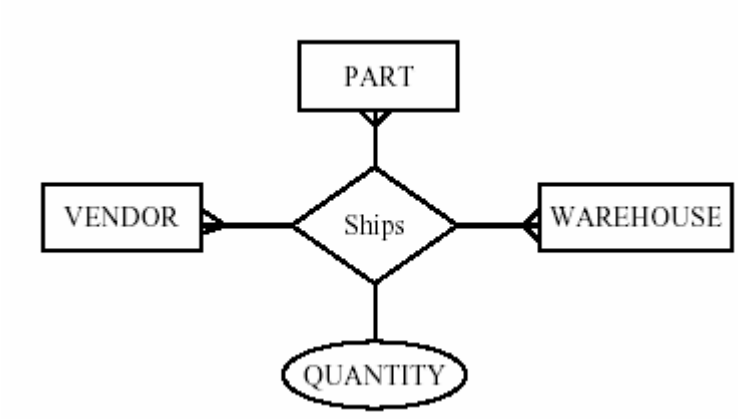
  Entire set is relationship set and each ordered pair is an instance of the relationship.



- **Ternary Relationship**

  A **Ternary** relationship is the one that involves three entities e.g.

  STUDENT-CLASS-FACULTY.



- **N-ary Relationship**

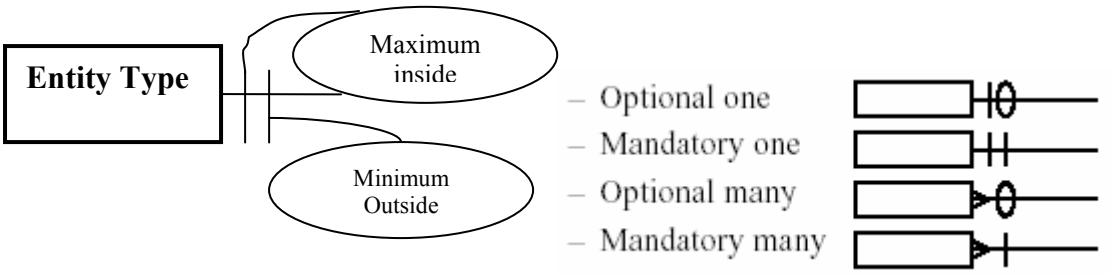
  Most relationships in data model are binary or at most ternary but we could define a relationship set linking any number of entity sets i.e. **n-ary** relationship

Entity sets involved in a relationship set need not be distinct. E.g.

Roommate = {(Student1, Student2) | Student1 ∈ Student Entity Set, Student2 ∈ Student Entity Set and Student 1 is the Roommate of Student2}

Relationship Cardinalities

The cardinality of a relationship is the number of entities to which another entity can map under that relationship. Symbols for maximum and minimum cardinalities are:

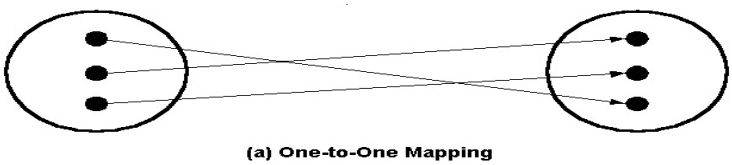


- **One-to-One mapping:**

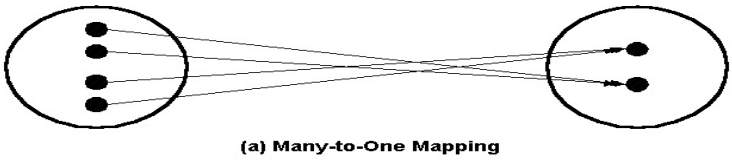
  A mapping R from X to Y is one-to-one if each entity in X is associated with at most one entity in Y and vice versa.

- **Many-to-One mapping:**

  A mapping R from X to Y is many-to-one if each entity in X is associated with at most one entity in Y but each entity in Y is associated with many entities in X.



(a) One-to-One Mapping



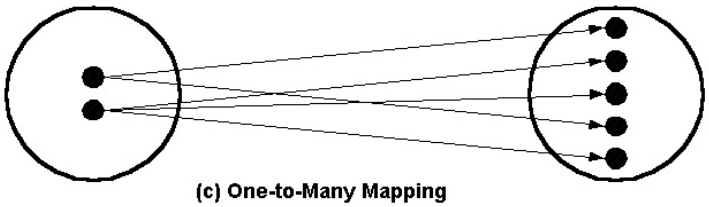
(a) Many-to-One Mapping

- **One-to-Many mapping:**

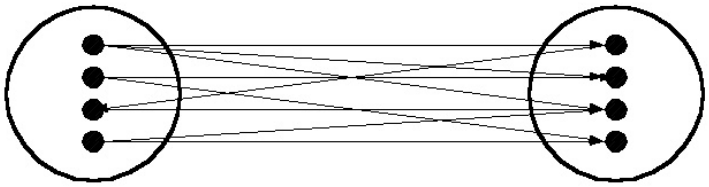
  A mapping R from X to Y is one-to-many if each entity in X is associated with many entities in Y but each entity in Y is associated with one entity in X.

- **Many-to-Many mapping:**

  A mapping R from X to Y is many-to-many if each entity from X is associated with many entities in Y and one entity in Y is associated with many entities in X.



(c) One-to-Many Mapping



(d) Many-to-Many Mapping

Lecture No. 10

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Page: 155 – 160
Hoffer	Page: 103 – 111

Overview of Lecture

- Cardinality Types
- Roles in ER Data Model
- Expression of Relationship in ER Data Model
- Dependency
- Existence Dependency
- Referential Dependency
- Enhancements in the ER-Data Model
- Subtype and Supertype entities

Recalling from the previous lecture we can say that that cardinality is just an expression which tells us about the number of instances of one entity which can be present in the second relation. Maximum cardinality tells us that how many instance of an entity can be placed in the second relation at most. Now we move onto discuss that what the minimum cardinality is.

Minimum Cardinality:
As the name suggests that the minimum cardinality is the inverse of the maximum cardinality so we can say that the minimum cardinality show us that how many instance of one entity can be placed in another relation at least. In simple words it can be said that the minimum cardinality tells that whether the link between two relations is optional or compulsory. It is very important to determine the minimum cardinality when designing a database because it defines the way a database system will be implemented.

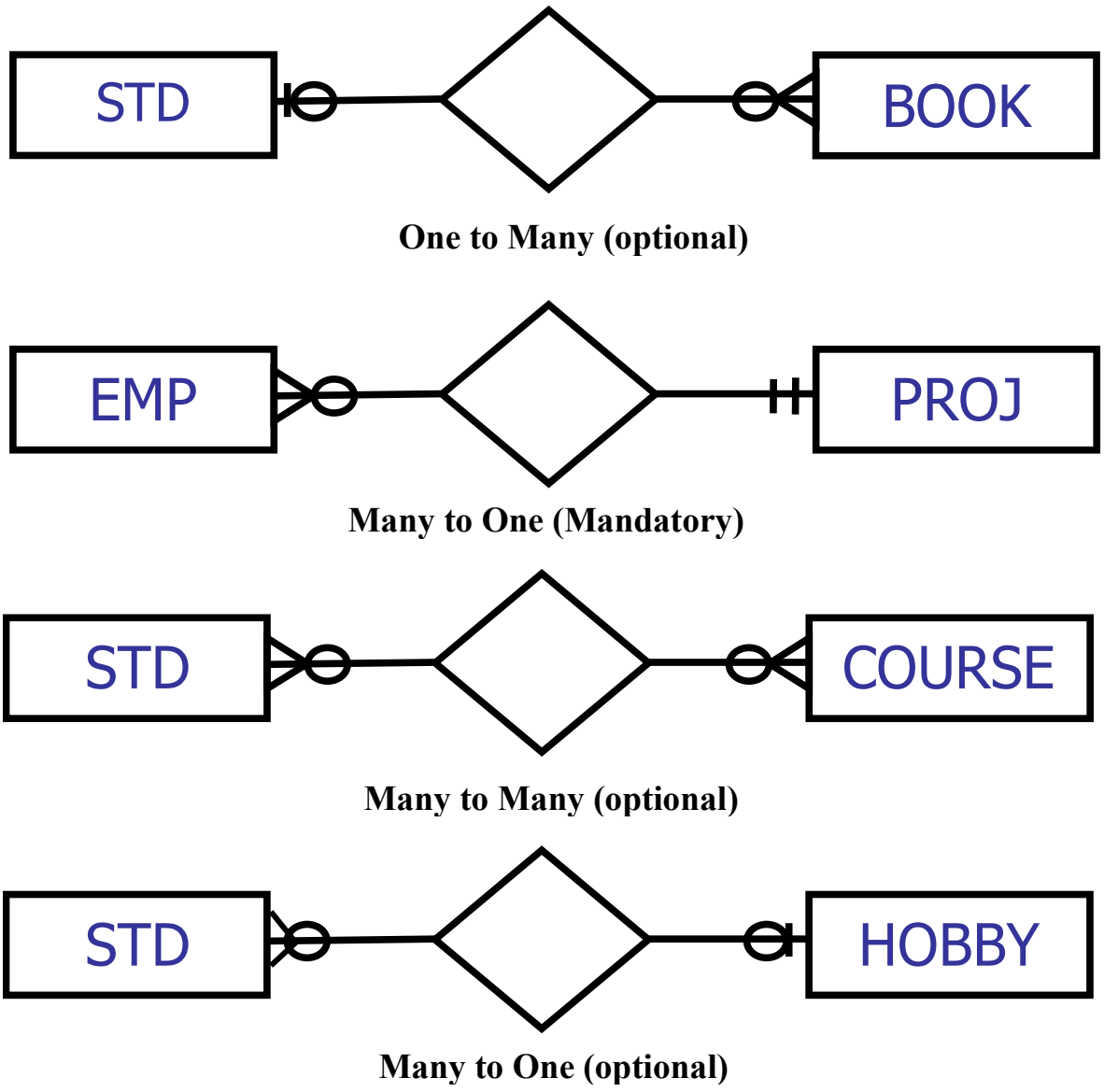


Fig 1: Different Cardinalities

In the figure-1 we have one to many cardinality between the entities. Maximum cardinalities are shown with the modifier that appears on the link and is adjacent to the entity rectangle. The other modifier which is next to the maximum cardinality modifier tells the minimum cardinality. The minimum cardinality modifier lies at more distance from the entity as compared to the maximum cardinality modifier.

Determination of the cardinalities is done by interviewing the users of the system and by the analysis of the organization.

The cardinality shown in First Part of the Figure-1 is shown using a relationship between a student and book; this can be a library scenario where students are borrowing books from the library. We can see in the diagram the shape ⟜ adjacent to the student entity it shows that the minimum cardinality for the student relationship is zero and maximum cardinality is one. Where as on the other side of the diagram the shape ⟜ adjacent to

the book entity show that at most there can be many instances of the book associated with a single instance of student entity, and that there can be at-least no instance associated with the student entity. In general library scenario we can say that one student can borrow at least no and at most many books. Hence the minimum and maximum cardinality is shown.

In the second part of the Figure-1 we see a relationship between the Employee and project entities, the relationship describes one to many association between the project and the employees, It shows that there can be one project having a number of employees, but for the existence of one employee at one project is necessary. So the minimum and maximum cardinality on the project side of the relationship is one, and employees associated with each project can be many at most and none at-least.

Third part of the Figure-1 shows the association between the student and the course entities. Here we can see that the relationship between the student and the course is zero at least and many at most on both the sides of the relationship. The minimum cardinality with zero minimum is also called the optional cardinality. It also shows that one student can have registered more that one subjects and one subject can also be taken by many students. Also it is not necessary for a student to get registered one subject.

In the fourth part of the Figure-1 we can see the one to many cardinality between the student and hobby entities the cardinality descriptors show that a student may have no or at most one hobby, but it is worthwhile to notice that the cardinality of the hobby with the student in many but optional, now we can say that one hobby can be associated to nay student but there is a chance that no hobby is associated to one student at a certain time.

Other Notations:

The notation mentioned above is known as crow's foot notation for the expression of ER-Diagrams, there can be other notation as well which can be used for creating ER-Diagrams; one of these notations is shown in the Figure-2. We can see that the one to many cardinality shown in the first part of the diagram is expresses with single and double arrows. The Single arrow in this case shows the one and double arrow show the many cardinality.

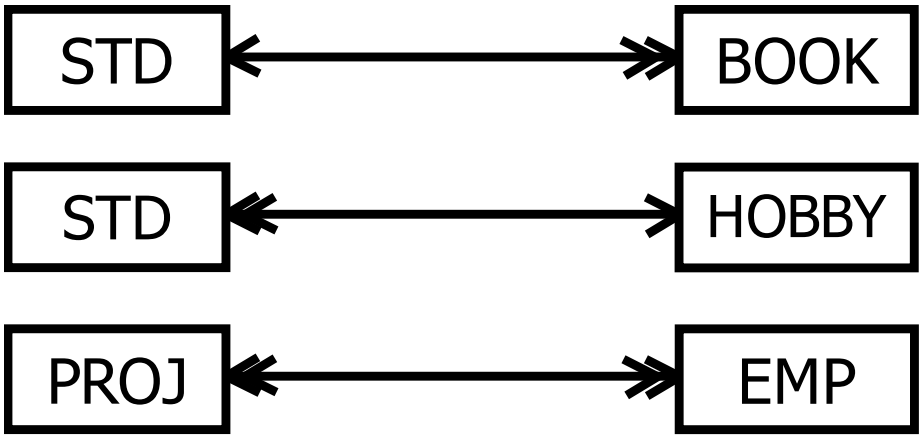


Fig. 2: Arrow-head notation

So the First part of the figure-2 show One to many cardinality, second part of the Figure shows many to one and the third part of the cardinality shows many to many cardinality between the entities involved.

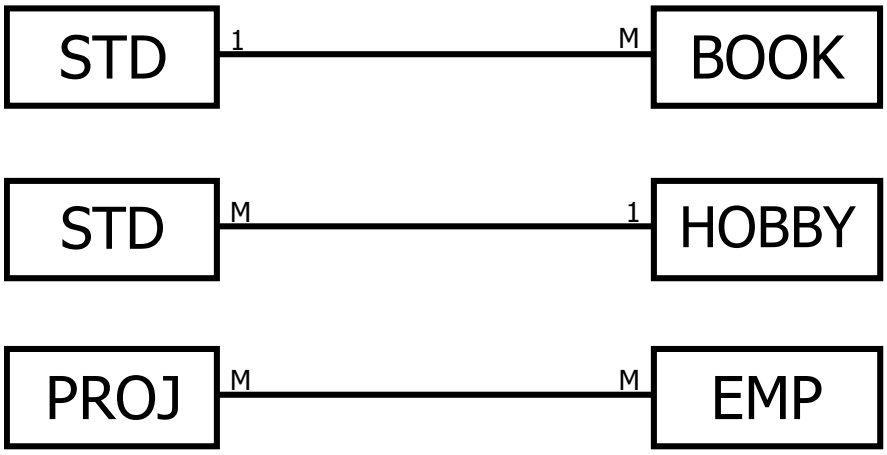


Fig. 3: Alphabetical notation

The above Figure shows another notation for creating ER-Diagrams which show that to show the one cardinality we have used 1 and for many cardinality M or N is used.

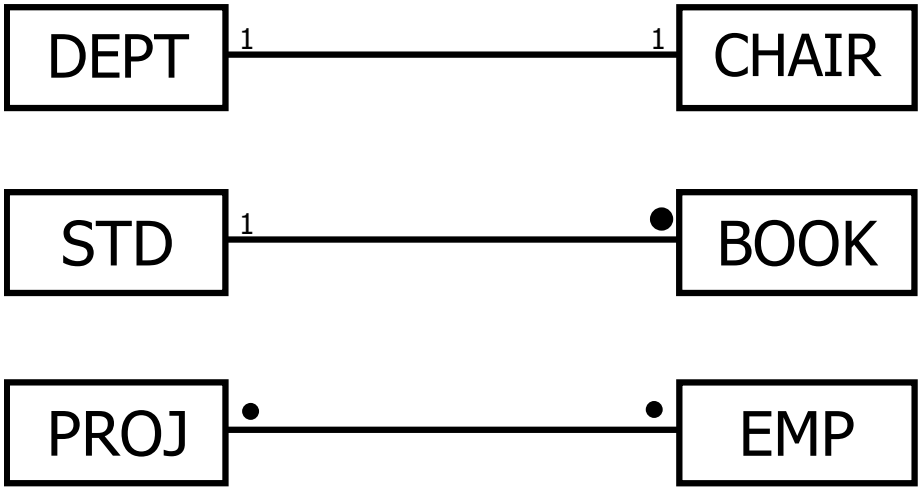


Fig. 4: Dot-based notation

Notations shown in the Figure-4 above as also used for creating ER-Diagrams where 1 is used for showing the single cardinality and the black filled Dot is used for showing many cardinality.

Roles in Relationships

The way an entity is involved in a relationship is called the role of the entity in the relationship. These details provide more semantics of the database. The role is generally clear from the relationship, but in some cases it is necessary to mention the role explicitly.
Two situations to mention the role explicitly

Recursive Relationship:
This is the situation when any attribute of one entity is associated with another attribute of the same entity. Such a link initiates from one entity and terminates on the same entity.

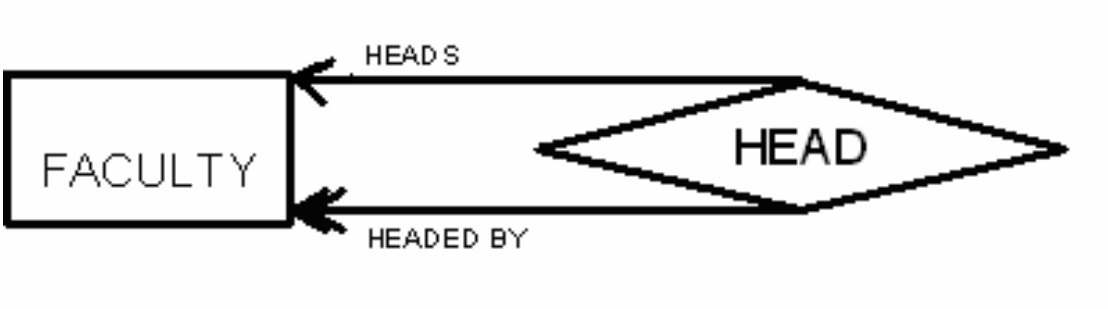


Fig-5: Roles in a unary relationship

Figure-5 above shows the recursive relationship which tells that in the faculty of a certain institute we can have one faculty member from among the same faculty as the head of the faculty. Now the role mentioned on the relationship tell that many Faculty instance are headed by one of the entity instance from the same faculty relation.

Multiple Relationships:
This is the second situation which needs the role to be mentioned on the relationship link when there is more than one relationship.

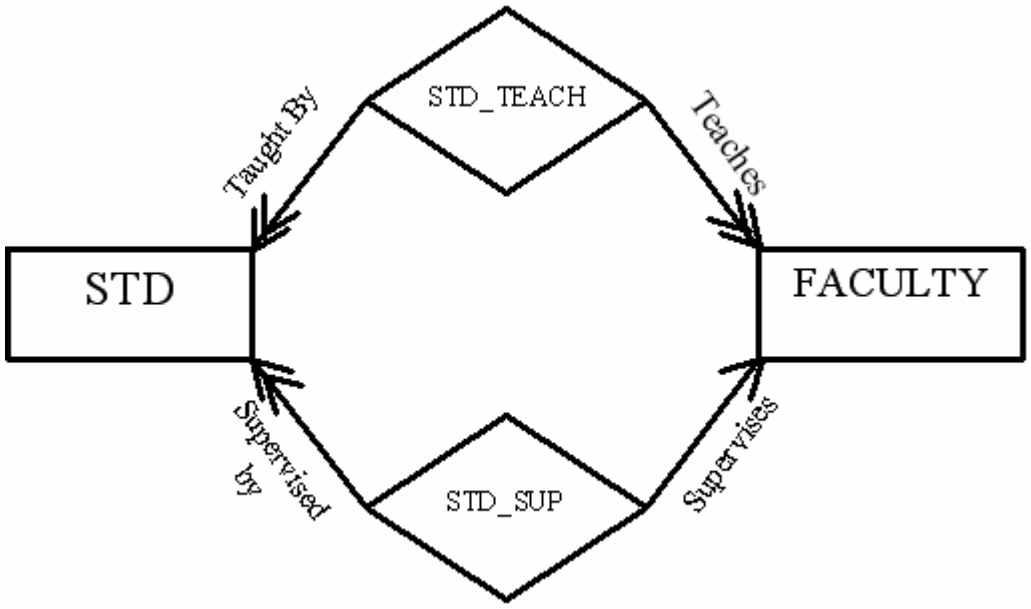


Fig. 6: Multiple relationships

As an example we can have a relationship of Faculty members and students as one faculty member may teach a number of students and at the same time one student may have been taught by a number of faculty members. This is one side of the picture. Now on the other side we can say that a faculty member may be supervising a number of students for their final projects. It shows two types of associations between the faculty and the students. So in this type of situation it is necessary to mention the role of the entities involved in the relationship.

Dependencies

Dependency is a type of constraint, for example once we define the cardinality or relationship among two entities it also is a constraint or check that tells that cardinality should be followed while populating data in relations. Similarly the dependency is a constraint. There are a number of dependency types which are expressed below:

The Existence dependency:
This is the type of dependency which exists when one entity instance needs instance of another entity for its existence. As we have seen earlier in case of employee of and organization and the projects associated with the employees there we see that employees are dependent on projects, it means that if no project is assigned to an employee it can not exist. In other words we can say that at a certain time an employee must be working on at least one project.

Identifier Dependency:
It means that the dependent entity incase of existence dependency does not have its own identifier and any external identifier is used to pick data for that entity. And to define a key in this entity the key of the parent entity is to be used in the key for this entity may be used as composite keys.

Referential Dependency:
This is the situation when the dependent entity has it own key for unique identification but the key used to show the reference with the parent entity is shown with the help of an attribute of the parent entity. Means to show the link of the parent entity with this entity there will be an attribute and a record in this entity will not exist without having a record in the parent entity. Despite of having its own identifier attribute.
This type of identifier or attribute in the weak entity is known as foreign key.

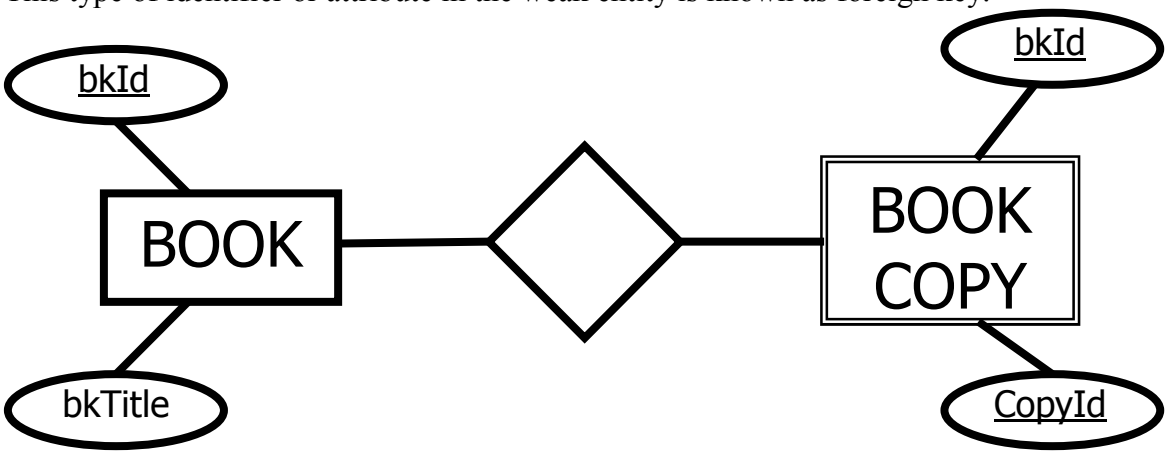


Fig-7

In the Figure-7 above the relation shown is expression the existence dependency where it is necessary for a book instance to exist if there exist the copies of the book with the same bkId.

Enhancements in E-R Data Model:

The topics that we have discussed so for constitute the basics of ER-Model. The model is further extended and strengthened with addition of some new concepts and modeling constructs, which are discussed below

Super-type and Subtypes

These are also relationships existing between entities, also referred to as generalized and specialized respectively let us examine the figure below to grasp the idea of super-type and subtype.

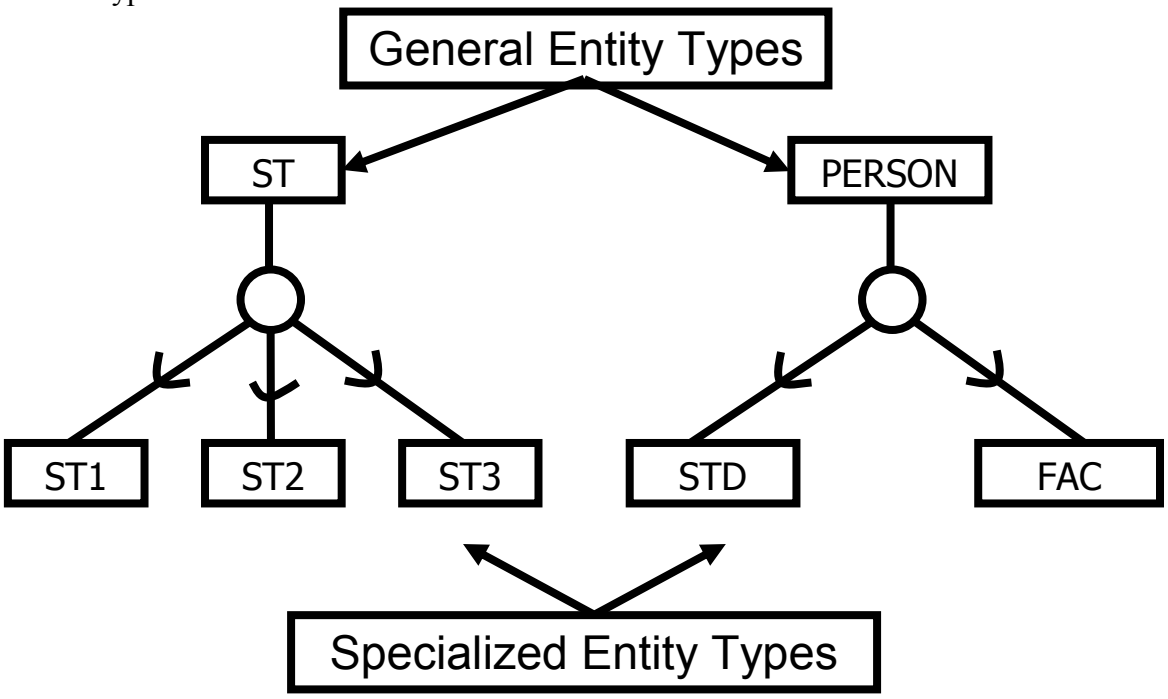


Fig-8 (Super-types and Subtypes)

In the Figure:8 show above there are different levels of existence of entities, at the top level we have general entity type, which are described as having a number of Subtype entities, these sub entities are in-turn acting as supertypes entities for a number of other entities. As we see in case of person supertype we can have further classify the person entity as Student (STD) and Teacher of Faculty member (FAC). Subtype entities are expressed with a link to the supertypes having an arc on the link—the arms of which

point to the supertype entity. As we move downward the distributed entities are known as specialized entities.

In the next Lecture the process of Generalization and Specialization will be discussed in detail.

Summary:

In this lecture we have discussed an important topic of cardinalities and their representation in the E-R data model. For a correct design the correct identification of cardinalities is important.

Lecture No. 11

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	

Overview of Lecture

- Inheritance
- Super type
- Subtypes
- Constraints
- Completeness
- Disjointness
- Subtype Discrimination

According to the Microsoft Dictionary of Computing

Inheritance Is

The transfer of the characteristics of a class in object-oriented programming to other classes derived from it. For example, if "vegetable" is a class, the classes "legume" and "root" can be derived from it, and each will inherit the properties of the "vegetable" class: name, growing season, and so on2. Transfer of certain properties such as open files, from a parent program or process to another program or process that the parent causes to run. Inheritance in the paradigm of database systems we mean the transfer of properties of one entity to some derived entities, which have been derived from the same entities.

Super types and Subtypes

Subtypes hold all the properties of their corresponding super-types. Means all those subtypes which are connected to a specific supertype will have all the properties of their supertype.

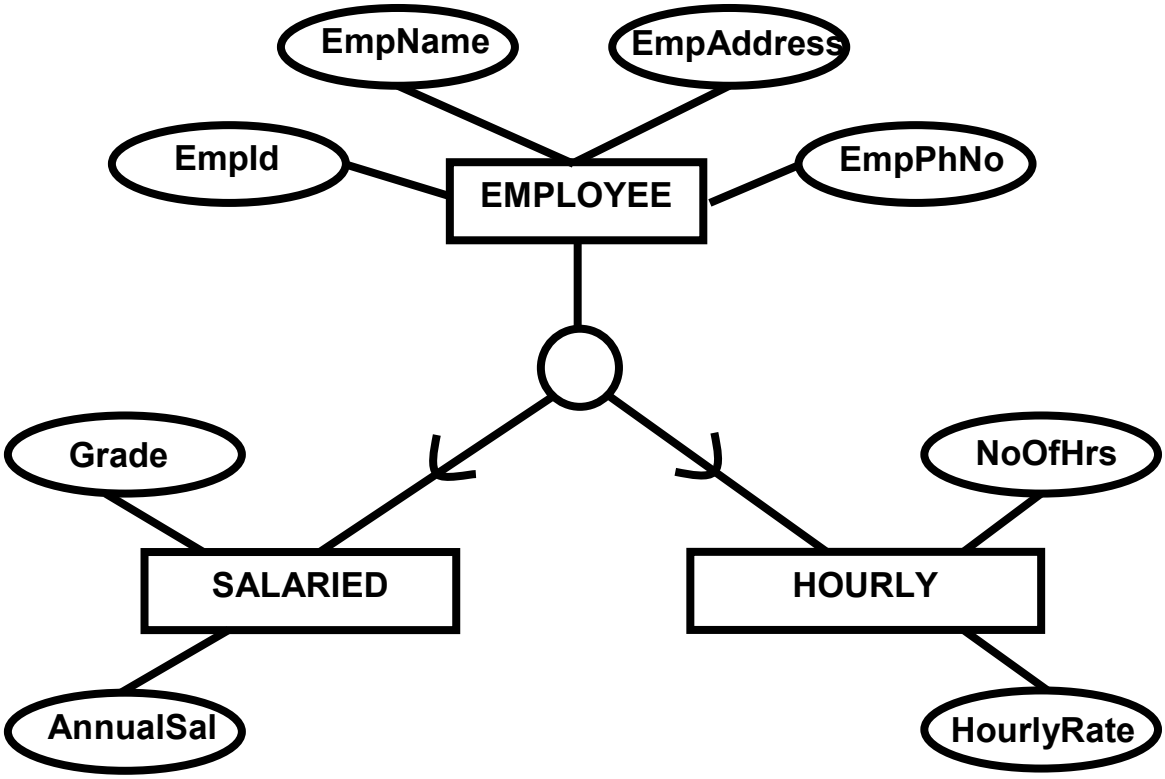


Fig-1 a

The Figure:1 above shows that the supertype and subtype relation between the SALARIED and HOURLY employees with the supertype entity EMPLOYEE, we can see that the attributes which are specific to the subtype entities are not shown with the supertype entity. Only those attributes are shown on the supertype entity which are to be inherited to the subtypes and are common to all the subtype entities associated with this supertype.

The example shows that there is a major entity or entity supertype name EMPLOYEE and has a number of attributes. Now that in a certain organization there can be a number of employees being paid on different payment criteria.

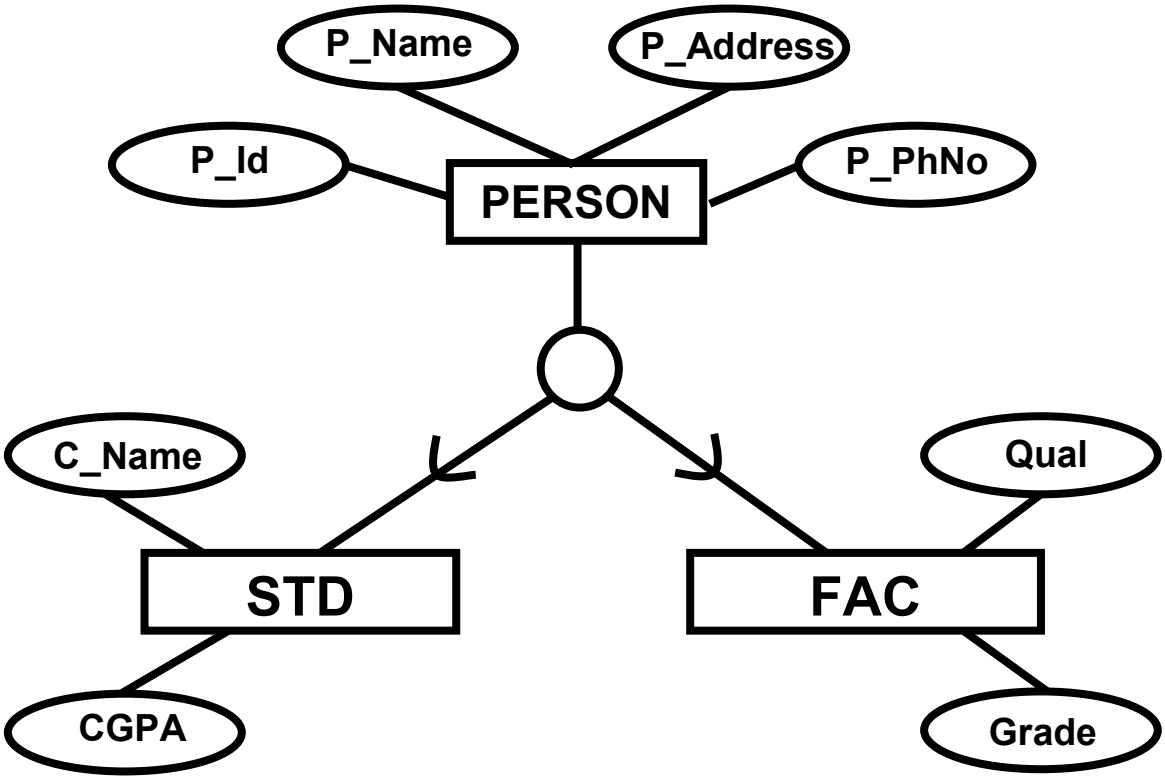


Fig – 1 b

The second example is that of student and the Faculty members who are at the super level same type of entities. Both the entities at the super level belong to the same entity of type Person. The distinct attributes of the student and faculty members are added later to he sub entities student and fac.

Supertype / subtype Relationship:

The use of supertype and subtype for the entities is very useful because it allows us to create hierarchy of the entities according to the attributes they have and we need not to write all the attributes again and again. We can group similar types of entities and the attributes associated with those entities at certain levels.

This also adds clarity to the definitions of the entities as it is not necessary to write the attribute again and again for all the entities.

Moreover it also eases the operation of removing or adding attributes from the entities, here it is worth noting that adding an attribute at the super entity level will add the

attribute to the below listed or derived sub entities and removing the attribute will remove the attribute from the entities at sublevels in the same way.

The process of identifying supertype and creating different type of sub entities is supported by the general knowledge of the designer about the organization and also based of the attributes of the entities which are entities existing in the system..

Specifying Constraints

Once there has been established a super/sub entity relationship there are a number of constraints which can be specified for this relationship for specifying further restrictions on the relationship.

Completeness Constraint

There are two types of completeness constraints, partial completeness constraints and total completeness constraints.

Total Completeness:

Total Completeness constraint exist only if we have a super type and some subtypes associated with that supertype, and the following situation exists between the super type and subtype.

All the instances of the supertype entity must be present in at one of the subtype entities, i.e.—there should be not instance of the supertype entity which does not belong to any of the subtype entity.

This is a specific situation when the supertype entities are very carefully analyzed for their associated subtype entities and no sub type entity is ignored when deriving sub entities from the supertype entity.

Partial Completeness Constraint:

This type of completeness constraint exists when it is not necessary for any supertype entity to have its entire instance set to be associated with any of the subtype entity.

This type of situation exists when we do not identify all subtype entities associated with a supertype entity, or ignore any subtype entity due to less importance of least usage in a specific scenario.

Disjointness Constraint

This rule or constraint defines the existence of a supertype entity in a subtype entity. There exist type types of disjoint rules.

- Disjointness rule
- Overlap rule

Disjoint constraint:
This constraint restricts the existence of one instance of any supertype entity to exactly one instance of any of the subtype entities.
Considering the example given in Fig 1a it is seen that there can be two types of employees, one which are fixed salary employees and the others are hourly paid employees. Now the disjoint rule tells that at a certain type an employee will be either hourly paid employee or salaried employee, he can not be placed in both the categories in parallel.

Overlap Rule:
This rule is in contrast with the disjoint rule, and tells that for one instance of any supertype entity there can be multiple instances existences of the of the instance for more then one subtype entities. Again taking the same example of the employee in an organization we can say that one employee who is working in an organization can be allowed to work for the company at hourly rates also once he has completed his duty as a salaried employee. In such a situation the employee instance record for this employee will be stored in both the sub entity types.

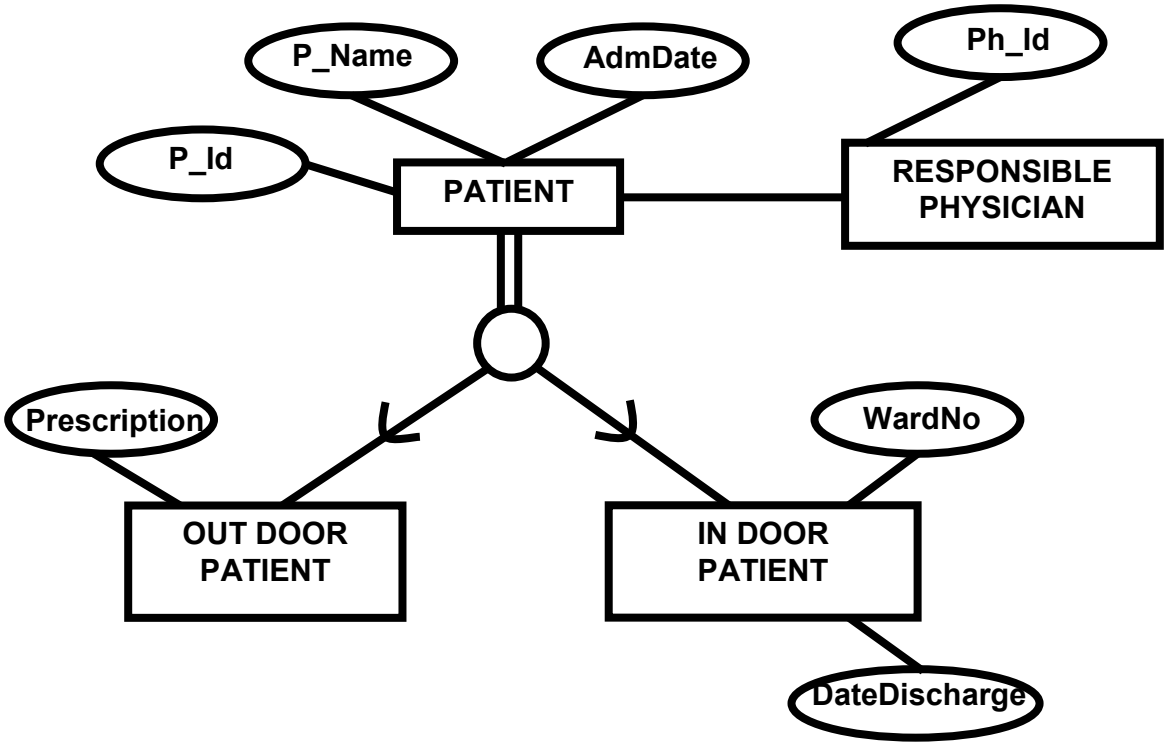


Fig 2-a

In the example the completeness of the relation is shown between the supertype entity and the subtype entity, it shows that for the data of patients we can have only two type of patients and one patient can be either an outdoor patient or indoor patient. In it we can see that we have identified all possible subtypes of the supertype patient. This implies a completeness constraint. One more thing to note here is the linked entity physician to the patient entity. And all the relationships associated with the supertype entity are inherited to subtype entities of the concerned supertype.

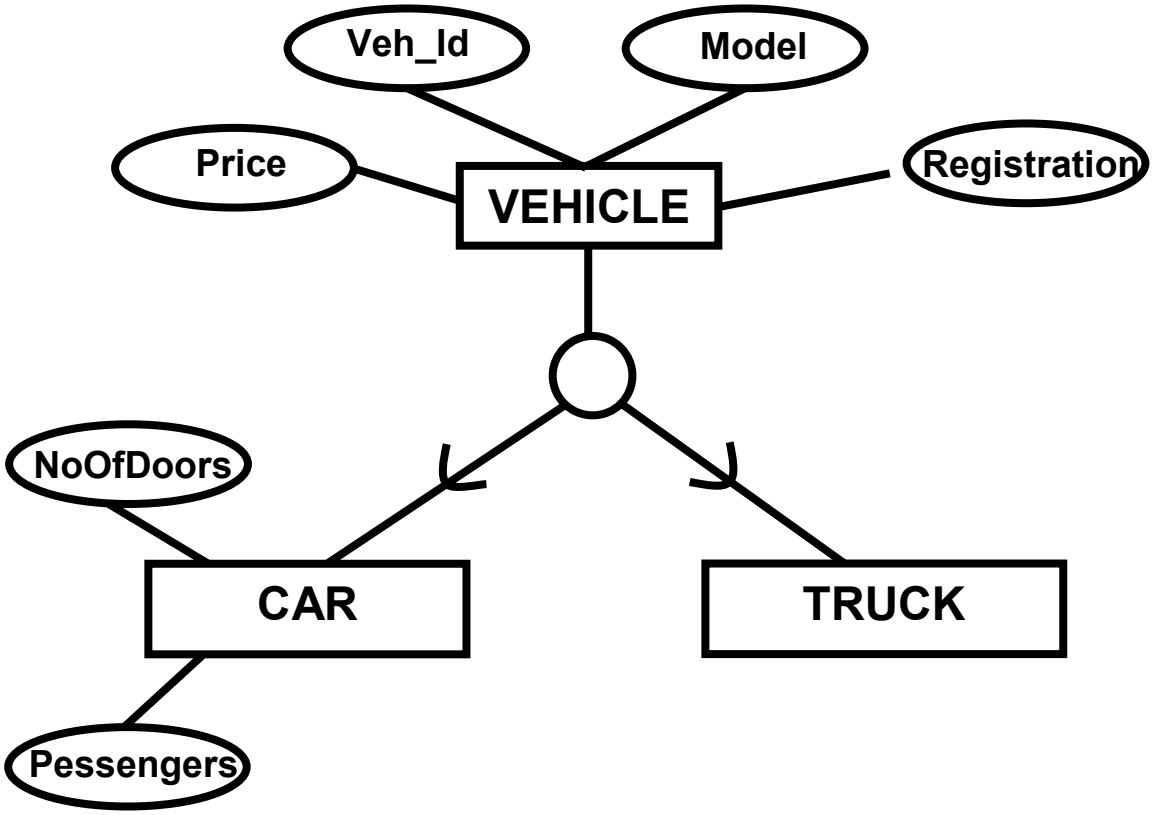


Fig 2-b

The Figure2b shows the supertype and subtype relationship among different type of vehicles. Here we can see that the Vehicle has only two subtypes, known as Truck and Car, As it is normal to have a number of other vehicles in the company of a certain type but when we have noted just a limited number of vehicles then it means that we are not interested in storing information for all the vehicles as separate entities. They may be stored in the vehicle entity type itself and distinct vehicle may be stored in the subtypes car and truck of the Vehicle.

This is a scenario where we have the freedom to store several entities and neglect others, and it is called as partial completeness constraint rule.

After the discussion of the Total Completeness and Partial completeness let us move to the next constraint that is disjointness and check for its examples.

Again in the Figure 2-a. we have the environment where patient entity type has two subtypes indoor and outdoor patient. To represent disjointness we place the letter “D” in the circle which is splitting the super entity type into two sub entity types. Suppose that the hospital has placed a restriction on the patient to be either a n indoor patient or

outdoor patient, in such a case there exists disjointness which specifies that the patients data can not be place in the database in both the subtype entities. It will be wither indoor or outdoor.

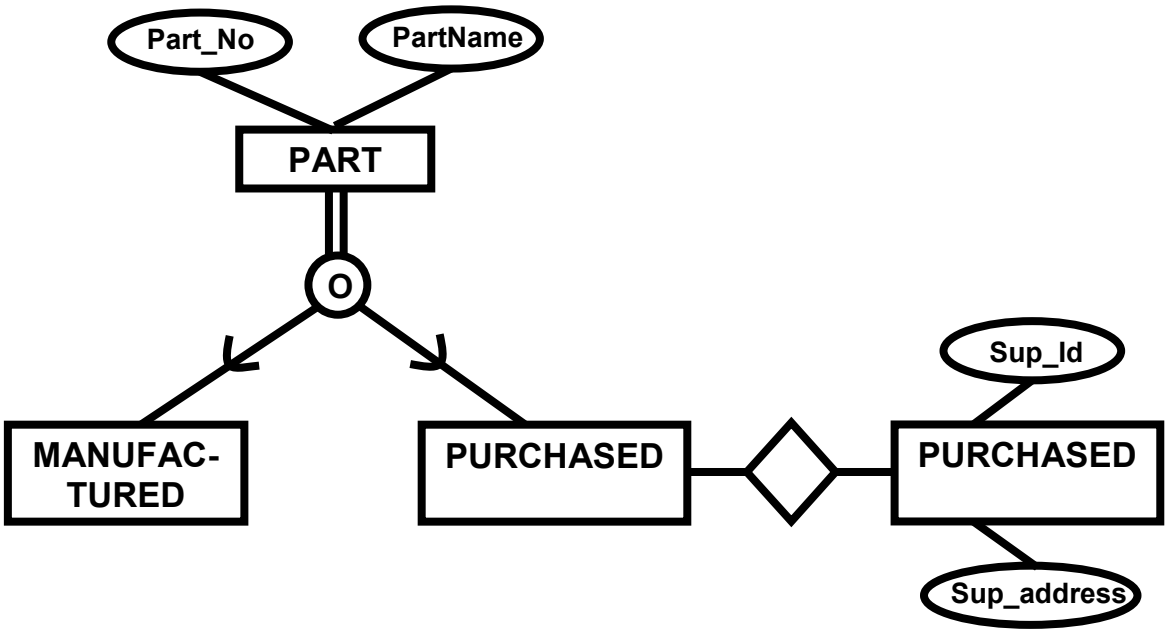


Fig- 3

The figure 3 above shows the second type of disjoint constraint which tells that the entity subtype instance can be repeated for any single entity supertype instance. We can see the relationship of a certain hardware company for the parts provided by the company to its clients. Now there may exist an overlapping situation for a certain part which is to be provided to a certain firm, but the manufactured quantity of that part is not enough to meet the specific order, In this situation the company purchases the remaining the deficient number of parts form the other suppliers. We can easily say that the data for that specific part is to be placed in both the entity subtypes. Because it belongs to both the subtype entities, this is an overlapping situation and expresses disjointness with overlapping. Another important thing which is to be noted here that the purchased part subtype entity has a relationship with another entity where the data for the suppliers is stored from whom the parts are bought. Now this relation does not have nay interaction with the manufactured parts relation as it is not connected with its supertype i.e.—parts supertype entity.

Considering the above discussed we can have four different types of combination existing for the supertype and subtype entities.

- Complete Disjoint
- Complete Overlapping
- Partial Disjoint
- Partial overlapping

Subtype Discriminator

This is a tool or a technique which provides us a methodology to determine that to which subtype one instance of a supertype belongs.

To determine the relation we place an attribute in the entity supertype which can specify through its value, that to which entity subtype it belongs.

For example we consider the example

There can be two different situations which specify the placement or relationship of a supertype entity instance in a subtype entity instance. First situation is that of disjoint situation where one supertype entity instance can be placed only in one subtype of that supertype. Let us consider the example of vehicles above in Figure-2-b it show that there can be two different vehicles car and truck associated with the supertype vehicle now if we place an attribute named Vehicle_type in the supertype we can easily determine the type of the associated subtype by placing a C for car and a T for truck instance of the vehicle.

The other situation where the Subtype discriminator is required the overlapping constraint; it is the situation where one supertype attribute can be placed in more than one subtype entities.

Considering again the part example shown in Figure 3, which has an overlapping constraint; In this situation we can have many solution one common solution is to place two attribute in the supertype one for manufactured and other one for purchased. We can combine them as a composite attribute, when we place Y for manufacture and N for

Purchased then it means the part is manufactured by the company, and similarly the following situation will give us further information

Attribute		
Manufacture	Purchased	Result
Y	Y	Manufacture Purchased
Y	N	Manufactured
N	Y	Purchased.

Significance of Subtype Discriminator:

Existence of subtype discriminator helps us a lot in finding the corresponding subtype entities, although we can find a subtype entity instance without having a subtype discriminator in the supertype but that involves lots of efforts and might consume a huge time in worst case situations.

This concludes out discussion of The ER Model in the course.

Lecture No. 12

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	

Overview of Lecture

In today's lecture we will discuss the ER Data model for an existing system and will go through a practice session for the logical design of the system

The system discusses is an examination section of an educational institute with the implementation of semester system.

Steps in the Study of system

Preliminary study of the system

- Students are enrolled in programs.
- The programs are based on courses
- Different courses are offered at the start of the semester
- Students enroll themselves for these courses at the start of semesters
- Enrolled courses by students and offered courses must not be same.
- The difference is due to the individual situation of every student, because if one student has not pass a certain course 'A' in the previous semester he will not be able to register for a course 'B' offered in this semester as the course 'A' is the prerequisite for course 'B'.
- After valid registration classes start.
- A Course which is offered is assigned to a teacher also
- There can be any mid term exams and in this system we have only one mid term
- All the students are given assignments and quizzes and are awarded marks against their performance.
- Result of the student is prepared on the basis of assignment marks, sessional and mid term marks and the final exam.
- GP (Grade point) for students is calculated in each subject.
- Average grade point is calculated on the basis of GPs in individual subjects

- And the Cumulative GPA is calculated for all the passed semesters.

Outputs Required

- Teachers and controller need class list or attendance sheet, class result; subject and overall
- Students need transcripts, semester result card, subject result

Entities associated with the system

- Students
- Teachers
- Controllers

Once the analysis of the system is done it is important to make a draft of the system using a standard tool which specifies the component and design of the system. This design is useful because anyone using the design can work on the existing system and clearly understand the working without working on the system from the scratch.

Tool used for such graphical design is Data Flow Diagram (DFD)
In the Figure -1 of the system we have a context diagram of the system which shows integration of different entities with the examination system, these include Registration system, controller, student and teacher entities.

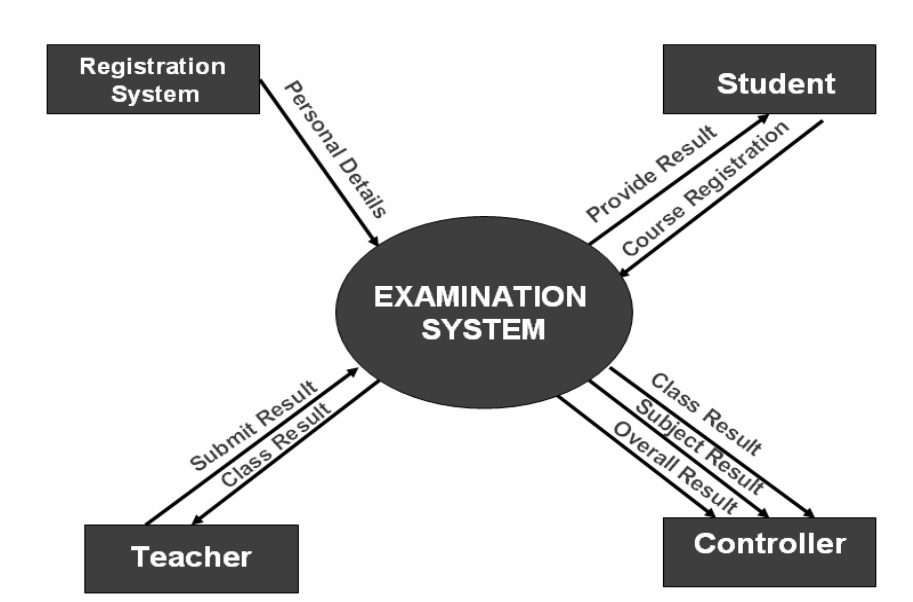


Fig-1

- From the diagram we can understand basic functionality of the system and can find how the data is flowing in the system and how different external entities are communicating or interacting with the system.

- First of all we have registration system, which provides the data of students to the systems once the registration process has been completed, this data is now free of errors in terms of validity of a certain student for a certain course or a semester.
- Second external entity interacting with the system is the teacher, a Teacher is given a list of students who are enrolled in a class and the registration system has declared them as valid students for that very course. Then the teacher allows those students in the class and continues the process of teaching the class, during this process the teacher takes test of the students and prepares papers for the students and also prepares quizzes to be submitted by students. All the data of students' attendance quizzes and assignments along-with different sessional results is then submitted by the teacher to the examination system which is responsible for preparation of results of the students
- Third interacting entity with the system is the controller's office it is provided with the semester overall result, subject results and also the result of each class fir performance evaluation and many other aspects.
- Fourth entity is student which externally interacts with the system for getting its result, the result is submitted to the student and may be in one of different forms such as, transcript and result card etc.

Level 0 Diagram

The three major modules which have been identified are given below our level 0 diagram will be based on these three modules and will elaborate and describe each of the modules in details.

- Subject registration
- Result submission
- Result calculation

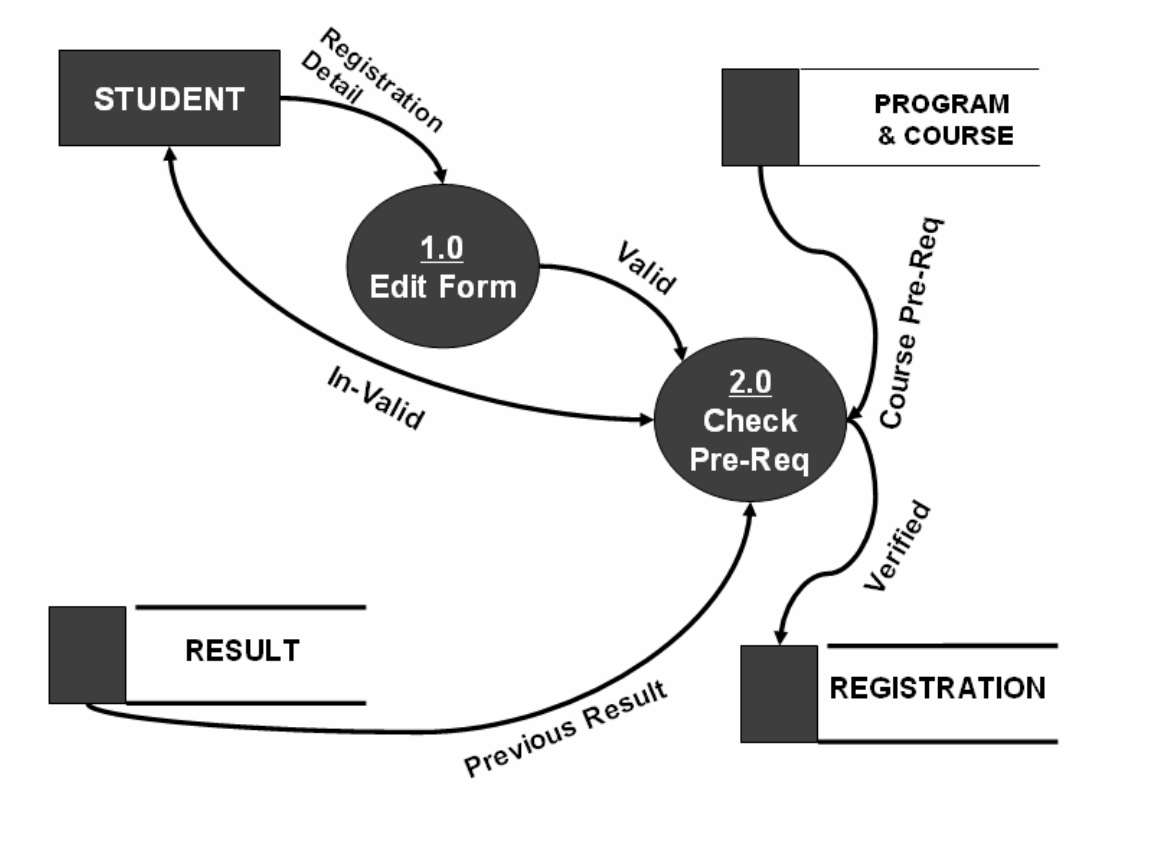


Fig 2

The first module identified in the system is the Registration of the students for the system As the DFD show a student applies for registration along-with certain registration information which is required by the system, Process 1.0 of the system checks the validity of information in the form if the Registration form is found to be valid the information in the form is passed onto the second process where the validity of registration is determined by checking certain prerequisites for the courses to which student wishes to be enrolled. After the prerequisite checking the data of the student is stored in a registration database for use by other processes in the system.

During this process the result of the students is also checked for the previous semester or previously studied subject to confirm whether the student has passed a certain pre-requisite subject before he can attempt to enroll for a second course which is based on that prerequisite.

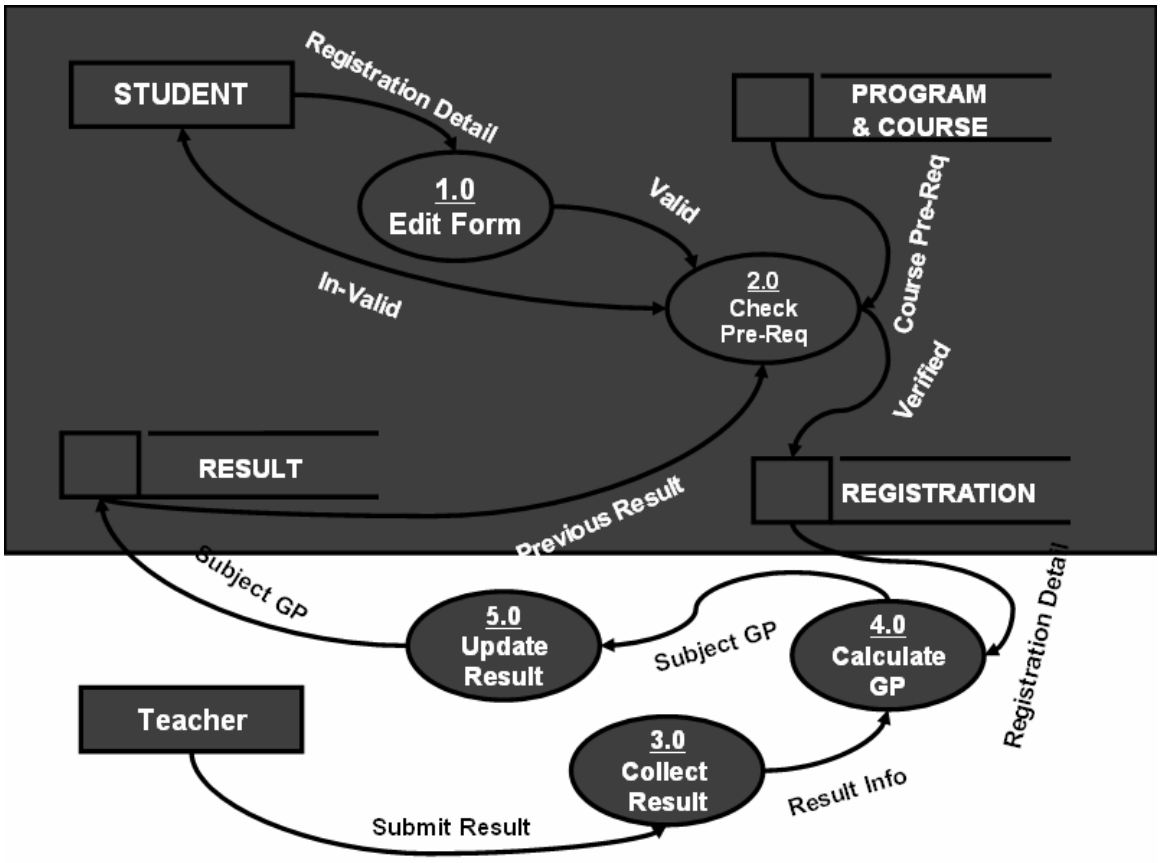


Fig-3

The Second DFD is in fact combination of the last diagram and some new details to the DFD this portion adds the result submission to the whole process of the system The teacher is the external entity here which is submitting the result, the result collection process is numbered 3.0, result is submitted by the teacher in parts, i.e. –separately for assignments, quizzes, tests, sessional and final result. The Collection process then forward the collected result to the Calculate GP Process, this process calculates the Grade point for the subject, the result with GP calculated is then moved forward to the update result process which then makes a change in the result data store by updating the result data for that specific student.

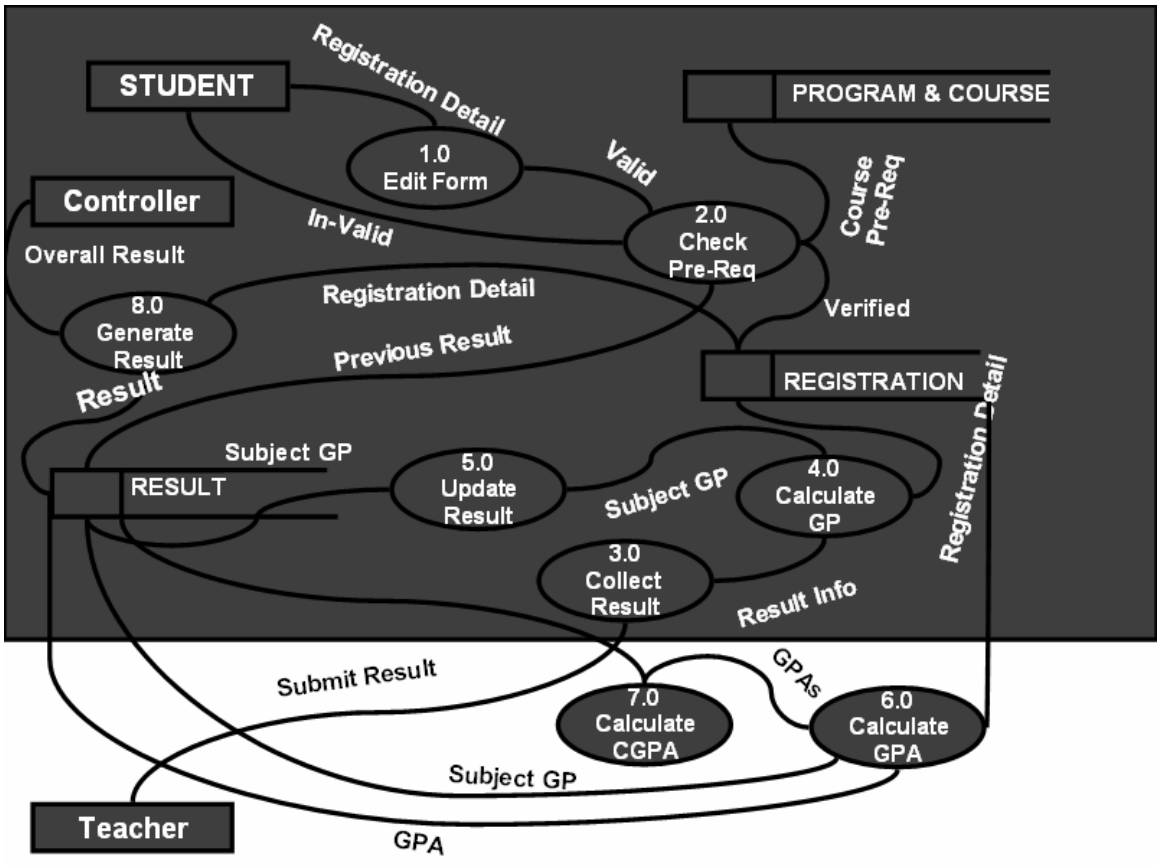


Fig-4

After the process of result submission the result for all the subjects is taken and the GPA is calculated, once the GPA is calculated the it is used for further calculation of CGPA and is forwarded to another process which is numbered 7.0 this process will calculate the CGPA by taking all the results of the current and previous semesters.

Further detailed diagram i.e.—Detailed DFD can be created using the given level 0 DFD and by expanding all the Processes further.

Cross Reference Matrix: doth:

This matrix is used to find out that what values or attributes will appear in which reports, for this purpose we write the major item names on a matrix in the row wise order and the reports which will be generated will be written on top or in column wise order.

Cross Reference Matrix

	Transcript	Semester Result Card	Attendance Sheet	Class Result (Subject Wise)	Class Result
Course_Name	✓	✓		✓	
CGPA	✓	✓			✓
Date	✓	✓		✓	✓
F_Name	✓	✓			
NameOfStudent		✓			
NameOfProgram		✓			
Reg_No	✓	✓	✓	✓	✓

This process infact is just cross link So the first Item transcript which may be or it will be needed by a specific student, second is Result card, next is attendance sheet then we have Class result (Subject wise) and finally the Class result as a whole, here by subject wise class result means that all the results of a specific class for a specific student considering each component, such as assignments, quizzes, sessional and terminal results.

Similarly all the mentioned items are marked with a tick which may needed by a certain output.

Let us see how the DFD and CRM are used in creating the ER-Diagram

The process of Creating ER-Diagram in fact lies in the Analysis phase and is started with identifying different entities which are present in the system. For this purpose we can use the DFD first of all.

Lets check our DFD, from there we can find the following entities.

Student	Controller
Courses	Teachers
Courses Offered	Programs
Registration	Results
Semester	

Here the point to be noted is that, we have picked the controller as the entity, although the controller is acting as an external entity for providing or getting information from the system, but in case of ER-Diagram the controller can not be represented as an entity because there is only one controller in any examination system and for such an entity instances a complete entity is not used.

So in this way we can exclude the controller entity, we will also take care of other entities before including them in our ED-Diagram. Another such example is results, which may not be as it is, added to the ER-Diagram, because there can be a number of result types at different stages of the Process, so there will be a number of different results.

We use our CRM in creating the ER-Diagram, because when we see the CRM, it has a number of item/attributes appearing on it, now from there we can see that whether these items belong to the same entity or more than one entity. And even if they belong to multiple entities we can find the relationship existing between those entities.

Considering our CRM we have transcript, it has a number of items appearing on it , as we know that there is to appear result for each semester on the transcript. So the attributes which belong to the personal information of the student shall be placed in the student entity and the data which belongs to the students' academic data will be placed in the courses or results entity for that student.

In the next phase we have to draw different entity type and the relationship which exist between those entities.

These we will discuss in the next lecture that how we draw relationships between different entities.

Lecture No. 13

Reading Material

Case Study	

Overview of Lecture

- E – R Diagram of Examination System
- Conceptual Data Base Design
- Relationships and Cardinalities between the entities
- In the previous lecture we discussed the Preliminary phase of the Examination system. We discussed the outputs required from the system and then we drew the data flow diagrams DFDs. From this lecture we will start the conceptual model of the system through E-R Diagram.

Identification of Entity Types of the Examination System

We had carried out a detailed preliminary study of the system, also drawn the data flow diagrams and then identified major entity types. Now we will identify the major attributes of the identities, then we will draw the relationships and cardinalities in between them and finally draw a complete E-R Diagram of the system.. So first of all we will see different attributes of the entities.

Program:
This entity means that what different courses are being offered by an institute, like MCS, BCS etc. Following are the major attributes of this entity:-

- **pr_Code.** It can be used as a primary key of the entity as it would always be unique for example MBA, MCS, etc.
- **max_Dur** It means that what is the maximum duration of any particular course , like 1 year , 2 years and so on.
- **no_of_Semesters** How many semesters this program has like four ,six and so on.
- **Pr_Lvl** This course is of undergraduate, graduate or post graduate level.

Student:

Following are the major attributes of this entity:-

- **Reg_No** This can be used as a primary key for this entity as it will be unique for every student.
- **st_Name** This would be the first name of all the students of an institute.
- **St_Father_name** This would represent the father's name of a student.
- **St_date_of_Birth.** The date of birth of all students including year , month and day.
- **st_Phone_no**
- **st_GPA** This is a very important attribute. Now to know the GPA of any student, we need to know the student reg no and the particular semester. So this is a multi valued attribute as to know the GPA, different attributes values are required. So this represented by a relation, which will be discussed in the relationships in between entities.
- **st_Subj_Detail** This is also a multi valued attribute ,as to know the marks in mid terms and final papers , student reg no and the particular subject are required

Teacher:
Following are the major attributes of this entity: -

- **teacher_Reg_No** This can be used as a primary key for this entity as it will be unique for every teacher.
- **teacher_Name** This would be the first name of all the teachers of an institute.
- **teacher_Father_name** This would represent the father's name of a teacher.
- **Qual**. The qualification of a teacher like Masters or Doctorate.
- **Experience** This can also be a multi-valued attribute or a single valued attribute. If only total experience of any teacher is required then it can be single valued, but if details are required as per the different appointments, then in that case it would be multi valued.
- **teacher_Sal** The total salary of the teacher.

There is one thing common in between teacher and student an entity that is the personal details of both, like name, father's name and addresses.

Course:
Following are the major attributes of this entity: -

- **course_Code** This can be used as a primary key for this entity as it will be unique for every course like CS-3207.
- **course_Name**
- **course_Prereq** This would also be a multi valued attribute as there can be a multiple requisites of any course . For example, Networking can have pre-requisites of Operating System and Data Structures. In this case this is a

recursive relation as pre-requisite of a course is a course. We will treat it as a recursive relation here.

- **Courses_Offered_in** This is also a multi valued attribute as to know the courses offered , program and semester both are required so this can also be represented by a relation

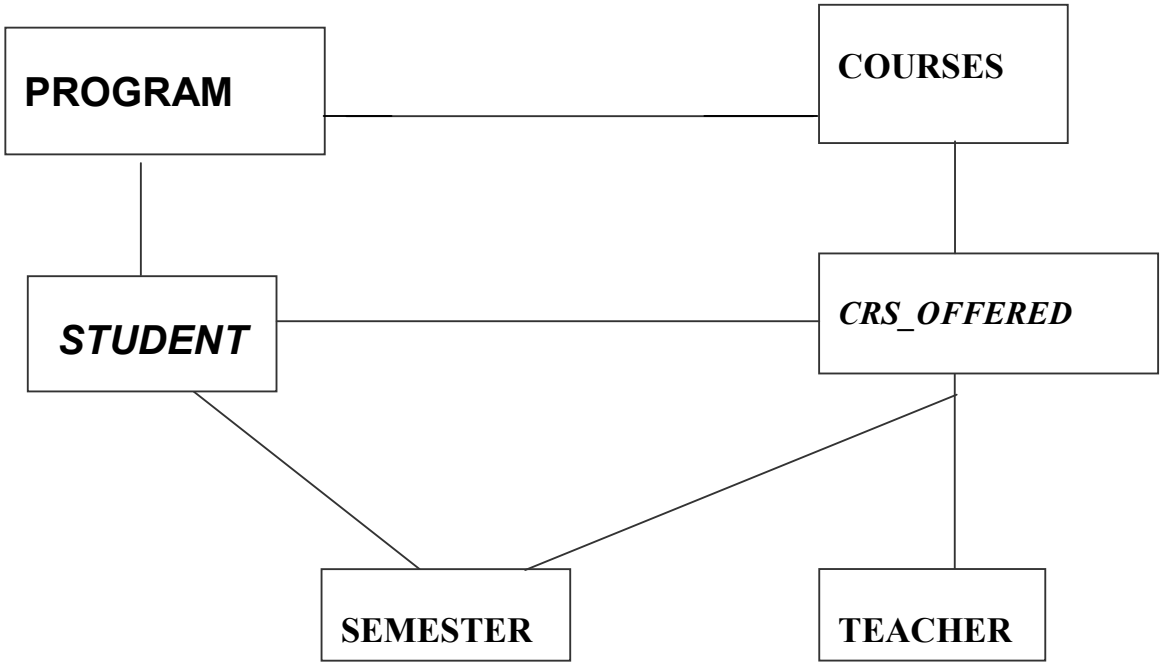
Semester:
Following are the major attributes of this entity: -

- **semester_Name** This can be used as a primary key for this entity as it will be unique for every semester like fall 2003 or spring 2004.
- **semester_Start_Date** The starting date of the semester
- **semester_End_Date** The ending date of the semester

Derived Attributes
There are certain attributes in the examination system which is derived like CGPA of a student can only be achieved from the semesters GPA. Similarly FPA of any particular semester can be achieved from subjects GPA of the semester. So this has to be kept in mind while drawing the E-R Diagram of the system.

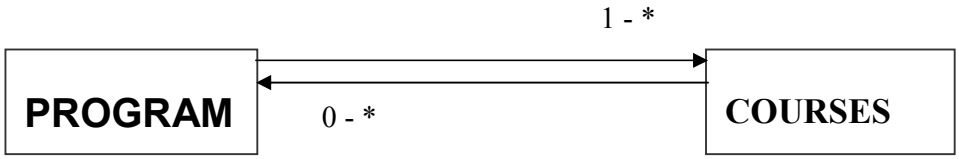
Relationships and Cardinalities in between Entities

Relationships and cardinalities in between entities is very important. We will now see the relationship of different entities one by one. The block diagrams of different entities are as under: -



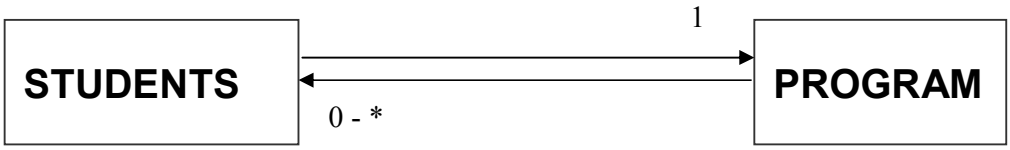
Program and Courses

The relationship between program and courses is that, if we want to know the courses in any particular program then the course codes and program codes are required. The cardinality between program and courses is of one to many (1 -*) , which means that any program will have minimum one course, and as many as required for that particular program. The cardinality of courses and program can be zero to many (0 - *). It means that if an institution wants, it can have a course without even any program as well. This course can be included in any program later on.



Students and Programs

The cardinality in between student and program is one, which means that every student can have minimum and maximum one program at any time. The cardinality in between programs and students can be zero to many (0 - *), which means that depending upon the requirements of any organization it can have a program which is presently not being offered to any students.



Semester and Course

The relationship in between semester and course is many to many. But it is essential to know the course offered during any particular semester so there is a requirement of an attribute, which is of relationship and when it is many to many it, can also serve as entity which is represented by a diamond in a rectangle. So here this can be a courses offered attribute, which would also be an entity. The primary key of semester that is semester code and primary key of course that is course code, after combining it becomes composite key which would be used to identify any particular course.

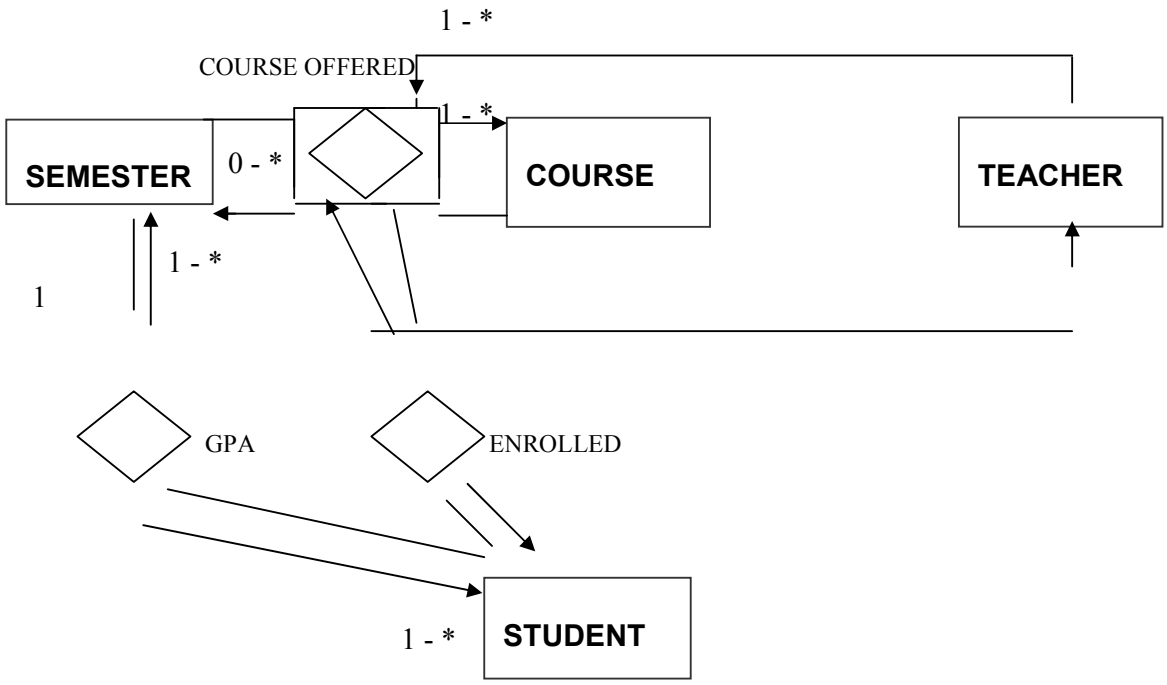
Course Offered and Teacher

There is a relationship in between course offered and teacher. The cardinality of course offered and teacher is one that is a teacher can only have one course at a time. Similarly the cardinality in between teacher and course offered is one to many, which means that a teacher can teach many courses.

© Virtual University of Pakistan

Student and Course Offered
The relationship in between student and course offered is many to many so this relationship is also through enrolled attribute, which can also serve as entity type. The primary keys of semester, course and student are used as composite key to identify, that which student has got which course.

Semester and Student
To find out GPA of any student the semester is also required to be known. So the relationship in between these two can be through result whose attribute GPA can be used. There is a many to many relationship in these two entities.



Conceptual Database Design

The outcome of analysis phase is the conceptual database design, which is drawn through E-R model. This design is independent of any tool or data model. This design can be implemented in multiple data models like network, relational or hierarchal models.

Logical Database Design

This is the next phase of database design, in which appropriate data model is chosen, and from here onwards it becomes tool dependent. We will select relational data model and our further lectures will be concerning relational data models

Conclusion

The E – R Model of Examination system of an educational institute discussed above is just a guideline. There can certainly be changes in this model depending upon the requirements of the organization and the outputs required. After drawing an E-R model, all the outputs, which are required, must be matched with the system. If it does not fulfill all the requirements then whole process must be rehashed once again. All necessary modifications and changes must be made before going ahead. For Example, if in this system attendance sheet of the students is required then program code, semester and course codes are required, this composite key will give the desired attendance sheet of the students.

© Virtual University of Pakistan
thegradup.com

Lecture No. 14

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section 6.1 – 6.3.3
"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill	

Overview of Lecture

- Logical Database Design
- Introduction to Relational Data Model
- Basic properties of a table
- Mathematical and database relations

From this lecture we are going to discuss the logical database design phase of database development process. Logical database design, like conceptual database design is our database design; it represents the structure of data that we need to store to fulfill the requirements of the users or organization for which we are developing the system. However there are certain differences between the two that are presented in the table below:

	Conceptual Database Design	Logical Database Design
1	Developed in a semantic data model (generally E-R data model)	In legacy data models (relational generally in current age)
2	Free of data model in which going to be implemented; many/any possible	Free of particular DBMS in which going to be implemented; many/any possible
3	Results from Analysis Phase	Obtained by translating the conceptual database design into another data model

4	Represented graphically	Descriptive
5	More expressive	Relatively less expressive
6	Going to be transformed and then implemented	Going to be implemented
You can think more, give a try		

Table 1: Differences between Conceptual and Logical Database Designs

As we have already discussed in previous lectures and as is given in row 2 of the above table, the conceptual database design can be transformed into any data model, like, hierarchical, network, relational or object-oriented. So the study of the logical database design requires first involves the study of the data model/(s) that we can possibly use for the purpose. However, in the current age, since early eighties, the most popular choice for the logical database design is the relational data model; so much popular that today it can be considered to be the only choice. Why? Because of its features we are going to discuss in today's lecture. That is why rather than studying different data models we will be studying only the relational data model. Once we study this, the development of logical database design is transformation of conceptual database design to relational one and the process is very simple and straightforward. So from today's lecture our discussion starts on the relational data model. Just for the sake of revision we repeat the definition of data model "a set of constructs/tools used to develop a database design; generally consists of three components which are constructs, manipulation language and integrity constraints". We have discussed it earlier that the later part of the definition (three components) fits precisely with the relational data model (RDM), that is, it has these components defined clearly.

Relational Data Model

The RDM is popular due to its two major strengths and they are:

- Simplicity
- Strong Mathematical Foundation

The RDM is simple, why, there is just one structure and that is a relation or a table. Even this single structure is very easy to understand, so a user of even of a moderate

genius can understand it easily. Secondly, it has a strong mathematical foundation that gives many advantages, like:

- Anything included/defined in RDM has got a precise meaning since it is based on mathematics, so there is no confusion.
- If we want to test something regarding RDM we can test it mathematically, if it works mathematically it will work with RDM (apart from some exceptions).
- The mathematics not only provided the RDM the structure (relation) but also well defined manipulation languages (relational algebra and relational calculus).
- It provided RDM certain boundaries, so any modification or addition we want to make in RDM, we have to see if it complies with the relational mathematics or not. We cannot afford to cross these boundaries since we will be losing the huge advantages provided by the mathematical backup.

"An IBM scientist E.F. Codd proposed the relational data model in 1970. At that time most database systems were based on one of two older data models (the hierarchical model and the network model); the relational model revolutionized the database field and largely replaced these earlier models. Prototype relational database management systems were developed in pioneering research projects at IBM and UC-Berkeley by the mid-70s, and several vendors were offering relational database products shortly thereafter. Today, the relational model is by far the dominant data model and is the foundation for the leading DBMS products, including IBM's DB2 family, Informix, Oracle, Sybase, Microsoft's Access and SQLServer, FoxBase, and Paradox. Relational database systems are ubiquitous in the marketplace and represent a multibillion dollar industry" [1]

The RDM is mainly used for designing/defining external and conceptual schemas; however to some extent physical schema is also specified in it. Separation of conceptual and physical levels makes data and schema manipulation much easier, contrary to previous data models. So the relational data model also truly supports "Three Level Schema Architecture".

Introduction to the Relational Data model

The RDM is based on a single structure and that is a relation. Speaking in terms of the E-R data model, both the entity types and relationships are represented using relations

in RDM. The relation in RDM is similar to the mathematical relation however database relation is also represented in a two dimensional structure called table. A table consists of rows and columns. Rows of a table are also called tuples. A row or tuple of a table represents a record or an entity instance, where as the columns of the table represent the properties or attributes.

stID	stName	clName	doB	sex
S001	M. Suhail	MCS	12/6/84	M
S002	M. Shahid	BCS	3/9/86	M
S003	Naila S.	MCS	7/8/85	F
S004	Rubab A.	MBA	23/4/86	F
S005	Ehsan M.	BBA	22/7/88	M

Table 2: A database relation represented in the form of a table

In the above diagram, a table is shown that consists of five rows and five columns. The top most rows contain the names of the columns or attributes whereas the rows represent the records or entity instances. There are six basic properties of the database relations which are:

- Each cell of a table contains atomic/single value

  A cell is the intersection of a row and a column, so it represents a value of an attribute in a particular row. The property means that the value stored in a single cell is considered as a single value. In real life we see many situations when a property/attribute of any entity contains multiple values, like, degrees that a person has, hobbies of a student, the cars owned by a person, the jobs of an employee. All these attributes have multiple values; these values cannot be placed as the value of a single attribute or in a cell of the table. It does not mean that the RDM cannot handle such situations, however, there are some special means that we have to adopt in these situations, and they can not be placed as the value of an attribute because an attribute can contain only a single value. The values of attributes shown in table 1 are all atomic or single.

- Each column has a distinct name; the name of the attribute it represents

  Each column has a heading that is basically the name of the attribute that the column represents. It has to be unique, that is, a table cannot have duplicated column/attribute names. In the table 2 above, the bold items in the first row represent the column/attribute names.

- The values of the attributes come from the same domain

  Each attribute is assigned a domain along with the name when it is defined. The domain represents the set of possible values that an attribute can have. Once the domain has been assigned to an attribute, then all the rows that are added into the table will have the values from the same domain for that particular column. For example, in the table 2 shown above the attribute doB (date of birth) is assigned the domain "Date", now all the rows have the date value against the attribute doB. This attribute cannot have a text or numeric value.

- The order of the columns is immaterial

  If the order of the columns in a table is changed, the table still remains the same. Order of the columns does not matter.

- The order of the rows is immaterial

  As with the columns, if rows' order is changed the table remains the same.

- Each row/tuple/record is distinct, no two rows can be same

  Two rows of a table cannot be same. The value of even a single attribute has to be different that makes the entire row distinct.

There are three components of the RDM, which are, construct (relation), manipulation language (SQL) and integrity constraints (two). We have discussed the relation so far; the last two components will be discussed later. In the next section we are going to

discuss the mathematical relations briefly that will help to link the mathematical relations with the database relations and will help in a better understanding of the later.

Mathematical Relations

Consider two sets

A = {x, y} B = {2, 4, 6}

Cartesian product of these sets (A x B) is a set that consists of ordered pairs where first element of the ordered pair belongs to set A where as second element belongs to set B, as shown below:

A X B= {(x,2), (x,4), (x,6), (y,2), (y,4), (y,6)}

A relation is some subset of this Cartesian product, For example,

- R1= {(x,2), (y,2),(x,6),(x,4)}
- R2 = {(x,4), (y,6), (y,4)}

The same notion of Cartesian product and relations can be applied to more than two sets, e.g. in case of three sets, we will have a relation of ordered triplets

Applying the same concept in a real world scenario, consider two sets Name and Age having the elements:

- Name = {Ali, Sana, Ahmed, Sara}
- Age = {15,16,17,18,…….,25}

Cartesian product of Name & Age

Name X Age= {(Ali,15), (Sana,15), (Ahmed,15), (Sara,15), …., (Ahmed,25), (Sara,25)}

Now consider a subset CLASS of this Cartesian product

CLASS = {(Ali, 18), (Sana, 17), (Ali, 20), (Ahmed, 19)}

This subset CLASS is a relation mathematically, however, it may represent a class in the real world where each ordered pair represents a particular student mentioning the name and age of a student. In the database context each ordered pair represents a tuple and elements in the ordered pairs represent values of the attributes. Think in this way, if Name and Age represent all possible values for names and ages of students, then any class you consider that will definitely be a subset of the Cartesian product of the Name and Age. That is, the name and age combination of all the students of any class

will be included in the Cartesian product and if we take out particulars ordered pairs that are related to a class then that will be a subset of the Cartesian product, a relation.

Database Relations

Let A1, A2, A3, …, An be some attributes and D1, D2, D3,…, Dn be their domains A relation scheme relates certain attributes with their domain in context of a relation. A relation scheme can be represented as:

R = (A1:D1, A2:D2, ……, An:Dn), for example,

STD Scheme = (stId:Text, stName: Text, stAdres:Text, doB:Date) OR

STD(stId, stName, stAdres, doB)

Whereas the stId, stName, stAdres and doB are the attribute names and Text, Text, Text and Date are their respective domains. A database relation as per this relation scheme can be:

STD={(stId:S001, stName:Ali, stAdres: Lahore, doB:12/12/76), (stId:S003, stName:A. Rehman, stAdres: RWP, doB:2/12/77)} OR

STD={(S001, Ali, Lahore, 12/12/76), (S003, A. Rehman, RWP, 2/12/77)}

The above relation if represented in a two dimensional structure will be called a table as is shown below:

stId	stName	stAdres	doB
S001	Ali	Lahore	12/12/76
S002	A. Rehman	RWP	2/12/77

With this, today's lecture is finished; the discussion on RDM will be continued in the next lecture.

Summary

In this lecture we have started the discussion on the logical database design that we develop from the conceptual database design. The later is generally developed using E-R data model, whereas for the former RDM is used. RDM is based on the theory of mathematical relations; a mathematical relation is subset of the Cartesian product of two or more sets. Relations are physically represented in the form of two-dimensional

structure called table, where rows/tuples represent records and columns represent the attributes.

Exercise:

Define different attributes (assigning name and domain to each) for an entity STUDENT, then apply the concept of Cartesian product on the domains of these attributes, then consider the records of your class fellows and see if it is the subset of the Cartesian product.

Lecture No. 15

Reading Material

Overview of Lecture

- Database and Math Relations
- Degree and Cardinality of Relation
- Integrity Constraints
- Transforming conceptual database design into logical database design
- Composite and multi-valued Attributes
- Identifier Dependency

In the previous lecture we discussed relational data model, its components and properties of a table. We also discussed mathematical and database relations. Now we will discuss the difference in between database and mathematical relations.

Database and Math Relations

We studied six basic properties of tables or database relations. If we compare these properties with those of mathematical relations then we find out that properties of both are the same except the one related to order of the columns. The order of columns in mathematical relations does matter, whereas in database relations it does not matter. There will not be any change in either math or database relations if we change the rows or tuples of any relation. It means that the only difference in between these two is of order of columns or attributes. A math relation is a Cartesian product of two sets. So if we change the order of theses two sets then the outcome of both will not be same. Therefore, the math relation changes by changing the order of columns. For Example , if there is a set A and a set B if we take Cartesian product of A and B then we take Cartesian product of B and A they will not be equal , so

$$\mathbf{A \times B \neq B \times A}$$

Rests of the properties between them are same.

Degree of a Relation

We will now discuss the degree of a relation not to be confused with the degree of a relationship. You would be definitely remembering that the relationship is a link or association between one or more entity types and we discussed it in E-R data model. However the degree of a relation is the number of columns in that relation. For Example consider the table given below:

STUDENT

StID	stName	clName	Sex
S001	Suhail	MCS	M
S002	Shahid	BCS	M
S003	Naila	MCS	F
S004	Rubab	MBA	F
S005	Ehsan	BBA	M

Table 1: The STUDENT table

Now in this example the relation STUDENT has four columns, so this relation has degree four.
Cardinality of a Relation
The number of rows present in a relation is called as cardinality of that relation. For example, in STUDENT table above, the number of rows is five, so the cardinality of the relation is five.

Relation Keys

The concept of key and all different types of keys is applicable to relations as well. We will now discuss the concept of foreign key in detail, which will be used quite frequently in the RDM.

Foreign Key
An attribute of a table B that is primary key in another table A is called as foreign key. For Example, consider the following two tables EMP and DEPT:

EMP (empId, empName, qual, depId)
DEPT (depId, depName, numEmp)

In this example there are two relations; EMP is having record of employees, whereas DEPT is having record of different departments of an organization. Now in EMP the primary key is empId, whereas in DEPT the primary key is depId. The depId which is primary key of DEPT is also present in EMP so this is a foreign key.

Requirements/Constraints of Foreign Key
Following are some requirements / constraints of foreign key:
There can be more than zero, one or multiple foreign keys in a table, depending on how many tables a particular table is related with. For example in the above example the EMP table is related with the DEPT table, so there is one foreign key depId,

whereas DEPT table does not contain any foreign key. Similarly, the EMP table may also be linked with DESIG table storing designations, in that case EMP will have another foreign key and alike.
The foreign key attribute, which is present as a primary key in another relation is called as home relation of foreign key attribute, so in EMP table the depId is foreign key and its home relation is DEPT.
The foreign key attribute and the one present in another relation as primary key can have different names, but both must have same domains. In DEPT, EMP example, both the PK and FK have the same name; they could have been different, it would not have made any difference however they must have the same domain.

The primary key is represented by underlining with a solid line, whereas foreign key is underlined by dashed or dotted line.
Primary Key : ———
Foreign Key : - - - - - - -
Integrity Constraints
Integrity constraints are very important and they play a vital role in relational data model. They are one of the three components of relational data model. These constraints are basic form of constraints, so basic that they are a part of the data model, due to this fact every DBMS that is based on the RDM must support them.

Entity Integrity Constraint:
It states that in a relation no attribute of a primary key (PK) can have null value. If a PK consists of single attribute, this constraint obviously applies on this attribute, so it cannot have the Null value. However, if a PK consists of multiple attributes, then none of the attributes of this PK can have the Null value in any of the instances.

Referential Integrity Constraint:
This constraint is applied to foreign keys. Foreign key is an attribute or attribute combination of a relation that is the primary key of another relation. This constraint states that if a foreign key exists in a relation, either the foreign key value must match the primary key value of some tuple in its home relation or the foreign key value must be completely null.

Significance of Constraints:
By definition a PK is a minimal identifier that is used to identify tuples uniquely. This means that no subset of the primary key is sufficient to provide unique identification of tuples. If we were to allow a null value for any part of the primary key, we would be demonstrating that not all of the attributes are needed to distinguish between tuples, which would contradict the definition.
Referential integrity constraint plays a vital role in maintaining the correctness, validity or integrity of the database. This means that when we have to ensure the proper enforcement of the referential integrity constraint to ensure the consistency and correctness of database. How? In the DEPT, EMP example above deptId in EMP is foreign key; this is being used as a link between the two tables. Now in every instance of EMP table the attribute deptId will have a value, this value will be used to get the name and other details of the department in which a particular employee works. If the value of deptId in EMP is Null in a row or tuple, it means this particular row is not related with any instance of the DEPT. From real-world scenario it means that this particular employee (whose is being represented by this row/tuple) has not been

assigned any department or his/her department has not been specified. These were two possible conditions that are being reflected by a legal value or Null value of the foreign key attribute. Now consider the situation when referential integrity constrains is being violated, that is, EMP.deptId contains a value that does not match with any of the value of DEPT.deptId. In this situation, if we want to know the department of an employee, then ooops, there is no department with this Id, that means, an employee has been assigned a department that does not exist in the organization or an illegal department. A wrong situation, not wanted. This is the significance of the integrity constraints.

Null Constraints:
A Null value of an attribute means that the value of attribute is not yet given, not defined yet. It can be assigned or defined later however. Through Null constraint we can monitor whether an attribute can have Null value or not. This is important and we have to make careful use of this constraint. This constraint is included in the definition of a table (or an attribute more precisely). By default a non-key attribute can have Null value, however, if we declare an attribute as Not Null, then this attribute must be assigned value while entering a record/tuple into the table containing that attribute. The question is, how do we apply or when do we apply this constraint, or why and when, on what basis we declare an attribute Null or Not Null. The answer is, from the system for which we are developing the database; it is generally an organizational constraint. For example, in a bank, a potential customer has to fill in a form that may comprise of many entries, but some of them would be necessary to fill in, like, the residential address, or the national Id card number. There may be some entries that may be optional, like fax number. When defining a database system for such a bank, if we create a CLIENT table then we will declare the must attributes as Not Null, so that a record cannot be successfully entered into the table until at least those attributes are not specified.

Default Value:
This constraint means that if we do not give any value to any particular attribute, it will be given a certain (default) value. This constraint is generally used for the efficiency purpose in the data entry process. Sometimes an attribute has a certain value that is assigned to it in most of the cases. For example, while entering data for the students, one attribute holds the current semester of the student. The value of this attribute is changed as a students passes through different exams or semesters during its degree. However, when a student is registered for the first time, it is generally registered in the first semesters. So in the new records the value of current semester attribute is generally 1. Rather than expecting the person entering the data to enter 1 in every record, we can place a default value of 1 for this attribute. So the person can simply skip the attribute and the attribute will automatically assume its default value.

Domain Constraint:
This is an essential constraint that is applied on every attribute, that is, every attribute has got a domain. Domain means the possible set of values that an attribute can have. For example, some attributes may have numeric values, like salary, age, marks etc. Some attributes may possess text or character values, like, name and address. Yet some others may have the date type value, like date of birth, joining date. Domain specification limits an attribute the nature of values that it can have. Domain is specified by associating a data type to an attribute while defining it. Exact data type name or specification depends on the particular tool that is being used. Domain helps

to maintain the integrity of the data by allowing only legal type of values to an attribute. For example, if the age attribute has been assigned a numeric data type then it will not be possible to assign a text or date value to it. As a database designer, this is your job to assign an appropriate data type to an attribute. Another perspective that needs to be considered is that the value assigned to attributes should be stored efficiently. That is, domain should not allocate unnecessary large space for the attribute. For example, age has to be numeric, but then there are different types of numeric data types supported by different tools that permit different range of values and hence require different storage space. Some of more frequently supported numeric data types include Byte, Integer, and Long Integer. Each of these types supports different range of numeric values and takes 1, 4 or 8 bytes to store. Now, if we declare the age attribute as Long Integer, it will definitely serve the purpose, but we will be allocating unnecessarily large space for each attribute. A Byte type would have been sufficient for this purpose since you won't find students or employees of age more than 255, the upper limit supported by Byte data type. Rather we can further restrict the domain of an attribute by applying a check constraint on the attribute. For example, the age attribute although assigned type Byte, still if a person by mistake enters the age of a student as 200, if this is year then it is not a legal age from today's age, yet it is legal from the domain constraint perspective. So we can limit the range supported by a domain by applying the check constraint by limiting it up to say 30 or 40, whatever is the rule of the organization. At the same time, don't be too sensitive about storage efficiency, since attribute domains should be large enough to cater the future enhancement in the possible set of values. So domain should be a bit larger than that is required today. In short, domain is also a very useful constraint and we should use it carefully as per the situation and requirements in the organization.

RDM Components

We have up till now studied following two components of the RDM, which are the Structure and Entity Integrity Constraints. The third part, that is, the Manipulation Language will be discussed in length in the coming lectures.

Designing Logical Database

Logical data base design is obtained from conceptual database design. We have seen that initially we studied the whole system through different means. Then we identified different entities, their attributes and relationship in between them. Then with the help of E-R data model we achieved an E-R diagram through different tools available in this model. This model is semantically rich. This is our conceptual database design. Then as we had to use relational data model so then we came to implementation phase for designing logical database through relational data model.

The process of converting conceptual database into logical database involves transformation of E-R data model into relational data model. We have studied both the data models, now we will see how to perform this transformation.

Transforming Rules

Following are the transforming rules for converting conceptual database into logical database design:

The rules are straightforward , which means that we just have to follow the rules mentioned and the required logical database design would be achieved

There are two ways of transforming first one is manually that is we analyze and evaluate and then transform. Second is that we have CASE tools available with us which can automatically convert conceptual database into required logical database design
If we are using CASE tools for transforming then we must evaluate it as there are multiple options available and we must make necessary changes if required.

Mapping Entity Types
Following are the rules for mapping entity types:
Each regular entity type (ET) is transformed straightaway into a relation. It means that whatever entities we had identified they would simply be converted into a relation and will have the same name of relation as kept earlier.
Primary key of the entity is declared as Primary key of relation and underlined.
Simple attributes of ET are included into the relation

For Example, figure 1 below shows the conversion of a strong entity type into equivalent relation:

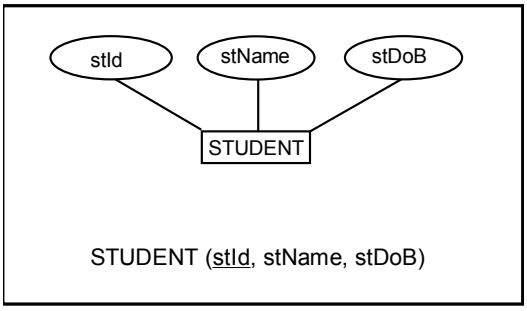


Fig. 1: An example strong entity type

Composite Attributes
These are those attributes which are a combination of two or more than two attributes. For address can be a composite attribute as it can have house no, street no, city code and country , similarly name can be a combination of first and last names. Now in relational data model composite attributes are treated differently. Since tables can contain only atomic values composite attributes need to be represented as a separate relation
For Example in student entity type there is a composite attribute Address, now in E-R model it can be represented with simple attributes but here in relational data model, there is a requirement of another relation like following:

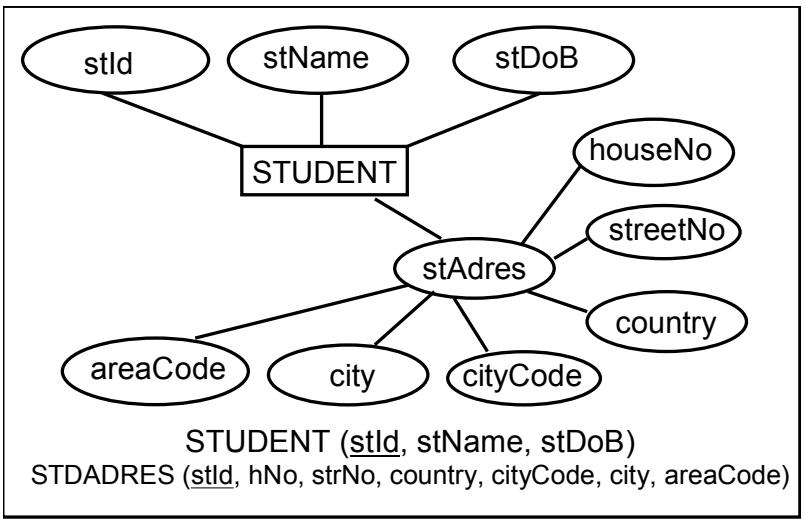


Fig. 2: Transformation of composite attribute

Figure 2 above presents an example of transforming a composite attribute into RDM, where it is transformed into a table that is linked with the STUDENT table with the primary key

Multi-valued Attributes

These are those attributes which can have more than one value against an attribute. For Example a student can have more than one hobby like riding, reading listening to music etc. So these attributes are treated differently in relational data model. Following are the rules for multi-valued attributes:-

An Entity type with a multi-valued attribute is transformed into two relations One contains the entity type and other simple attributes whereas the second one has the multi-valued attribute. In this way only single atomic value is stored against every attribute

The Primary key of the second relation is the primary key of first relation and the attribute value itself. So in the second relation the primary key is the combination of two attributes.

All values are accessed through reference of the primary key that also serves as

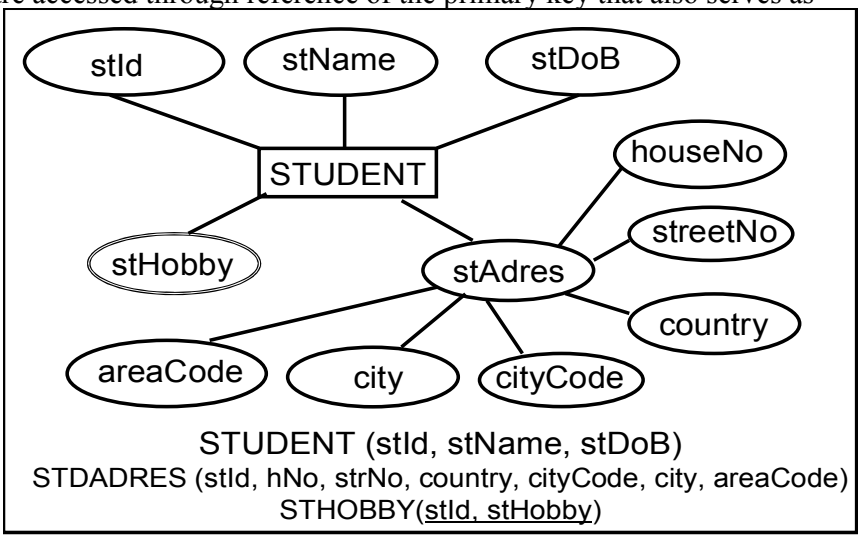


Fig. 3: Transformation of multi-valued attribute

foreign key.

Conclusion

In this lecture we have studied the difference between mathematical and database relations. The concepts of foreign key and especially the integrity constraints are very important and are basic for every database. Then how a conceptual database is transformed into logical database and in our case it is relational data model as it is the most widely used. We have also studied certain transforming rules for converting E-R data model into relational data model. Some other rule for this transformation will be studied in the coming lectures

You will receive exercise at the end of this topic.

Lecture No. 16

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Page 209

Overview of Lecture:

- Mapping Relationships
- Binary Relationships
- Unary Relationships
- Data Manipulation Languages

In the previous lecture we discussed the integrity constraints. How conceptual database is converted into logical database design, composite and multi-valued attributes. In this lecture we will discuss different mapping relationships.

Mapping Relationships

We have up till now converted an entity type and its attributes into RDM. Before establishing any relationship in between different relations, it is must to study the cardinality and degree of the relationship. There is a difference in between relation and relationship. Relation is a structure, which is obtained by converting an entity type in E-R model into a relation, whereas a relationship is in between two relations of relational data model. Relationships in relational data model are mapped according to their degree and cardinalities. It means before establishing a relationship there cardinality and degree is important.

Binary Relationships

Binary relationships are those, which are established between two entity type. Following are the three types of cardinalities for binary relationships:

- One to One
- One to Many
- Many to Many

In the following treatment in each of these situations is discussed.

One to Many:

In this type of cardinality one instance of a relation or entity type is mapped with many instances of second entity type, and inversely one instance of second entity type is mapped with one instance of first entity type. The participating entity types will be transformed into relations as has been already discussed. The relationship in this particular case will be implemented by placing the PK of the entity type (or corresponding relation) against one side of relationship will be included in the entity type (or corresponding relation) on the many side of the relationship as foreign key (FK). By declaring the PK-FK link between the two relations the referential integrity constraint is implemented automatically, which means that value of foreign key is either null or matches with its value in the home relation.

For Example, consider the binary relationship given in the figure 1 involving two entity types PROJET and EMPLOYEE. Now there is a one to many relationships between these two. On any one project many employees can work and one employee can work on only one project.

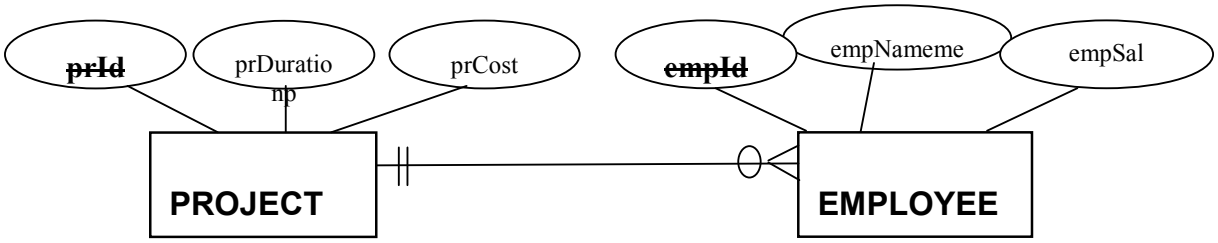


Fig. 1: A one to many relationship

The two participating entity types are transformed into relations and the relationship is implemented by including the PK of PROJECT (prId) into the EMPLOYEE as FK. So the transformation will be:

PROJECT (prId, prDura, prCost)
EMPLOYEE (empId, empName, empSal, prId)

The PK of the PROJECT has been included in EMPLOYEE as FK; both keys do not need to have same name, but they must have the same domain.

Minimum Cardinality:
This is a very important point, as minimum cardinality on one side needs special attention. Like in previous example an employee cannot exist if project is not assigned. So in that case the minimum cardinality has to be one. On the other hand if an instance of EMPLOYEE can exist with out being linked with an instance of the PROJECT then the minimum cardinality has to be zero. If the minimum cardinality is zero, then the FK is defined as normal and it can have the Null value, on the other hand if it is one then we have to declare the FK attribute(s) as Not Null. The Not Null constraint makes it a must to enter the value in the attribute(s) whereas the FK constraint will enforce the value to be a legal one. So you have to see the minimum cardinality while implementing a one to many relationship.

Many to Many Relationship:
In this type of relationship one instance of first entity can be mapped with many instances of second entity. Similarly one instance of second entity can be mapped with many instances of first entity type. In many to many relationship a third table is created for the relationship, which is also called as associative entity type. Generally,

the primary keys of the participating entity types are used as primary key of the third table.

For Example, there are two entity types BOOK and STD (student). Now many students can borrow a book and similarly many books can be issued to a student, so in this manner there is a many to many relationship. Now there would be a third relation as well which will have its primary key after combining primary keys of BOOK and STD. We have named that as transaction TRANS. Following are the attributes of these relations: -

- STD (stId, sName, sFname)
- BOOK (bkId, bkTitle, bkAuth)
- TRANS (stId,bkId, isDate,rtDate)

Now here the third relation TRANS has four attributes first two are the primary keys of two entities whereas the last two are issue date and return date.

One to One Relationship:

This is a special form of one to many relationship, in which one instance of first entity type is mapped with one instance of second entity type and also the other way round. In this relationship primary key of one entity type has to be included on other as foreign key. Normally primary key of compulsory side is included in the optional side. For example, there are two entities STD and STAPPLE (student application for scholarship). Now the relationship from STD to STAPPLE is optional whereas STAPPLE to STD is compulsory. That means every instance of STAPPLE must be related with one instance of STD, whereas it is not a must for an instance of STD to be related to an instance of STAPPLE, however, if it is related then it will be related to one instance of STAPPLE, that is, one student can give just one scholarship application. This relationship is shown in the figure below:

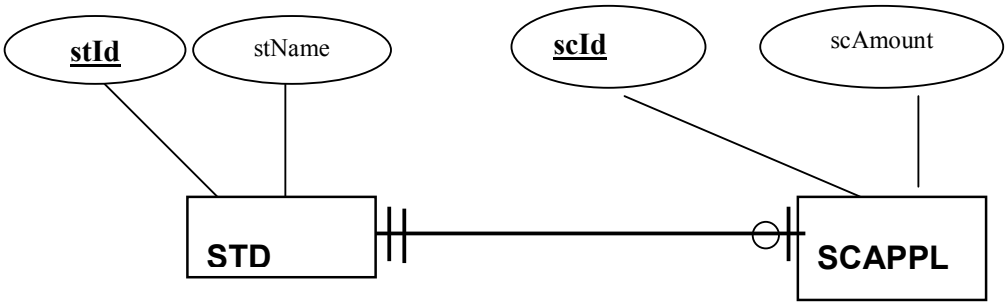


Fig. 2: A one to one relationship

While transforming, two relations will be created, one for STD and HOBBY each. For relationship PK of either one can be included in the other, it will work. But preferably, we should include the PK of STD in HOBBY as FK with Not Null constraint imposed on it.

STD (stId, stName)

STAPPLE (scId, scAmount, stId)

The advantage of including the PK of STD in STAPPLE as FK is that any instance of STAPPLE will definitely have a value in the FK attribute, that is, stId. Whereas if we do other way round; we include the PK of STAPPLE in STD as FK, then since the relationship is optional from STD side, the instances of STD may have Null value in the FK attribute (scId), causing the wastage of storage. More the number records with Null value more wastage.

Unary Relationship

These are the relationships, which involve a single entity. These are also called recursive relationships. Unary relationships may have one to one, one to many and many to many cardinalities. In unary one to one and one to may relationships, the PK of same entity type is used as foreign key in the same relation and obviously with the different name since same attribute name cannot be used in the same table. The example of one to one relationship is shown in the figure below:

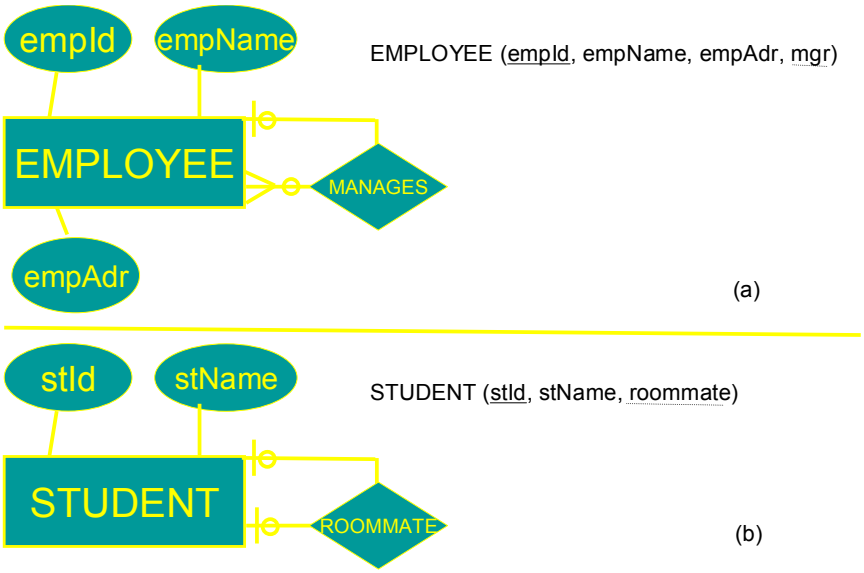


Fig. 3: One to one relationships (a) one to many (b) one to one and their transformation

In many to many relationships another relation is created with composite key. For example there is an entity type PART may have many to many recursive relationships, meaning one part consists of many parts and one part may be used in many parts. So in this case this is a many to many relationship. The treatment of such a relationship is shown in the figure below:

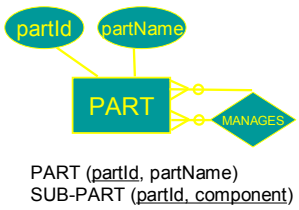


Fig. 4: Recursive many to many relationship and transformation

Super / Subtype Relationship:

Separate relations are created for each super type and subtypes. It means if there is one super type and there are three subtypes, so then four relations are to be created. After creating these relations then attributes are assigned. Common attributes are assigned to super type and specialized attributes are assigned to concerned subtypes. Primary key of super type is included in all relations that work for both link and

identity. Now to link the super type with concerned subtype there is a requirement of descriptive attribute, which is called as discriminator. It is used to identify which subtype is to be linked. For Example there is an entity type EMP which is a super type, now there are three subtypes, which are salaried, hourly and consultants. So now there is a requirement of a determinant, which can identify that which subtypes to be consulted, so with empId a special character can be added which can be used to identify the concerned subtype.

Summary of Mapping E-R Diagram to Relational DM:

We have up till now studied that how conceptual database design is converted into logical database. E-R data model is semantically rich and it has number of constructs for representing the whole system. Conceptual database is free of any data model, whereas logical database the required data model is chosen; in our case it is relational data model. First we identified the entity types, weak and strong entity types. Then we converted those entities into relations. After converting entities into relations then attributes are identified, different types of attributes are identified. Then relationships were made, in which cardinality and degree was identified. In ternary relationship, where three entities are involved, in this as well another relation is created to establish relationship among them. Then finally we had studied the super and sub types in which primary key of super type was used for both identity and link.

Data Manipulation Languages

This is the third component of relational data model. We have studied structure, which is the relation, integrity constraints both referential and entity integrity constraint. Data manipulation languages are used to carry out different operations like insertion, deletion and updation of data in database. Following are the two types of languages:

Procedural Languages:
These are those languages in which what to do and how to do on the database is required. It means whatever operation is to be done on the database that has to be told that how to perform.

Non -Procedural Languages:
These are those languages in which only what to do is required, rest how to do is done by the manipulation language itself.

Structured query language (SQL) is the most widely language used for manipulation of data. But we will first study Relational Algebra and Relational Calculus, which are procedural and non – procedural respectively.

Relational Algebra

Following are few major properties of relational algebra:

- **Relational algebra operations work on one or more relations to define another relation leaving the original intact. It means that the input for relational algebra can be one or more relations and the output would be another relation, but the original participating relations will remain unchanged and intact.** Both operands and results are relations, so output from one operation can become input to another operation. It means that the input and output both are relations so they can be used iteratively in different requirements.
- Allows expressions to be nested, just as in arithmetic. This property is called closure.
- There are five basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.
- These perform most of the data retrieval operations needed.
- It also has Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.

Exercise:

- Consider the example given in Ricardo book on page 216 and transform it into relational data model. Make any necessary assumptions if required.

Lecture No. 17

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Chapter 6

Overview of Lecture:

- Five Basic Operators of Relational Algebra
- Join Operation

In the previous lecture we discussed about the transformation of conceptual database design into relational database. In E-R data model we had number of constructs but in relational data model it was only a relation or a table. We started discussion on data manipulation languages (DML) of relational data model (SDM). We will now study in detail the different operators being used in relational algebra.

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. There are five basic operations of relational algebra. They are broadly divided into two categories:

Unary Operations:
These are those operations, which involve only one relation or table. These are Select and Project

Binary Operations:
These are those operations, which involve pairs of relations and are, therefore called as binary operations. The input for these operations is two relations and they produce a new relation without changing the original relations. These operations are:

- Union

- Set Difference
- Cartesian Product

The Select Operation:

The select operation is performed to select certain rows or tuples of a table, so it performs its action on the table horizontally. The tuples are selected through this operation using a predicate or condition. This command works on a single table and takes rows that meet a specified condition, copying them into a new table. Lower Greek letter sigma (**σ**) is used to denote the selection. The predicate appears as subscript to σ. The argument relation is given in parenthesis following theσ. As a result of this operation a new table is formed, without changing the original table. As a result of this operation all the attributes of the resulting table are same, which means that degree of the new and old tables are same. Only selected rows / tuples are picked up by the given condition. While processing a selection all the tuples of a table are looked up and those tuples, which match a particular condition, are picked up for the new table. The degree of the resulting relation will be the same as of the relation itself.

| σ | = | r(R) |

The select operation is commutative, which is as under: -

$$\sigma_{c1} (\sigma_{c2}(R)) = \sigma_{c2} (\sigma_{c1}(R))$$

If a condition 2 (c2) is applied on a relation R and then c1 is applied, the resulting table would be equivalent even if this condition is reversed that is first c1 is applied and then c2 is applied.

For example there is a table STUDENT with five attributes.

STUDENT

stId	stName	stAdr	prName	curSem
S1020	Sohail Dar	H#14, F/8-4,Islamabad.	MCS	4
S1038	Shoaib Ali	H#23, G/9-1,Islamabad	BCS	3
S1015	Tahira Ejaz	H#99, Lala Rukh Wah.	MCS	5
S1018	Arif Zia	H#10, E-8, Islamabad.	BIT	5

Fig. 1: An example STDUDENT table

The following is an example of select operation on the table STUDENT:

σ Curr_Sem > 3 (STUDENT)

The components of the select operations are clear from the above example; σ is the symbol being used (operato), "curr_sem > 3" written in the subscript is the predicate and STUDENT given in parentheses is the table name. The resulting relation of this command would contain record of those students whose semester is greater than three as under:

σ Curr_Sem > 3 (STUDENT)

stId	stName	stAdr	prName	curSem
S1020	Sohail Dar	H#14, F/8-4,Islamabad.	MCS	4
S1015	Tahira Ejaz	H#99, Lala Rukh Wah.	MCS	5
S1018	Arif Zia	H#10, E-8, Islamabad.	BIT	5

Fig. 2: Output relation of a select operation

In selection operation the comparison operators like <, >, =, <=, >=, <> can be used in the predicate. Similarly, we can also combine several simple predicates into a larger predicate using the connectives *and* (∧) and *or* (∨). Some other examples of select operation on the STUDENT table are given below:

σ stId = 'S1015' (STUDENT)

σ prName <> 'MCS' (STUDENT)

The Project Operator

The Select operation works horizontally on the table on the other hand the Project operator operates on a single table vertically, that is, it produces a vertical subset of the table, extracting the values of specified columns, eliminating duplicates, and placing the values in a new table. It is unary operation that returns its argument relation, with certain attributes left out. Since relation is a set any duplicate rows are eliminated. Projection is denoted by a Greek letter (∏). While using this operator all the rows of selected attributes of a relation are part of new relation. For example consider a relation FACULTY with five attributes and certain number of rows.

FACULTY

FacId	facName	Dept	Salary	Rank
F2345	Usman	CSE	21000	lecturer
F3456	Tahir	CSE	23000	Asst Prof
F4567	Ayesha	ENG	27000	Asst Prof
F5678	Samad	MATH	32000	Professor

Fig. 3: An example FACULY table

If we apply the projection operator on the table for the following commands all the rows of selected attributes will be shown, for example:

$\Pi_{\text{FacId, Salary}}$ **(FACULTY)**

FacId	Salary
F2345	21000
F3456	23000
F4567	27000
F5678	32000

Fig. 4: Output relation of a project operation on table of figure 3

Some other examples of project operation on the same table can be:

Π Fname, Rank (Faculty)

Π Facid, Salary,Rank (Faculty)

Composition of Relational Operators:

The relational operators like select and project can also be used in nested forms iteratively. As the result of an operation is a relation so this result can be used as an input for other operation. For Example if we want the names of faculty members along with departments, who are assistant professors then we have to perform both the select and project operations on the FACULTY table of figure 3. First selection operator is applied for selecting the associate professors, the operation outputs a relation that is given as input to the projection operation for the required attributes.

$$\Pi_{\text{facName, dept}} (\sigma_{\text{rank='Asst Prof'}} (\text{FACULTY}))$$

The output of this command will be

facName	Dept
Tahir	CSE
Ayesha	ENG

Fig. 5: Output relation of nested operations' command

We have to be careful about the nested command sequence. For example in the above nested operations example, if we change the sequence of operations and bring the projection first then the relation provided to select operation as input will not have the attribute of rank and so then selection operator can't be applied, so there would be an error. So although the sequence can be changed, but the required attributes should be there either for selection or projection.

The Union Operation:

We will now study the binary operations, which are also called as set operations. The first requirement for union operator is that the both the relations should be union compatible. It means that relations must meet the following two conditions:

- Both the relations should be of same degree, which means that the number of attributes in both relations should be exactly same
- The domains of corresponding attributes in both the relations should be same.

Corresponding attributes means first attributes of both relations, then second and so on.

It is denoted by U. If R and S are two relations, which are union compatible, if we take union of these two relations then the resulting relation would be the set of tuples either in R or S or both. Since it is set so there are no duplicate tuples. The union operator is commutative which means: -

R U S = S U R

For Example there are two relations COURSE1 and COURSE2 denoting the two tables storing the courses being offered at different campuses of an institute? Now if we want to know exactly what courses are being offered at both the campuses then we will take the union of two tables:

COURSE1

crId	progId	credHrs	courseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market

COURSE2

crId	progId	credHrs	courseTitle
C4567	P9873	4	Financial Management
C8944	P4567	4	Electronics

COURSE1 U COURSE2

crId	progId	credHrs	courseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C4567	P9873	4	Financial Management
C5678	P9873	3	Money & Capital Market
C8944	P4567	4	Electronics

Fig. 5: Two tables and output of union operation on those tables

So in the union of above two courses there are no repeated tuples and they are union compatible as well.

The Intersection Operation:
The intersection operation also has the requirement that both the relations should be union compatible, which means they are of same degree and same domains. It is represented by∩. If R and S are two relations and we take intersection of these two relations then the resulting relation would be the set of tuples, which are in both R and S. Just like union intersection is also commutative.

R ∩ S = S ∩ R

For Example, if we take intersection of COURSE1 and COURSE2 of figure 5 then the resulting relation would be set of tuples, which are common in both.

COURSE1 ∩ COURSE2

crId	progId	credHrs	courseTitle
C4567	P9873	4	Financial Management

Fig. 6: Output of intersection operation on COURSE1 and COURSE 2 tables of figure 5

The union and intersection operators are used less as compared to selection and projection operators.

The Set Difference Operator:
If R and S are two relations which are union compatible then difference of these two relations will be set of tuples that appear in R but do not appear in S. It is denoted by (-) for example if we apply difference operator on Course1 and Course2 then the resulting relation would be as under:

COURSE1 – COURSE2

CID	ProgID	Cred_Hrs	CourseTitle
C2345	P1245	3	Operating Sytems
C3456	P1245	4	Database Systems
C5678	P9873	3	Money & Capital Market

Fig. 7: Output of difference operation on COURSE1 and COURSE 2 tables of figure 5

Cartesian product:

The Cartesian product needs not to be union compatible. It means they can be of different degree. It is denoted by X. suppose there is a relation R with attributes (A1, A2,...An) and S with attributes (B1, B2……B_n). The Cartesian product will be:

R X S

The resulting relation will be containing all the attributes of R and all of S. Moreover, all the rows of R will be merged with all the rows of S. So if there are m attributes and C rows in R and n attributes and D rows in S then the relations R x S will contain m + n columns and C * D rows. It is also called as cross product. The Cartesian product is also commutative and associative. For Example there are two relations COURSE and STUEDNT

COURSE	
crId	**courseTitle**
C3456	Database Systems
C4567	Financial Management
C5678	Money & Capital Market

STUDENT	
stId	stName
S101	Ali Tahir
S103	Farah Hasan

COURSE X STUDENT

crId	**courseTitle**	**stId**	**stName**
C3456	Database Systems	S101	Ali Tahir
C4567	Financial Management	S101	AliTahr
C5678	Money & Capital Market	S101	Ali Tahir
C3456	Database Systems	S103	Farah Hasan
C4567	Financial Management	S103	Farah Hasan
C5678	Money & Capital Market	S103	Farah Hasan

Fig. 7: Input tables and output of Cartesian product

Join Operation:

Join is a special form of cross product of two tables. In Cartesian product we join a tuple of one table with the tuples of the second table. But in join there is a special requirement of relationship between tuples. For example if there is a relation STUDENT and a relation BOOK then it may be required to know that how many books have been issued to any particular student. Now in this case the primary key of STUDENT that is stId is a foreign key in BOOK table through which the join can be made. We will discuss in detail the different types of joins in our next lecture.

In this lecture we discussed different types of relational algebra operations. We will continue our discussion in the next lecture.

© Virtual University of Pakistan

Lecture No. 18

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section 6.6.1 – 6.6.3

Overview of Lecture:

- Types of Joins
- Relational Calculus
- Normalization

In the previous lecture we have studied the basic operators of relational algebra along with different examples. From this lecture we will study the different types of joins, which are very important and are used extensively in relational calculus.

Types of Joins

Join is a special form of cross product of two tables. It is a binary operation that allows combining certain selections and a Cartesian product into one operation. The join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes. Following are the different types of joins: -

1. Theta Join
2. Equi Join
3. Semi Join
4. Natural Join
5. Outer Joins

We will now discuss them one by one

Theta Join:

In theta join we apply the condition on input relation(s) and then only those selected rows are used in the cross product to be merged and included in the output. It means that in normal cross product all the rows of one relation are mapped/merged with all the rows of second relation, but here only selected rows of a relation are made cross product with second relation. It is denoted as under: -

$R \mathbf{X}_{\theta} S$

If R and S are two relations then ϴ is the condition, which is applied for select operation on one relation and then only selected rows are cross product with all the rows of second relation. For Example there are two relations of FACULTY and COURSE, now we will first apply select operation on the FACULTY relation for selection certain specific rows then these rows will have across product with COURSE relation, so this is the difference in between cross product and theta join. We will now see first both the relation their different attributes and then finally the cross product after carrying out select operation on relation.
From this example the difference in between cross product and theta join becomes clear.

FACULTY

facId	facName	dept	salary	rank
F234	Usman	CSE	21000	lecturer
F235	Tahir	CSE	23000	Asso Prof
F236	Ayesha	ENG	27000	Asso Prof
F237	Samad	ENG	32000	Professor

COURSE

crCode	crTitle	fId
C3456	Database Systems	F234
C3457	Financial Management	
C3458	Money & Capital Market	F236
C3459	Introduction to Accounting	F237

(σ rank = 'Asso Prof'(FACULTY)) X COURSE

facId	facName	dept	salary	rank	crCode	crTitle	fId
F235	Tahir	CSE	23000	Asso Prof	C3456	Database Systems	F234
F235	Tahir	CSE	23000	Asso Prof	C3457	Financial Management	
F235	Tahir	CSE	23000	Asso Prof	C3458	Money & Capital Market	F236
F235	Tahir	CSE	23000	Asso Prof	C3459	Introduction to Accounting	F237
F236	Ayesha	ENG	27000	Asso Prof	C3456	Database Systems	F234
F236	Ayesha	ENG	27000	Asso Prof	C3457	Financial Management	
F236	Ayesha	ENG	27000	Asso Prof	C3458	Money & Capital Market	F236

F236	Ayesha	ENG	27000	Asso Prof	C3459	Introduction to Accounting	F237

Fig. 1: Two tables with an example of theta join

In this example after fulfilling the select condition of Associate professor on faculty relation then it is cross product with course relation

Equi–Join:

This is the most used type of join. In equi–join rows are joined on the basis of values of a common attribute between the two relations. It means relations are joined on the basis of common attributes between them; which are meaningful. This means on the basis of primary key, which is a foreign key in another relation. Rows having the same value in the common attributes are joined. Common attributes appear twice in the output. It means that the attributes, which are common in both relations, appear twice, but only those rows, which are selected. Common attribute with the same name is qualified with the relation name in the output. It means that if primary and foreign keys of two relations are having the same names and if we take the equi – join of both then in the output relation the relation name will precede the attribute name. For Example, if we take the equi – join of faculty and course relations then the output would be as under: -

FACULTY $\bowtie$ FACULTY.facId=COURSE.fId COURSE

facId	**facName**	**dept**	**salary**	**rank**	**crCode**	**crTitle**	**fID**
F234	Usman	CSE	21000	lecturer	C3456	Database Systems	F234
F236	Ayesha	ENG	27000	Asso Prof	C3458	Money & Capital Market	F236
F237	Samad	ENG	32000	Professor	C3459	Introduction to Accounting	F237

Fig. 2: Equi-join on tables of figure 1

In the above example the name of common attribute between the two tables is different, that is, it is facId in FACULTY and fId in COURSE, so it is not required to qualify; however there is no harm doing it still. Now in this example after taking equi–join only those tuples are selected in the output whose values are common in both the relations.

Natural Join:

This is the most common and general form of join. If we simply say join, it means the natural join. It is same as equi–join but the difference is that in natural join, the common attribute appears only once. Now, it does not matter which common attribute should be part of the output relation as the values in both are same. For Example if we take the natural join of faculty and course the output would be as under: -

FACULTY facId, fId COURSE ⋈

facId	facName	dept	salary	rank	crCode	crTitle
F234	Usman	CSE	21000	Lecturer	C3456	Database Systems
F236	Ayesha	ENG	27000	Asso Prof	C3458	Money & Capital Market
F237	Samad	ENG	32000	Professor	C3459	Introduction to Accounting

Fig. 4: Natural join of FACULTY and COURSE tables of figure 1

In this example the common attribute appears only once, rest the behavior is same. Following are the different types of natural join:-

Left Outer Join:

In left outer join all the tuples of left relation remain part of the output. The tuples that have a matching tuple in the second relation do have the corresponding tuple from the second relation. However, for the tuples of the left relation, which do not have a matching record in the right tuple have Null values against the attributes of the right relation. The example is given in figure 5 below. It can be described in another way. Left outer join is the equi-join plus the non matching rows of the left side relation having Null against the attributes of right side relation.

Right Outer Join:

In right outer join all the tuples of right relation remain part of the output relation, whereas on the left side the tuples, which do not match with the right relation, are left as null. It means that right outer join will always have all the tuples of right relation and those tuples of left relation which are not matched are left as Null.

COURSE

bkId	bkTitile	stId
B10001	Intro to Database Systems	S104
B10002	Programming Fundamentals	S101
B10003	Intro Data Structures	S101
B10004	Modern Operating Systems	S103
B10005	Computer Architecture	
B10006	Advanced Networks	S104

STUDENT

stId	stName
S101	Ali Tahir
S103	Farah Hasan
S104	Farah Naz
S106	Asmat Dar
S107	Liaqat Ali

COURSE left outer join STUDENT

bkId	bkTitile	BOOK.stId	STUDENT.stId	stName
B10001	Intro to Database Systems	S104	S104	Farah Naz
B10002	Programming Fundamentals	S101	S101	Ali Tahir
B10003	Intro Data Structures	S101	S101	Ali Tahir
B10004	Modern Operating Systems	S103	S103	Farah Hasan
B10006	Advanced Networks	S104	S104	Farah Naz
B10005	Computer Architecture	Null	Null	Null

COURSE right outer join STUDENT				
bkId	**bkTitile**	**BOOK.stId**	**STUDENT.stId**	**stName**
B10001	Intro to Database Systems	S104	S104	Farah Naz
B10002	Programming Fundamentals	S101	S101	Ali Tahir
B10003	Intro Data Structures	S101	S101	Ali Tahir
B10004	Modern Operating Systems	S103	S103	Farah Hasan
B10006	Advanced Networks	S104	S104	Farah Naz
Null	Null	Null	S106	Asmat Dar
Null	Null	Null	S107	Liaqat Ali

Fig. 5: Input tables and left outer join and right outer join

Outer Join:

In outer join all the tuples of left and right relations are part of the output. It means that all those tuples of left relation which are not matched with right relation are left as Null. Similarly all those tuples of right relation which are not matched with left relation are left as Null.

COURSE outer join STUDENT				
bkId	**bkTitile**	**BOOK.stId**	**STUDENT.stId**	**stName**
B10001	Intro to Database Systems	S104	S104	Farah Naz
B10002	Programming Fundamentals	S101	S101	Ali Tahir
B10003	Intro Data Structures	S101	S101	Ali Tahir
B10004	Modern Operating Systems	S103	S103	Farah Hasan
B10006	Advanced Networks	S104	S104	Farah Naz
B10005	Computer Architecture	Null	Null	Null
Null	Null	Null	S106	Asmat Dar
Null	Null	Null	S107	Liaqat Ali

Fig. 6: outer join operation on tables of figure 5

Semi Join:

In semi join, first we take the natural join of two relations then we project the attributes of first table only. So after join and matching the common attribute of both relations only attributes of first relation are projected. For Example if we take the semi join of two relations faculty and course then the resulting relation would be as under:-

FACULTY $\ltimes$ COURSE

facId	facName	Dept	Salary	Rank
F234	Usman	CSE	21000	lecturer
F236	Ayesha	ENG	27000	Asso Prof
F237	Samad	ENG	32000	Professor

Fig. 7: Semi-join operation on tables of figure 1

Now the resulting relation has attributes of first relation only after taking the natural join of both relations.

Relational Calculus

Relational Calculus is a nonprocedural formal relational data manipulation language in which the user simply specifies what data should be retrieved, but not how to retrieve it. It is an alternative standard for relational data manipulation languages. The relational calculus is not related to the familiar differential and integral calculus in mathematics, but takes its name from a branch of symbolic logic called the predicate calculus. It has two following forms: -

- **Tuple Oriented Relational Calculus**
- **Domain Oriented Relational Calculus**

Tuple Oriented Relational Calculus:

In tuple oriented relational calculus we are interested primarily in finding relation tuples for which a predicate is true. To do so we need tuple variables. A tuple variable is a variable that takes on only the tuples of some relation or relations as its range of values. It actually corresponds to a mathematical domain. We specify the range of a tuple variable by a statement such as: -

RANGE OF S IS STUDENT

Here, S is the tuple variable and STUDENT is the range, so that S always represents a tuple of STUDENT. It is expressed as

{S | P (S)}

We will read it as find the set of all tuples S such that P(S) is true, where P implies the predicate condition now suppose range of R is STUDENT

{R | R.Credits > 50}

We will say like find the stuId, stuName, majors etc of all students having more than 50 credits.

Domain Oriented Relational Calculus:

Normalization

There are four types of anomalies, which are of concern, redundancy, insertion, deletion and updation. Normalization is not compulsory, but it is strongly

recommended that normalization must be done. Because normalized design makes the maintenance of database much easier. While carrying out the process of normalization, it should be applied on each table of database. It is performed after the logical database design. This process is also being followed informally during conceptual database design as well.

Normalization Process

There are different forms or levels of normalization. They are called as first, second and so on. Each normalized form has certain requirements or conditions, which must be fulfilled. If a table or relation fulfills any particular form then it is said to be in that normal form. The process is applied on each relation of the database. The minimum form in which all the tables are in is called the normal form of entire database. The main objective of normalization is to place the database in highest form of normalization.

Summary

In this lecture we have studied the different types of joins, with the help of which we can join different tables. We can get different types of outputs from joins. Then we studied relational calculus in which we briefly touched upon tuple and domain oriented relational calculus. Lastly we started the process of normalization which is a very important topic and we will discuss in detail this topic in the coming lectures.

Exercise:

Draw two tables of PROJECT and EMPLOYEE along with different attribute, include a common attribute between the two to implement the PK/FK relationship and populate both the tables. Then apply all types of joins and observe the difference in the output relations

Lecture No. 19

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section 7.1 – 7.7
"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill	

Overview of Lecture:

- Functional Dependency
- Inference Rules
- Normal Forms

In the previous lecture we have studied different types of joins, which are used to connect two different tables and give different output relations. We also started the basics of normalization. From this lecture we will study in length different aspects of normalization.

Functional Dependency

Normalization is based on the concept of functional dependency. A functional dependency is a type of relationship between attributes.

Definition of Functional Dependency

If A and B are attributes or sets of attributes of relation R, we say that B is functionally dependent on A if each value of A in R has associated with it exactly one value of B in R.

We write this as A⟶ B, read as "A functionally determines B" or " A determines B". This does not mean that A causes B or that the value of B can be calculated from the value of A by a formula, although sometimes that is the case. It simply means that if we know the value of A and we examine the table of relation R, we will find only one value of B in all the rows that have the given value of A at any one time. Thus then the two rows have the same A value, they must also have the same B value. However, for a given B value, there may be several different A values. When a functional dependency exits, the attributes or set of attributes on the left side of the arrow is called a determinant. Attribute of set of attributes on left side are called determinant and on right are called dependants. If there is a relation R with attributes (a,b,c,d,e)

a ⟶ b,c,d

d ⟶e

For Example there is a relation of student with following attributes. We will establish the functional dependency of different attributes: -

STD (stId,stName,stAdr,prName,credits)

stId ⟶ stName,stAdr,prName,credits

prName ⟶ credits

Now in this example if we know the stID we can tell the complete information about that student. Similarly if we know the prName , we can tell the credit hours for any particular subject.

Functional Dependencies and Keys:

We can determine the keys of a relation after seeing its functional dependencies. The determinant of functional dependency that determines all attributes of that table is the super key. Super key is an attribute or a set of attributes that identifies an entity uniquely. In a table, a super key is any column or set of columns whose values can be used to distinguish one row from another. A minimal super key is the candidate key , so if a determinant of functional dependency determines all attributes of that relation then it is definitely a super key and if there is no other functional dependency whereas a subset of this determinant is a super key then it is a candidate key. So the functional dependencies help to identify keys. We have an example as under: -

EMP (eId,eName,eAdr,eDept,prId,prSal)

eId ⟶(eName,eAdr,eDept)

eId,prId ⟶ prSal

Now in this example in the employee relation eId is the key from which we can uniquely determine the employee name address and department . Similarly if we know the employee ID and project ID we can find the project salary as well. So FDs help in finding out the keys and their relation as well.

Inference Rules

Rules of Inference for functional dependencies, called inference axioms or Armstrong axioms, after their developer, can be used to find all the FDs logically implied by a set of FDs.These rules are sound , meaning that they are an immediate consequence of the definition of functional dependency and that any FD that can be derived from a given set of FDs using them is true. They are also complete, meaning they can be used to derive every valid reference about dependencies .Let A,B,C and D be subsets of attributes of a relation R then following are the different inference rules: -

Reflexivity:
If B is a subset of A, then A ⟶ B. This also implies that A ⟶ A always holds. Functional dependencies of this type are called trivial dependencies. For Example
StName,stAdr ⟶ stName
stName ⟶ stName

Augmentation:
If we have A ⟶ B then AC. ⟶ BC. For Example
If stId ⟶ stName then
StId,stAdr ⟶ stName,stadr

Transitivity:
If A ⟶ B and B ⟶ C, then A ⟶ C
If stId ⟶ prName and prName ⟶ credits then
stId ⟶ credits

Additivity or Union:
If A ⟶ B and A ⟶ C, then A ⟶ BC
If empId ⟶ eName and empId ⟶ qual Then we can write it as
empId ⟶ qual

Projectivity or Decomposition
If A ⟶ BC then A ⟶ B and A ⟶ C
If empId ⟶ eName,qual Then we can write it as
empId ⟶ eName and empID ⟶ qual

Pseudo transitivity:
If A ⟶ B and CB ⟶ D, then AC ⟶ D
If stID ⟶ stName and stName,fName ⟶ stAdr Then we can write it as
StId,fName ⟶ stAdr

Normal Forms

Normalization is basically; a process of efficiently organizing data in a database. There are two goals of the normalization process: eliminate redundant data (for example, storing the same data in more than one table) and ensure data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored. We will now study the first normal form

First Normal Form:
A relation is in first normal form if and only if every attribute is single valued for each tuple. This means that each attribute in each row , or each cell of the table, contains only one value. No repeating fields or groups are allowed. An alternative way of describing first normal form is to say that the domains of attributes of a relation are atomic, that is they consist of single units that cannot be broken down further. There is no multivalued (repeating group) in the relation multiple values create problems in performing operations like select or join. For Example there is a relation of Student

STD(stIdstName,stAdr,prName,bkId)

stId	stName	stAdr	prName	bkId
S1020	Sohail Dar	I-8 Islamabad	MCS	B00129
S1038	Shoaib Ali	G-6 Islamabad	BCS	B00327
S1015	Tahira Ejaz	L Rukh Wah	MCS	B08945, B06352
S1018	Arif Zia	E-8, Islamabad.	BIT	B08474

Now in this table there is no unique value for every tuple, like for S1015 there are two values for bookId. So to bring it in the first normal form.

stId	stName	stAdr	prName	bkId
S1020	Sohail Dar	I-8 Islamabad	MCS	B00129
S1038	Shoaib Ali	G-6 Islamabad	BCS	B00327
S1015	Tahira Ejaz	L Rukh Wah	MCS	B08945
S1015	Tahira Ejaz	L Rukh Wah	MCS	B06352
S1018	Arif Zia	E-8, Islamabad.	BIT	B08474

Now this table is in first normal form and for every tuple there is a unique value.

Second Normal Form:
A relation is in second normal form (2NF) if and only if it is in first normal form and all the nonkey attributes are fully functionally dependent on the key. Clearly, if a relation is in 1NF and the key consists of a single attribute, the relation is automatically in 2NF. The only time we have to be concerned about 2NF is when the key is composite. Second normal form (2NF) addresses the concept of removing duplicative data. It remove subsets of data that apply to multiple rows of a table and place them in separate tables. It creates relationships between these new tables and their predecessors through the use of foreign keys.

Summary

Normalization is the process of structuring relational database schema such that most ambiguity is removed. The stages of normalization are referred to as normal forms and progress from the least restrictive (First Normal Form) through the most restrictive (Fifth Normal Form). Generally, most database designers do not attempt to implement anything higher than Third Normal Form or Boyce-Codd Normal Form.

We have started the process of normalization in this lecture. We will cover this topic in detail in the coming lectures.

Exercise:

Draw the tables of an examination system along with attributes and bring those relations in First Normal Form.

Lecture No. 20

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Section 7.7 – 7.10
"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill	

Overview of Lecture:

- Second and Third Normal Form
- Boyce - Codd Normal Form
- Higher Normal Forms

In the previous lecture we have discussed functional dependency, the inference rules and the different normal forms. From this lecture we will study in length the second and third normal forms.

Second Normal Form

A relation is in second normal form if and only if it is in first normal form and all nonkey attributes are fully functionally dependent on the key. Clearly if a relation is in 1NF and the key consists of a single attribute, the relation is automatically 2NF. The only time we have to be concerned 2NF is when the key is composite. A relation that is not in 2NF exhibits the update, insertion and deletion anomalies we will now see it with an example. Consider the following relation.

CLASS (crId, stId, stName, fId, room, grade)

crId, stId ⟶ stName, fId, room, grade

stId ⟶ stName

crId ⟶ fId, room

Now in this relation the key is course ID and student ID. The requirement of 2NF is that all non-key attributes should be fully dependent on the key there should be no

partial dependency of the attributes. But in this relation student ID is dependent on student name and similarly course ID is partially dependent on faculty ID and room, so it is not in second normal form. At this level of normalization, each column in a table that is not a determiner of the contents of another column must itself be a function of the other columns in the table. For example, in a table with three columns containing customer ID, product sold, and price of the product when sold, the price would be a function of the customer ID (entitled to a discount) and the specific product. If a relation is not in 2NF then there are some anomalies, which are as under:

-

- Redundancy
- Insertion Anomaly
- Deletion Anomaly
- Updation Anomaly

The general requirements of 2NF are:-

- Remove subsets of data that apply to multiple rows of a table and place them in separate rows.
- Create relationships between these new tables and their predecessors through the use of foreign keys.

Consider the following table which has the anomalies:-

crId	StId	stName	fId	room	grade
C3456	S1020	Suhail Dar	F2345	104	B
C5678	S1020	Suhail Dar	F4567	106	
C3456	S1038	Shoaib Ali	F2345	104	A
C5678	S1015	Tahira Ejaz	F4567	106	B

Now the first thing is that the table is in 1NF because there are no duplicate values in any tuple and all cells contain atomic value. The first thing is the redundancy. Like in this table of CLASS the course ID C3456 is being repeated for faculty ID F2345 and similarly the room no 104 is being repeated twice. Second is the insertion anomaly. Suppose we want to insert a course in the table, but this course has not been registered to any student. But we cannot enter the student ID, because no student has registered this course yet. So we can also not insert this course. This is called as insertion anomaly which is wrong state of database. Next is the deletion anomaly. Suppose there is a course which has been enrolled by one student only. Now due to some

reason, we want to delete the record of student. But here the information about the course will also be deleted, so in this way this is the incorrect state of database in which infact we want to delete the information about the student record but along with this the course information has also been deleted. So it is not reflecting the actual system. Now the next is updation anomaly. Suppose a course has been registered by 50 students and now we want to change the class rooms of all the students. So in this case we will have to change the records of all the 50 students. So this is again a deletion anomaly. The process for transforming a 1NF table to 2NF is:

- Identify any determinants other than the composite key, and the columns they determine.
- Create and name a new table for each determinant and the unique columns it determines.
- Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
- Delete the columns you just moved from the original table except for the determinant which will serve as a foreign key.
- The original table may be renamed to maintain semantic meaning.

Now to remove all these anomalies from the table we will have to decompose this table, into different tables as under:

CLASS (crId, stId, stName, fId, room, grade)

crId, stId $\longrightarrow$ stName, fId, room, grade

stId $\longrightarrow$ stName crId $\longrightarrow$ fId, room

Now this table has been decomposed into three tables as under:-

STD (stId, stName)

COURSE (crId, fId, room)

CLASS (crId, stId, grade)

So now these three tables are in second normal form. There are no anomalies available now in this form and we say this as 2NF.

Third Normal Form

A relational table is in third normal form (3NF) if it is already in 2NF and every non-key column is non-transitively dependent upon its primary key. In

other words, all nonkey attributes are functionally dependent only upon the primary key.

Transitive Dependency

Transitive dependency is one that carries over another attribute. Transitive dependency occurs when one non-key attribute determines another non-key attribute. For third normal form we concentrate on relations with one candidate key, and we eliminate transitive dependencies. Transitive dependencies cause insertion, deletion, and update anomalies. We will now see it with an example:-

STD(stId, stName, stAdr, prName, prCrdts)

stId ⟶ stName, stAdr, prName, prCrdts

prName⟶ prCrdts

Now here the table is in second normal form. As there is no partial dependency of any attributes here. The key is student ID . The problem is of transitive dependency in which a non-key attribute can be determined by a non-key attribute. Like here the program credits can be determined by program name, which is not in 3NF. It also causes same four anomalies, which are due to transitive dependencies. For Example:-

STUDENT

stId	stName	stAdr	prName	prCrdts
S1020	Sohail Dar	I-8 Islamabad	MCS	64
S1038	Shoaib Ali	G-6 Islamabad	BCS	132
S1015	Tahira Ejaz	L Rukh Wah	MCS	64
S1015	Tahira Ejaz	L Rukh Wah	MCS	64
S1018	Arif Zia	E-8, Islamabad.	BIT	134

Now in this table all the four anomalies are exists in the table. So we will have to remove these anomalies by decomposing this table after removing the transitive dependency. We will see it as under: -

STD (stId, stName, stAdr, prName, prCrdts)

stId ⟶ stName, stAdr, prName, prCrdts

prName ⟶ prCrdts

The process of transforming a table into 3NF is:

- Identify any determinants, other the primary key, and the columns they determine.
- Create and name a new table for each determinant and the unique columns it determines.
- Move the determined columns from the original table to the new table. The determinate becomes the primary key of the new table.
- Delete the columns you just moved from the original table except for the determinate which will serve as a foreign key.
- The original table may be renamed to maintain semantic meaning.

STD (stId, stName, stAdr, prName)
PROGRAM (prName, prCrdts)
We have now decomposed the relation into two relations of student and program. So the relations are in third normal form and are free of all the anomalies

Boyce - Codd Normal Form

A relation is in Boyce-Codd normal form if and only if every determinant is a candidate key. A relation R is said to be in BCNF if whenever **X -> A** holds in R, and A is not in X, then X is a candidate key for R. It should be noted that most relations that are in 3NF are also in BCNF. Infrequently, a 3NF relation is not in BCNF and this happens only if

(a) the candidate keys in the relation are composite keys (that is, they are not single attributes),
(b) there are more than one candidate keys in the relation, and
(c) the keys are not disjoint, that is, some attributes in the keys are common.

The BCNF differs from the 3NF only when there are more than one candidate keys and the keys are composite and overlapping. Consider for example, the relationship:

enrol (sno, sname, cno, cname, date-enrolled)

Let us assume that the relation has the following candidate keys:

(sno,cno)
(sno,cname)
(sname,cno)
(sname, cname)

(we have assumed sname and cname are unique identifiers). The relation is in 3NF but not in BCNF because there are dependencies

sno -> sname
cno -> cname

Where attributes are part of a candidate key are dependent on part of another candidate key. Such dependencies indicate that although the relation is about some entity or association that is identified by the candidate keys e.g. (sno, cno), there are attributes that are not about the whole thing that the keys identify. For example, the above relation is about an association (enrolment) between students and subjects and therefore the relation needs to include only one identifier to identify students and one identifier to identify subjects. Provided that two identifiers about the students (sno, sname) and two keys about subjects (cno, cname) means that some information about the students and subjects which is not needed is being provided. This provision of information will result in repetition of information and the anomalies that we discussed at the beginning of this chapter. If we wish to include further information about students and courses in the database, it should not be done by putting the information in the present relation but by creating new relations that represent information about entities student and subject.

These difficulties may be overcome by decomposing the above relation in the following three relations:

(sno, sname)
(cno, cname)
(sno, cno, date-of-enrolment)

We now have a relation that only has information about students, another only about subjects and the third only about enrolments. All the anomalies and repetition of information have been removed.

Higher Normal Forms

After BCNF are the fourth, a fifth and domain key normal form exists. Although till BCNF normal form tables are in required form, but if we want we can move on to fourth and fifth normal forms as well. 4NF deals with multivalued dependency, fifth deals with possible loss less decompositions; DKNF reduces further chances of any possible inconsistency.

Summary

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and correctly modified. This means that all tables in a relational database should be in the third normal form (3NF). A relational table is in 3NF if and only if all non-key columns are (a) mutually independent and (b) fully dependent upon the primary key. Mutual independence means that no non-key column is dependent upon any combination of the other columns. The first two normal forms are intermediate steps to achieve the goal of having all tables in 3NF. In order to better understand the 2NF and higher forms, it is necessary to understand the concepts of functional dependencies and loss less decomposition.

Exercise:

The tables of Examination System which were brought in 1NF in previous lecture bring those tables into 2 and 3NF.

Lecture No. 21

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Page 238
"Modern Database Management", Fred McFadden, Jeffrey Hoffer, Benjamin/Cummings	Chapter 6

Overview of Lecture:

- Summary of normalization
- A normalization example
- Introduction to physical DB design phase

Normalization Summary

Normalization is a step by step process to make DB design more efficient and accurate. A normalized database helps the DBA to maintain the consistency of the database. However, the normalization process is not a must, rather it is a strongly recommended activity performed after the logical DB design phase. Not a must means, that the consistency of the database can be maintained even with an un-normalized database design, however, it will make it difficult for the designer. Un-normalized relations are more prone to errors or inconsistencies.

The normalization is based on the FDs. The FDs are not created by the designer, rather they exist in the system being developed and the designer identifies them. Normalization forms exist up to 6NF starting from 1NF, however, for most of the situations 3NF is sufficient. Normalization is performed through Analysis or Synthesis process. The input to the process is the logical database design and the FDs that exist in the system. Each individual table is checked for the normalization considering the relevant FDs; if any normalization requirement for a particular normal form is being violated, then it is sorted out generally by splitting the table. The

process is applied on all the tables of the design hence the database is called to be in a particular normal form.

Normalization Example

In the following an example of normalization process has been discussed. This example is taken from Ricardo book, page 238. The example comprehensively explains the stages of the normalization process. The approach adopted for the normalization is analysis approach, whereby a singe large table is assumed involving all the attributes required in the system. Later, the table is decomposed into smaller tables by considering the FDs existing in the system. As has been discussed before, the FDs have to be identified by the designer they are not described as regular from b y the users. So the example also explains the transforming of real-world scenarios into FDs.

An example table is given containing all the attributes that are used in different applications in the system under study. The table named WORK consists of the attributes:

WORK (projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept, rating)

The purpose of most of the attributes is clear from the name, however, they are explained in the following facts about the system. The facts mentioned in the book are italicized and numbered followed by the explanation.

1- *Each project has a unique name, but names of employees and managers are not unique.*

This fact simply illustrates that values in the projName attribute will be unique so this attribute can be used as identifier if required however the attributes empName, empMgr and projMgr are not unique so they cannot be used as identifiers

2- *Each project has one manager, whose name is stored in projMgr*

The projMgr is not unique as mentioned in 1, however, since there is only one manager for a project and project name is unique, so we can say that if we know the project name we can determine a single project manager, hence the FD

projName → projMgr

3- *Many employees may be assigned to work on each project, and an employee may be assigned to more than one project. The attribute 'hours' tells the number of hours per week that a particular employee is assigned to work on a particular project.*

Since there are many employees working on each project so the projName attribute cannot determine the employee working on a project, same is the case with empId that it cannot determine the particular project an employee is working since one employee is working on many projects. However, if we combine both the empId and projName then we can determine the number of hours that an employee worked on a particular project within a week, so the FD

empId, projName → hours

4- *Budget stores the budget allocated for a project and startDate stores the starting date of a project*

Since the project name is unique, so if we know the project name we can determine the budget allocated for it and also the starting date of the project

projName → budget, startDate

5- *Salary gives the annual salary of the employee*

empId → salary, empName

Although empId has not been mentioned as unique, however, it is generally assumed that attribute describing Id of something are unique, so we can define the above FD.

6- *empMgr gives the name of the employee's manager, who is not the same as project manager.*

Project name is determined by project name, however one employee may work on many projects, so we can not determine the project manager of an employee thourgh the Id of employee. However, empMgr is the manager of employee and can be known from employee Id, so FD in 5 can be extended

empId → salary, empName, empMgr

7- *empDept give the employee's department. Department names are unique. The employee's manager is the manager of the employee's department.*

empDept → empMgr

Because empDept is unique and there is one manager for each department. At the same time, because each employee works in one department, we can also say that

empId → empDept so the FD in 6 is further extended

empId → salary, empName, empMgr, empDept

8- *Rating give the employee's rating for a particular project. The project manager assigns the rating at the end of employee's work on that project*

Like 'hours' attribute, the attribute 'rating' is also determined by two attributes, the projName and empId, because many employees work on one project and one employee may work on many projects. So to know the rating of an employee on a particular project we need to know the both, so the FD

projName, empId → rating

In all we have the following four FDs:

1) empId → salary, empName, empMgr, empDept
2) projName, empId → rating, hours
3) projName → projMgr, budget, startDate
4) empDept → empMgr

Normalization

So we identified the FDs in our example scenario, now to perform the normalization process. For this we have to apply the conditions of the normal forms on our tables. Since we have got just one table to begin with so we start our process on this table:

WORK(projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept, rating)

First Normal Form:
Seeing the data in the example in the book or assuming otherwise that all attributes contain the atomic value, we find out the table is in the 1NF.

Second Normal Form:
Seeing the FDs, we find out that the PK for the table is a composite one, comprising of empId, projName. We did not include the determinant of fourth FD, that is, the empDept, in the PK because empDept is dependent on empId and empID is included in our proposed PK. However, with this PK (empID, projName) we have got partial dependencies in the table through FDs 1 and 3 where we see that some attributes are being determined by subset of our PK which is the violation of the requirement for the 2NF. So we split our table based on the FDs 1 and 3 as follows:

PROJECT (~~projName~~, projMgr, startDate)
EMPLOYEE (~~empId~~, empName, salary, empMgr, empDept)
WORK (~~projName, empId~~, hours, rating)

All the above three tables are in 2NF since they are in 1NF and there is no partial dependency in them.

Third Normal Form

Seeing the four FDs, we find out that the tables are in 2NF and there is no transitive dependency in PROJECT and WORK tables, so these two tables are in 3NF. However, there is a transitive dependency in EMNPLOYEE table since FD 1 say empId → empDept and FD 4 say empDept → empMgr. To remove this transitive dependency we further split the EMPLOYEE table into following two:

EMPLOYEE (~~empId~~, empName, salary, empDept)
DEPT (~~empDept~~, empMgr)
Hence finally we got four tables

PROJECT (projName, projMgr, startDate)
EMPLOYEE (empId, empName, salary, empDept)
WORK (projName, empId, hours, rating)
DEPT (empDept, empMgr)

These four tables are in 3NF based on the given FD, hence the database has been normalized up to 3NF.

Physical Database Design

After completing the logical database design and then normalizing it, we have to establish the physical database design. Throughout the processes of conceptual and logical database designs and the normalization, the primary objective has been the storage efficiency and the consistency of the database. So we have been following good design principles. In the physical database design, however, the focus shifts from storage efficiency to the efficiency in execution. So we deliberately violate some of the rules that we studied earlier, however, this shift in focus should never ever lead to incorrect state of the database. The correctness of the database we have to maintain in any case. When we do not follow the good design principles then it makes it difficult to maintain the consistency or correctness of the database. Since the violation is deliberate, that is, we are aware of the dangers due to violations and we know the reasons for these violations so we have to take care of the possible threats and adopt appropriate measures. Finally, there are different possibilities and we as designers have to adopt particular ones based on certain reasons or objectives. We have to be clear about our objectives.

The physical DB design involves:

- Transforms logical DB design into technical specifications for storing and retrieving data
- Does not include practically implementing the design however tool specific decisions are involved

It requires the following input:

- Normalized relations (the process performed just before)

- Definitions of each attribute (means the purpose or objective of the attributes. Normally stored in some form of data dictionary or a case tool or may be on paper)
- Descriptions of data usage (how and by whom data will be used)
- Requirements for response time, data security, backup etc.
- Tool to be used

Decisions that are made during this process are:

- Choosing data types (precise data types depend on the tool to be used)
- Grouping attributes (although normalized)
- Deciding file organizations
- Selecting structures
- Preparing strategies for efficient access

That is all about today's lecture, the discussion continues in the next lecture.

Summary

In today's lecture we summarized the normalization process and also saw an example to practically implement the process. We have introduced our next topic that is the physical DB design. We will discuss this topic in the lectures to be followed.

Lecture No. 22

Overview of Lecture

- Data Volume and Usage Analysis
- Designing Fields
- Choosing Data Type
- Coding Techniques
- Coding Example
- Controlling Data Integrity

The Physical Database Design Considerations and Implementation

The physical design of the database is one of the most important phases in the computerization of any organization. There are a number of important steps involved in the physical design of the database. Steps are carried out in sequence and need to be performed precisely so that the result of the first step is properly used as input to the next step.
Before moving onto the Physical database design the design of the database should have undergone the following steps,
Normalization of relations
Volume estimate
Definition of each attribute
Description of where and when data is used (with frequencies)
Expectation or requirements of response time and data security.
Description of the technologies.

For the physical database design we need to check the usage of the data in term of its size and the frequency. This critical decision is to be made to ensure that proper structures are used and the database is optimized for maximum performance and efficiency.
The following steps are necessary once we have the prerequisite complete:

Select the appropriate attribute and a corresponding data type for the attribute.
The process of selecting the attribute to be placed in a specific relation in the physical design. Need considerable care as it is one of the most important and basic aspects for the creation of the database.

Grouping of attributes in the logical order so that the relation is created in such a way that no information is missing from the relation and also no redundant or unnecessary information is placed in the relation.
Looking at the logical design at the time of transformation into physical design there may be stages when the information combined logically in the logical design looks odd when transforming the design into a physical one.

Arrangement of Similar records into the secondary memory (hard disk)

The scheme of storage on hard disk is important as it leads to the efficiency and management of the data on disk. Different types of data access mechanism are available and are useful for rapid access, storage, and modification of data.
Different types of database structures can be used for placement of data on disks, management of data in the forms of indexes and different database architecture is vital and leads to better retrieval and recovery of records.

Preparing queries and handling strategies for the proper usage of the database, so that any type of input or output operation performed on the database is executed in an optimized and efficient way.

DESIGNING FIELDS

Field is the smallest unit of application data recognized by system software, such as a programming language or any database management system.
Designing fields in the databases' physical design as discussed earlier is a major issue and needs to be dealt with great care and accuracy. Data types are the structure defined for placing data in the attributes. Each data type is appropriate for use with certain type of data.
4 major objectives for using data types when specifying attributes in a database are given as under:
Minimized usage of storage space
Represent all possible values
Improve data integrity
Support all data manipulation

The correct data type selection and decision for proper domain of the attribute is very necessary as it provides a number of benefits.
Most common data types used in the available DBMS of the day have the following set of common attributes.

Data type	Description	Max Size: PL/SQL
VARCHAR2(size)	Variable length character string having maximum length *size* bytes. You must specify size	32767 bytes minimum is 1
VARCHAR	Now deprecated - VARCHAR is a synonym for VARCHAR2 but this usage may change in future versions.	
CHAR(size)	Fixed length character data of length size bytes. This should be used for fixed length data. Such as codes A100, B102...	32767 bytes Default and minimum size is 1 byte.
NUMBER(p,s)	Magnitude 1E-130 .. 10E125 maximum precision of 126 binary digits, which is roughly equivalent to 38 decimal digits The scale s can range from -84 to 127. For floating point don't specify p,s REAL has a maximum precision of 63 binary digits, which is roughly equivalent to 18 decimal digits	

LONG	Character data of variable length (A bigger version the VARCHAR2 data type)	32760 bytes Note this is smaller than the maximum width of a LONG column
DATE	Valid date range	from January 1, 4712 BC to December 31, **9999** AD. (in Oracle7 = 4712 AD)
RAW(size)	Raw binary data of length size bytes. You must specify size for a RAW value.	32767 bytes
LONG RAW	Raw binary data of variable length. (not interpreted by PL/SQL)	32760 bytes Note this is smaller than the maximum width of a LONG RAW column
BLOB	Binary Large Object	4Gigabytes

CODING AND COMPRESSION TECHNIQUES:

There a re some attributes which have some sparse set of values, these values when they are represented in any data type are hard to express, for this purpose some codes are used. As the codes defined by the database administrator or the programmer consume less space so they are better for use in situations where we have large number of records and wastage of small amount of space in each record can lead to loss of huge amount of data storage space. Thus causing lowered database efficiency.

STID	***STNAME***	***HOBBY***
S1020	Sohail Dar	Reading
S1038	Shoaib Ali	Gardening
S1015	Tahira Ejaz	Reading
S1015	Tahira Ejaz	Movie
S1018	Arif Zia	Reading

Coding techniques are also useful for compression of data values appearing the data, by replacing those data values with the smaller sized codes we can further reduce the space needed by the data for storage in the database.

Following tables give the use of codes and their utilization in the database environment
Coding Example:

Student

STID	***STNAME***	***HOBBY***
S1020	Sohail Dar	R
S1038	Shoaib Ali	G
S1015	Tahira Ejaz	R
S1015	Tahira Ejaz	M
S1018	Arif Zia	R

Hobby Table

CODE	***HOBBY***
R	Reading
G	Gardening
M	Movies

In the above example we have seen the implementation of the codes as replacement to the data in the actual table, here we actually allocated codes to different hobbies and then replace the codes instead of writing the codes in the table.

We get a number of benefits by the use of data types and the benefit can be in a number of dimensions.

Default value

Default values are the values which are associated with a specific attribute and can help us to reduce the chances of inserting incorrect values in the attribute space. And also it can help us preventing the attribute value be left empty.

Range Control

Range control implemented over the data can be very easily achieved by using any data type. As the data type enforces the entry of data in the field according to the limitations of the data type.

Null Value Control

As we already know that a null value is an empty value and is distinct from zero and spaces, Databases can implement the null value control by using the different data types or their build in mechanisms.

Referential Integrity

Referential Integrity means to keep the input values for a specific attribute in specific limits in comparison to any other attribute of the same or any other relation.

Lecture No. 23

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill

Overview of Lecture

Physical Record and De-normalization

Partitioning

In the previous lecture, we have studied different data types and the coding techniques. We have reached now on implementing our database in which relations are now normalized. Now we will make this database efficient form implementation point of view.

Physical Record and Denormalization

Denormalization is a technique to move from higher to lower normal forms of database modeling in order to speed up database access. Denormalization process is applied for deriving a physical data model from a logical form. In logical data base design we group things logically related through same primary key. In physical database design fields are grouped, as they are stored physically and accessed by DBMS. In general it may decompose one logical relation into separate physical records, combine some or do both. There is a valid reason for denormalization that is to enhance the performance. However, there are several indicators, which will help to identify systems, and tables, which are potential denormalization candidates. These are:

Many critical queries and reports exist which rely upon data from more than one table. Often times these requests need to be processed in an on-line environment.

Repeating groups exist which need to be processed in a group instead of individually.

Many calculations need to be applied to one or many columns before queries can be successfully answered.

Tables need to be accessed in different ways by different users during the same timeframe.

Certain columns are queried a large percentage of the time. Consider 60% or greater to be a cautionary number flagging denormalization as an option.

We should be aware that each new RDBMS release usually bring enhanced performance and improved access options that may reduce the need for denormalization. However, most of the popular RDBMS products on occasion will require denormalized data structures. There are many different types of denormalized tables, which can resolve the performance problems caused when accessing fully normalized data. Denormalization must balance the need for good system response time with the need to maintain data, while avoiding the various anomalies or problems associated with denormalized table structures. Denormalization goes hand-in-hand with the detailed analysis of critical transactions through view analysis. View analysis must include the specification of primary and secondary access paths for tables that comprise end-user views of the database. A fully normalized database schema can fail to provide adequate system response time due to excessive table join operations

Denormalization Situation 1:

Merge two Entity types into one with one to one relationship. Even if one of the entity type is optional, so joining can lead to wastage of storage, however if two accessed together very frequently their merging might be a wise decision. So those two relations must be merged for better performance, which have one to one relationship.

Denormalization Situation 2:

Many to many binary relationships mapped to three relations. Queries needing data from two participating ETs need joining of three relations that is expensive. Join is an expensive operation from execution point of view. It takes time and lot of resources. Now suppose there are two relations STUDENT and COURSE and there exits a many to many relationship in between them. So there are three relations STUDENT, COURSE and ENROLLED in between them. Now if we want to see that a student has enrolled how many courses. So to get this we will have to join three relations, first the STUDENT and ENROLLED and then joining it with COURSE, which is quite expensive. The relation created against relationship is merged with one of the relation created against participating ETs. Now the join operation will be performed only once.
Consider the following many to many relationship:-
EMP (empID, eName,pjId,Sal)
PROJ (pjId,pjName)
WORK (empId.pjId,dtHired,Sal)
This is a many to many relationship in between EMP and PROJ with a relationship of WORK. So now if we by de-normalizing these relations and merge the WORK relation with PROJ relation, which is comparatively smaller one. But in this case it is violating 2NF and anomalies of 2NF would be there. But there would be only one join operation involved by joining two tables, which increases the efficiency.
EMP (empID, eName,pjId,Sal)
PROJ (pjId,pjName, empId,dtHired,Sal)
So now it is up to you that you want to weigh the drawbacks and advantages of denormalization.

Denormalization Situation 3:

Reference Data: One to many situation when the ET on side does not participate in any other relationship, then many side ET is appended with reference data rather than the foreign key. In this case the reference table should be merged with the main table.

We can see it with STUDENT and HOBBY relations. One student can have one hobby and one hobby can be adopted by many students. Now in this case the hobby can be merged with the student relation. So in this case although redundancy of data would be there, but there would not be any joining of two relations, which will have a better performance.

Partitioning

De-normalization leads to merging different relations, whereas partitioning splits same relation into two. The general aims of data partitioning and placement in database are to

1. Reduce workload (e.g. data access, communication costs, search space)
2. Balance workload
3. Speed up the rate of useful work (e.g. frequently accessed objects in main memory)

There are two types of partitioning:-

Horizontal Partitioning

Vertical Partitioning

Horizontal Partitioning:
Table is split on the basis of rows, which means a larger table is split into smaller tables. Now the advantage of this is that time in accessing the records of a larger table is much more than a smaller table. It also helps in the maintenance of tables, security, authorization and backup. These smaller partitions can also be placed on different disks to reduce disk contention. Some of the types of horizontal partitioning are as under:-

Range Partitioning:
In this type of partitioning range is imposed on any particular attribute. So in this way different partitions are made on the basis of those ranges with the help of select statement. For Example for those students whose ID is from 1-1000 are in partition 1 and so on. This will improve the overall efficiency of the database. In range partition the partitions may become unbalanced. So in this way few partitions may be overloaded.

Hash Partitioning:
It is a type of horizontal partitioning. In this type particular algorithm is applied and DBMS knows that algorithm. So hash partitioning reduces the chances of unbalanced partitions to a large extent.

List Partitioning:
In this type of partitioning the values are specified for every partition. So there is a specified list for all the partitions. So there is no range involved in this rather there is a list of values.
Summary:
De-normalization can lead to improved processing efficiency. The objective is to improve system response time without incurring a prohibitive amount of additional

data maintenance requirements. This is especially important for client-server systems. Denormalization requires thorough system testing to prove the effect that denormalized table structures have on processing efficiency. Furthermore, unseen ad hoc data queries may be adversely affected by denormalized table structures. Denormalization must be accomplished in conjunction with a detailed analysis of the tables required to support various end-user views of the database. This analysis must include the identification of primary and secondary access paths to data. Similarly before carrying out partitioning of the table thorough analysis of the relations is must.

Exercise:

Critically examine the tables drawn for Examination system and see if there is a requirement of denormalization and partitioning and then carry out the process.

Lecture No. 24

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill

Overview of Lecture

- Vertical Partitioning
- Replication
- Structured Query Language (SQL)

In the previous lecture we were discussing physical data base design, in which we studied denormalization and its different aspects. We also studied the horizontal partitioning. In this lecture we will study vertical partitioning.

Vertical Partitioning

Vertical partitioning is done on the basis of attributes. Same table is split into different physical records depending on the nature of accesses. Primary key is repeated in all vertical partitions of a table to get the original table. In contrast to horizontal partitioning, vertical partitioning lets you restrict which columns you send to other destinations, so you can replicate a limited subset of a table's columns to other machines. We will now see it with the example of a student relation as under: -

STD (stId, sName, sAdr, sPhone, cgpa, prName, school, mtMrks, mtSubs, clgName, intMarks, intSubs, dClg, bMarks, bSubs)

Now in this relation the student relation has number of attributes. It is in 3NF . But the nature of accesses in this relation is different. So now we will partition this relation vertically as under.

STD (stId, sName, sAdr, sPhone, cgpa, prName)

STDACD (sId, school, mtMrks, mtSubs, clgName, intMarks, intSubs, dClg, bMarks, bSubs)

Replication

The process of copying a portion of the database from one environment to another and keeping subsequent copies of the data in synchronization with the original source Changes made to the original source are propagated to the copies of the data in other environments. It is the final form of denormalization. It increases the access speed and decreases failure damage of the database. In replication entire table or part of table can be replicated. Replication is normally adopted in those applications, where updation is not very frequent, because if updation is frequent so then it will have problems of updation in all the copies of database relations. This will also slow down the speed of database.

Clustering Files

Clustering is a process, which means to place records from different tables to place in adjacent physical locations, called clusters. It increases the efficiency since related records are placed close to each other. Clustering is also suitable to relatively static situations. The advantage of clustering is that while accessing the records it is easy to access. Define cluster, define the key of the cluster, and include the tables into the cluster while creating associating the key with it.

Summary of Physical Database Design

Database design is the process of transforming a logical data model into an actual physical database. A logical data model is required before you can even begin to design a physical database. The first step is to create an initial physical data model by transforming the logical data model into a physical implementation based on an understanding of the DBMS to be used for deployment. To successfully create a physical database design you will need to have a good working knowledge of the features of the DBMS including:

- In-depth knowledge of the database objects supported by the DBMS and the physical structures and files required to support those objects.
- Details regarding the manner in which the DBMS supports indexing, referential integrity, constraints, data types, and other features that augment the functionality of database objects.

- Detailed knowledge of new and obsolete features for particular versions or releases of the DBMS to be used.
- Knowledge of the DBMS configuration parameters that are in place.
- Data definition language (DDL) skills to translate the physical design into actual database objects.

Armed with the correct information, you can create an effective and efficient database from a logical data model. The first step in transforming a logical data model into a physical model is to perform a simple translation from logical terms to physical objects. Of course, this simple transformation will not result in a complete and correct physical database design – it is simply the first step. The transformation consists of the following things:

- Transforming entities into tables
- Transforming attributes into columns
- Transforming domains into data types and constraints

There are many decisions that must be made during the transition from logical to physical. For example, each of the following must be addressed:

- The nullability of each column in each table
- For character columns, should fixed length or variable length be used
- Should the DBMS be used to assign values to sequences or identity columns?
- Implementing logical relationships by assigning referential constraints
- Building indexes on columns to improve query performance
- Choosing the type of index to create: b-tree, bit map, reverse key, hash, partitioning, etc.
- Deciding on the clustering sequence for the data
- Other physical aspects such as column ordering, buffer pool specification, data files, denormalization, and so on.

Structured Query Language

SQL is an ANSI standard computer language for accessing and manipulating databases. SQL is standardized, and the current version is referred to as SQL-92. Any SQL-compliant database should conform to the standards of SQL at the time. If not,

they should state which flavor of SQL (SQL-89 for example) so that you can quickly figure out what features are and are not available. The standardization of SQL makes it an excellent tool for use in Web site design. Most Web application development toolkits, most notably Allaire's Cold Fusion and Microsoft's Visual InterDev, rely on SQL or SQL-like statements to connect to and extract information from databases. A solid foundation in SQL makes hooking databases to Web sites all the simpler. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database. This tutorial will provide you with the instruction on the basics of each of these commands as well as allow you to put them to practice using the SQL Interpreter.

Benefits of Standard SQL:
Following are the major benefits of SQL:-

Reduced training cost

- Application portability
- Application longevity
- Reduced dependence on a single vendor
- Cross-system communication**SQL is used for any type of interaction with the database through DBMS. It can be used for creating tables, insertion in the table and deletion as we well and other operations also.**

MS SQL Server

The DBMS for our course is Microsoft’s SQL Server 2000 desktop edition. There are two main tools Query Analyzer and Enterprise Manager; both can be used. For SQL practice we will use Query Analyzer. So you must use this software for the SQL queries. So we will be using this software for our SQL queries.

Summary:

In this lecture we have studied the vertical partitioning, its importance and methods of applying. We have also studied replication and clustering issues. We then started with the basics of SQL and in the next lectures we will use SQL Server for the queries.

Exercise:

Critically examine the tables drawn for Examination system and see if there is a requirement of vertical partitioning and then carry out the process. Also install the SQL Server and acquaint yourself with this software.

Lecture No. 25

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill

Overview of Lecture

- Structured Query Language (SQL)

In the previous lecture we have studied the partitioning and replication of data. From this lecture onwards we will study different rules of SQL for writing different commands.

Rules of SQL Format

SQL, at its simplest, is a basic language that allows you to "talk" to a database and extract useful information. With SQL, you may read, write, and remove information from a database. SQL commands can be divided into two main sub languages. The Data Definition Language (DDL) contains the commands used to create and destroy databases and database objects. After the database structure is defined with DDL, database administrators and users can utilize the Data Manipulation Language to insert, retrieve and modify the data contained within it. Following are the rules for writing the commands in SQL:-

- Reserved words are written in capital like SELECT or INSERT.
- User-defined identifiers are written in lowercase
- Identifiers should be valid, which means that they can start with @,_ alphabets ,or with numbers. The maximum length can be of 256. The reserved words should not be used as identifiers.
- Those things in the command which are optional are kept in [ ]
- Curly braces means required items
- | means choices
- [,…..n] means n items separated by comma

Consider the following example:-

SELECT [ALL|DISTINCT]

{*|select_list}

FROM {table|view[,…n]}

Select * from std

Data Types in SQL Server

In Microsoft SQL Server™, each column, local variable, expression, and parameter has a related data type, which is an attribute that specifies the type of data (integer, character, money, and so on) that the object can hold. SQL Server supplies a set of system data types that define all of the types of data that can be used with SQL Server. The set of system-supplied data types is shown below:-

Integers:

- Biggint

  Integer (whole number) data from -2^{63} (-9,223,372,036,854,775,808) through $2^{63}-1$ (9,223,372,036,854,775,807).

- Int

  Integer (whole number) data from -2^{31} (-2,147,483,648) through $2^{31} - 1$ (2,147,483,647).

- Smallint

  Integer data from -2^{15} (-32,768) through $2^{15} - 1$ (32,767).

- Tinyint

  Integer data from 0 through 255.

bit

Integer data with either a 1 or 0 value.

Decimal and Numeric

- Decimal

  Fixed precision and scale numeric data from $-10^{38} +1$ through $10^{38} -1$.

- Numeric

  Functionally equivalent to decimal.

Text:

It handles the textual data. Following are the different data types.

- Char: By default 30 characters, max 8000

- Varchar: Variable length text, max 8000
- Text: Variable length automatically
- nchar, nvarchar, ntext

Money:

It is used to handle the monetary data

- Small money: 6 digits, 4 decimal
- Money: 15 digits, 4 decimal

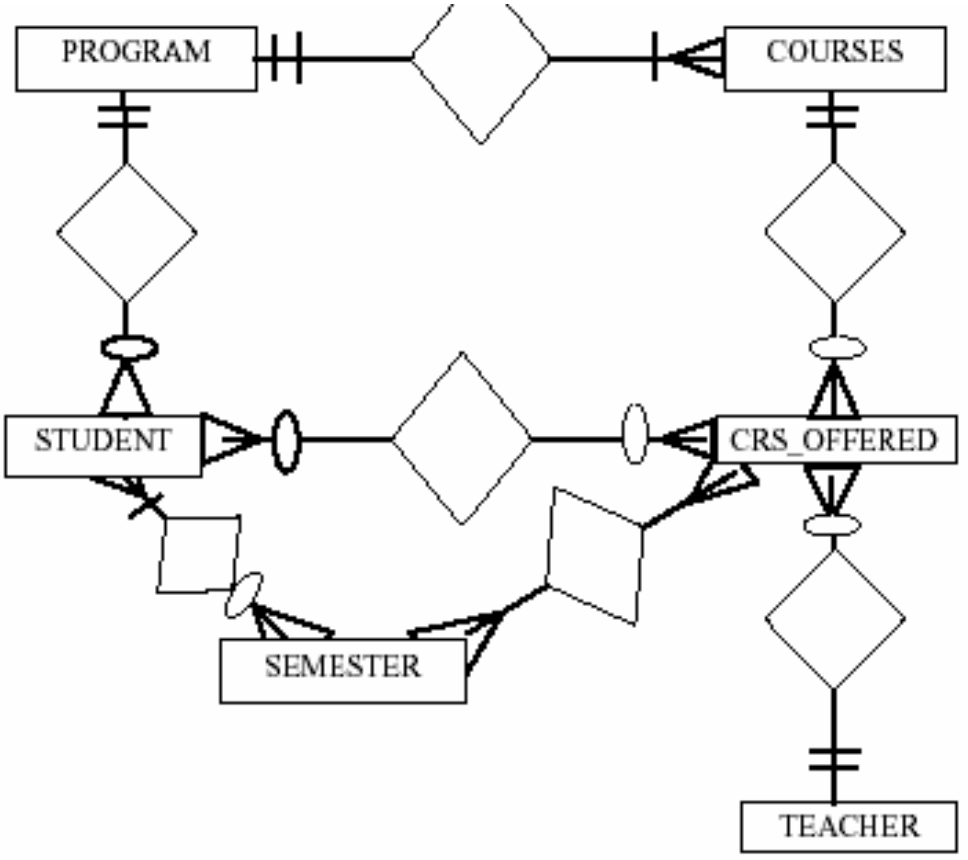
Floating point:

- Float
- Real

Date:

- Smalldatetime
- datetime

Examination System Database



We will now transfer this conceptual database design into relational database design as under:-

PROGRAM (prName, totSem, prCredits)

COURSE (crCode, crName, crCredits, prName)

SEMESTER (semName, stDate, endDate)

CROFRD (crCode, semName, facId)

FACULTY (facId, fName, fQual, fSal, rank)

STUDENT (stId, stName, stFName, stAdres,stPhone, prName, curSem, cgpa)

ENROLL (stId, crCode, semName, mTerm,sMrks, fMrks, totMrks, grade, gp)

SEM_RES (stId, semName, totCrs, totCrdts, totGP, gpa)

It is used to specify a database scheme as a set of definitions expressed in a DDL. DDL statements are compiled, resulting in a set of tables stored in a special file called

a data dictionary or data directory. The data directory contains metadata (data about data) the storage structure and access methods used by the database system are specified by a set of definitions in a special type of DDL called a data storage and definition language

Data Manipulation is retrieval, insertion, deletion and modification of information from the database. A DML is a language, which enables users to access and manipulate data. The goal is to provide efficient human interaction with the system. There are two types of DML.First is Procedural: in which the user specifies what data is needed and how to get it Second is Nonprocedural: in which the user only specifies what data is needed

The category of SQL statements that control access to the data and to the database. Examples are the GRANT and REVOKE statements.

Summary:

In today's lecture we have read about the basics of SQL. It is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database. In the end we have also seen the different types of SQL commands and their functions.

Exercise:

Practice the basic commands of SQL like SELECT, INSERT and CREATE.

Lecture No. 26

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill

Overview of Lecture

- Different Commands of SQL

In the previous lecture we have seen the database of an examination system. We had drawn the ER model and then the relational model, which was normalized. In this lecture we will now start with different commands of SQL.

Categories of SQL Commands

We have already read in our previous lecture that there are three different types of commands of SQL, which are DDL, DML and DCL. We will now study DDL.

DDL

It deals with the structure of database.The DDL (Data Definition Language) allows specification of not only a set of relations, but also the following information for each relation:

- The schema for each relation.
- The domain of values associated with each attribute.
- Integrity constraints.
- The set of indices for each relation.
- Security and authorization information.
- Physical storage structure on disk.

Following are the three different commands of DDL:-

Create

The first data management step in any database project is to create the database. This task can range from the elementary to the complicated, depending on your needs and the database management system you have chosen. Many modern systems (including Personal Oracle7) include graphical tools that enable you to completely build the database with the click of a mouse button. This timesaving feature is certainly helpful, but you should understand the SQL statements that execute in response to the mouse clicks. This command is used to create a new database table. The table is created in

the current default database. The name of the table must be unique to the database. The name must begin with a letter and can be followed by any combination of alphanumeric characters. The name is allowed to contain the underscore character (_). This command can be used to create permanent disk-based or temporary in-memory database tables. Data stored in a temporary table is lost when the server is shutdown. To create a temporary table the "**AS TEMP**" attribute must be specified. Note that querying against a temporary in-memory table is generally faster than querying against a disk-based table. This command is non-transactional. If no file size is given for a disk-based table, the table will be pre-allocated to 1MB. If no filegrowth is given, the default is 50%. It is used to create new tables, fields, views and indexes. It is used to create database. The format of statement is as under:
CREATE DATABASE db_name

For Example CREATE DATABASE EXAM. So now in this example database of exam has been created. Next step is to create tables. There are two approaches for creating the tables, which are:

- Through SQL Create command
- Through Enterprise Manager

Create table command is used to:

- Create a table
- Define attributes of the table with data types
- **Define different constraints on attributes, like primary and foreign keys, check constraint, not null, default value etc.**

The format of create table command is as under:

```
CREATE                                                    TABLE
  [     database_name.[     owner     ]     .     |     owner.     ]     table_name
  (               {               <               column_definition               >
    |           column_name           AS           omputed_column_expression
    | < table_constraint >
      }
      | [ { PRIMARY KEY | UNIQUE } [ ,...n ] ]
```

Let us now consider the CREATE statement used to create the Airport table definition for the Airline Database.

CREATE TABLE Airport

(airport char(4) not null,

name varchar(20),

checkin varchar(50),

resvtns varchar(12),

flightinfo varchar(12));

Table Name.(Airport)

The name chosen for a table must be a valid name for the DBMS.

Column Names. (Airport, Name, ..., FlightInfo)

The names chosen for the columns of a table must also be a valid name for the DBMS.

Data Types

Each column must be allocated an appropriate data type. In addition, key columns, i.e. columns used to uniquely identify individual rows of a given table, may be specified to be NOT NULL. The DBMS will then ensure that columns specified as NOT NULL always contain a value.

The column definition is explained as under:

< column_definition > ::= { *column_name data_type* }
[DEFAULT *constant_expression*]
[< column_constraint >] [...*n*]

The column constraint is explained as under:

< column_constraint > ::= [CONSTRAINT constraint_name]
{ [NULL | NOT NULL]
| [{ PRIMARY KEY | UNIQUE }
]
| [[FOREIGN KEY]
REFERENCES ref_table [**(** ref_column **)**]

[ON DELETE { CASCADE | NO ACTION }]
[ON UPDATE { CASCADE | NO ACTION }]
]

| CHECK(logical_expression)

}

)

We will now see some examples of CREATE command. This is a very simple command for creating a table.

```
CREATE TABLE Program (
        prName char(4),
        totSem tinyint,
        prCredits smallint)
```

If this command is to written in SQL Server, it will be written in Query Analyzer. We will now see an example in which has more attributes comparatively along with different data types:

```
CREATE TABLE Student
      (stId char(5),
       stName  char(25),
       stFName char(25),
       stAdres text,
       stPhone char(10),
        prName char(4)
       curSem  smallint,
       cgpa real)
```

In this example there are more attributes and different data types are also there. We will now see an example of creating a table with few constraints:

```
CREATE TABLE Student (
      stId  char(5)  constraint  ST_PK  primary  key   constraint ST_CK check (stId
      like‘S[0-          9][0-9][0-9][0-9]'),
       stName  char(25) not null,
       stFName char(25),
       stAdres text,
       stPhone char(10),
        prName char(4),
        curSem  smallint default 1,
         cgpa real)
```

Every constraint should be given a meaningful name as it can be referred later by its name. The check constraint checks the values for any particular attribute. In this way different types of constraints can be enforced in any table by CREATE command.

Summary

Designing a database properly is extremely important for the success of any application. In today's lecture we have seen the CREATE command of SQL. How different constraints are applied on this command with the help of different examples. This is an important command and must be practiced as it is used to create database and different tables. So create command is part of DDL.

Exercise:

Create a database of Exam System and create table of student with different constraints in SQL Server.

Lecture No. 27

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill
"Teach Yourself SQL in 21 Days", Second Edition Que Series.

Overview of Lecture

Data Manipulation Language

In the previous lecture we were studying DDL in which we studied the CREATE command along with different examples. We also saw different constraints of create command. In this lecture we will study the ALTER and other SQL commands with examples.

Alter Table Statement

The purpose of ALTER statement is to make changes in the definition of a table already created through Create statement. It can add, and drop the attributes or constraints, activate or deactivate constraints. It modifies the design of an existing table. The format of this command is as under:

Syntax

ALTER TABLE table {

ADD [COLUMN] column type [(size)] [DEFAULT default] |

ALTER [COLUMN] column type [(size)] [DEFAULT default] |

ALTER [COLUMN] column SET DEFAULT default |

DROP [COLUMN] column |

RENAME [COLUMN] column TO columnNew

}

The ALTER TABLE statement has these parts:

Part	Description
Table	The name of the table to be altered.

Column	The name of the column to be altered or added to or deleted from table.
ColumnNew	The new name of the altered column
Type	The data type of column.
Size	The size of the altered column in characters or bytes for text or binary columns.
Default	An expression defining the new default value of the altered column. Can contain literal values, and functions of these values

Using the ALTER TABLE statement, we can alter an existing table in several ways. We can:

- Use ADD COLUMN to add a new column to the table. Specify the name, data type, an optional size, and an optional default value of the column.
- Use ALTER COLUMN to alter type, size or default value of an existing column.
- Use DROP COLUMN to delete a column. Specify only the name of the column.
- Use RENAME COLUMN to rename an existing column. We cannot add, delete or modify more than one column at a time.We will now see an example of alter command

```
ALTER TABLE Student
    add constraint fk_st_pr
    foreign key  (prName) references
                        Program (prName)
```

This is a simple example, in which we have incorporated a constraint and the names are meaningful, so that if in the future we have to refer them, we can do so. We will now see an example of removing or changing attribute.

```
ALTER TABLE student
      ALTER COLUMN stFName char (20)
ALTER TABLE student
     Drop column curSem
ALTER TABLE student
    Drop constraint ck_st_pr
```

Now in these examples either an attribute is deleted or altered by using the keywords of Drop and Alter. We will now see an example in which few or all rows will be removed, or whole table is required to be removed. The TRUNCATE is used to delete all the rows of any table but rows would exist. The DELETE is used to delete one or many records. If we want to remove all records we must use TRUNCATE. Next is the DROP table command, which is used to drop the complete table from the database.

```
TRUNCATE TABLE table_name
Truncate table class
```

Delete can also be used
DROP TABLE table_name

Data Manipulation Language
The non-procedural nature of SQL is one of the principle characteristics of all 4GLs - Fourth Generation Languages - and contrasts with 3GLs (eg, C, Pascal, Modula-2, COBOL, etc) in which the user has to give particular attention to how data is to be accessed in terms of storage method, primary/secondary indices, end-of-file conditions, error conditions (eg, Record NOT Found), and so on. The Data Manipulation Language (DML) is used to retrieve, insert and modify database information. Data Manipulation is retrieval, insertion, deletion and modification of information from the database. SQL is a non-procedural language that is, it allows the user to concentrate on specifying what data is required rather than concentrating on the how to get it. There are two types of DML.First is procedural in which: the user specifies what data is needed and how to get it. Second is nonprocedural in which the user only specifies what data is needed. The DML component of SQL comprises of following basic statements:
Insert To add new rows to tables.
Select To retrieve rows from tables
Update To modify the rows of tables

Insert
The INSERT command in SQL is used to add records to an existing table. We will now see the format of insert command as under:

```
INSERT [INTO] table
    {[ ( column_list ) ]
     { VALUES
        ( { DEFAULT | NULL | expression } [ ,...n] )
      }
  }
  | DEFAULT VALUES
```

The basic format of the INSERT...VALUES statement adds a record to a table using the columns you give it and the corresponding values you instruct it to add. You must follow three rules when inserting data into a table with the INSERT...VALUES statement:
The values used must be the same data type as the fields they are being added to.
The data's size must be within the column's size. For instance, you cannot add an 80-character string to a 40-character column.
The data's location in the VALUES list must correspond to the location in the column list of the column it is being added to. (That is, the first value must be entered into the first column, the second value into the second column, and so on.)
The rules mentioned above must be followed. We will see the examples of the insert statement in the coming lectures.
Summary
SQL provides three statements that can be used to manipulate data within a database. The INSERT statement has two variations. The INSERT...VALUES statement inserts a set of values into one record. The INSERT...SELECT statement is used in combination with a SELECT statement to insert multiple records into a table based on the contents of one or more tables. The SELECT statement can join multiple tables, and the results of this

join can be added to another table. The UPDATE statement changes the values of one or more columns based on some condition. This updated value can also be the result of an expression or calculation.

The DELETE statement is the simplest of the three statements. It deletes all rows from a table based on the result of an optional WHERE clause. If the WHERE clause is omitted, all records from the table are deleted. Modern database systems supply various tools for data manipulation. Some of these tools enable developers to import or export data from foreign sources. This feature is particularly useful when a database is upsized or downsized to a different system. Microsoft Access, Microsoft and Sybase SQL Server, and Personal Oracle7 include many options that support the migration of data between systems.

Exercise:

Try inserting values with incorrect data types into a table. Note the errors and then insert values with correct data types into the same table.

Lecture No. 28

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill
"Teach Yourself SQL in 21 Days", Second Edition Que Series.

In the previous lecture we started the data manipulation language, in which we were discussing the Insert statement, which is used to insert data in an existing table. In today's lecture we will first see an example of Insert statement and then discuss the other SQL Commands.

The INSERT statement allows you to insert a single record or multiple records into a table. It has two formats:

INSERT INTO table-1 [(column-list)] VALUES (value-list)

And,

INSERT INTO table-1 [(column-list)] (query-specification)

The first form inserts a single row into table-1 and explicitly specifies the column values for the row. The second form uses the result of query-specification to insert one or more rows into table-1. The result rows from the query are the rows added to the insert table. Both forms have an optional column-list specification. Only the columns listed will be assigned values. Unlisted columns are set to null, so unlisted columns must allow nulls. The values from the VALUES Clause (first form) or the columns from the query-specification rows (second form) are assigned to the corresponding column in column-list in order. If the optional column-list is missing, the default column list is substituted. The default column list contains all columns in table-1 in the order they were declared in CREATE TABLE.

The VALUES Clause in the INSERT Statement provides a set of values to place in the columns of a new row. It has the following general format:

VALUES (value-1 [, value-2]...)

Value-1 and value-2 are Literal Values or Scalar Expressions involving literals. They can also specify NULL. The values list in the VALUES clause must match the

explicit or implicit column list for INSERT in degree (number of items). They must also match the data type of corresponding column or be convertible to that data type.

We will now see an example of INSERT statement for that we have the table of COURSE with following attributes: -

COURSE (crCode, crName, crCredits, prName)

The INSERT statement is as under:

INSERT INTO course VALUES ('CS-211', 'Operating Systems', 4, 'MCS')

This is a simple INSERT statement; we have not used the attribute names because we want to enter values for all the attributes. So here it is important to enter the values according to the attributes and their data types. We will now see an other example of insert statement:

INSERT INTO course (crCode, crName) VALUES ('CS-316', Database Systems')

In this example we want to enter the values of only two attributes, so it is important that other two attributes should not be NOT NULL. So in this example we have entered values of only two particular attributes. We will now see another example of INSERT statement as under:

INSERT INTO course ('MG-103', 'Intro to Management', NULL, NULL)

In this example we have just entered the values of first two attributes and rest two are NULL. So here we have not given the attribute names and just placed NULL in those values.

Select Statement

Select statement is the most widely used SQL Command in Data Manipulation Language. It is not only used to select rows but also the columns. The SQL SELECT statement queries data from tables in the database. The statement begins with the SELECT keyword. The basic SELECT statement has 3 clauses:

- SELECT
- FROM
- WHERE

The SELECT clause specifies the table columns that are retrieved. The FROM clause specifies the tables accessed. The WHERE clause specifies which table rows are used. The WHERE clause is optional; if missing, all table rows are used. The SELECT clause is mandatory. It specifies a list of columns to be retrieved from the tables in the FROM clause. The FROM clause always follows the SELECT clause. It lists the tables accessed by the query. The WHERE clause is optional. When specified, it always follows the FROM clause. The WHERE clause filters rows from the FROM clause tables. Omitting the WHERE clause specifies that all rows are used. The syntax for the SELECT statement is:

SELECT {*|col_name[,….n]} FROM table_name

This is the simplest form of SELECT command. In case of * all the attributes of any table would be available. If we do not mention the * then we can give the names of particular attribute names. Next is the name of the table from where data is required. We will now see different examples of SELECT statement using the following table:
STUDENT

stId	**stName**	**prName**	**cgpa**
S1020	Sohail Dar	MCS	2.8
S1038	Shoaib Ali	BCS	2.78
S1015	Tahira Ejaz	MCS	3.2
S1034	Sadia Zia	BIT	
S1018	Arif Zia	BIT	3.0

So the first query is
Q: Get the data about studentsSELECT * FROM students
The output of this query is as under:

	stId	**stName**	**prName**	**cgpa**
1	S1020	Sohail Dar	MCS	2.8
2	S1038	Shoaib Ali	BCS	2.78
3	S1015	Tahira Ejaz	MCS	3.2
4	S1034	Sadia Zia	BIT	
5	S1018	Arif Zia	BIT	3.0

We will now see another query, in which certain specific data is required form the table: The query is as under:
Q: Give the name of the students with the program nameThe SQL Command for the query is as under:

SELECT stName, prName
FROM student
The output for the command is as under:

	stName	**prName**
1	Sohail Dar	MCS

2	Shoaib Ali	BCS
3	Tahira Ejaz	MCS
4	Sadia Zia	BIT
5	Arif Zia	BIT

Attribute Allias

SELECT {*|col_name [[AS] alias] [, …n]} FROM tab_name

Now in this case if all the attributes are to be selected by * then we cannot give the name of attributes. The AS is also optional here then we can write the name of attribute what we want. We will now see an example.

SELECT stName as 'Student Name', prName 'Program' FROM Student

The output of this query will be as under:

	Student Name	**Program**
1	Sohail Dar	MCS
2	Shoaib Ali	BCS
3	Tahira Ejaz	MCS
4	Sadia Zia	BIT
5	Arif Zia	BIT

In the column list we can also give the expression; value of the expression is computed and displayed. This is basically used where some arithmetic operation is performed, in which that operation is performed on each row and then that result is displayed as an output. We will now see it with an example:
Q Display the total sessional marks of each student obtained in each subject

The SQL Command for the query will be as under:

Select stId, crCode, mTerm + sMrks 'Total out of 50' from enroll

The DISTINCT keyword is used to return only distinct (different) values. The SELECT statement returns information from table columns. But what if we only want to select distinct elements With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement. The format is as under:

SELECT DISTINCT column_name(s)

FROM table_name

We will now see it with an example

Q Get the program names in which students are enrolled

The SQL Command for this query is as under:

SELECT DISTINCT prName FROM Student

	programs
1	BCS
2	BIT
3	MCS
4	MBA

The "WHERE" clause is optional. When specified, it always follows the FROM clause. The "WHERE" clause filters rows from "FROM" clause tables. Omitting the WHERE clause specifies that all rows are used. Following the WHERE keyword is a logical expression, also known as a predicate. The predicate evaluates to a SQL logical value -- true, false or unknown. The most basic predicate is a comparison:

Color = 'Red'

This predicate returns:

- True -- If the color column contains the string value -- 'Red',
- False -- If the color column contains another string value (not 'Red'), or
- Unknown -- If the color column contains null.

Generally, a comparison expression compares the contents of a table column to a literal, as above. A comparison expression may also compare two columns to each other. Table joins use this type of comparison.

In today's we have studied the SELECT statement with different examples. The keywords SELECT and FROM enable the query to retrieve data. You can make a broad statement and include all tables with a SELECT * statement or you can rearrange or retrieve specific tables. The keyword DISTINCT limits the output so that you do not see duplicate values in a column. In the coming lecture we will see further SQL Commands of Data Manipulation Language.

Lecture No. 29

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill
"Teach Yourself SQL in 21 Days", Second Edition Que Series.

Overview of Lecture

Data Manipulation Language

In the previous lecture we have studied the SELECT statement, which is the most widely used SQL statement. In this lecture we will study the WHERE clause. This is used to select certain specific rows.

The WHERE clause allows you to filter the results from an SQL statement - select, insert, update, or delete statement. The rows which satisfy the condition in the where clause are selected. The format of WHERE clause is as under:

SELECT [ALL|DISTINCT]

{*|column_list [alias][,…..n]} FROM table_name

[WHERE <search_condition>]

Here WHERE is given in square brackets, which means it is optional. We will see the search condition as under:

Search Condition

{ [NOT] < predicate > | (< search_condition >) }

[{ AND | OR } [NOT] { < predicate > |

(< search_condition >) }]

} [,...n]

< predicate > ::=

{ expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }

```
                    expression
    | string_expression [ NOT ] LIKE string_expression
    | expression [ NOT ] BETWEEN expression AND
          expression
    | expression IS [ NOT ] NULL
    | expression [ NOT ] IN ( subquery | expression [ ,...n ] )
    | expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }
      { ALL | SOME | ANY} ( subquery )
    | EXISTS ( subquery )
}
```

In this format where clause is used in expressions using different comparison operators. Those rows, which fulfill the condition, are selected in the output.

```
SELECT  *
FROM supplier
WHERE supplier_name = 'IBM';
```

In this first example, we have used the WHERE clause to filter our results from the supplier table. The SQL statement above would return all rows from the supplier table where the supplier_name is IBM. Because the * is used in the select, all fields from the supplier table would appear in the result set. We will now see another example of where clause.

```
SELECT supplier_id
FROM supplier
WHERE supplier_name = 'IBM'
or supplier_city = 'Karachi';
```

We can define a WHERE clause with multiple conditions. This SQL statement would return all supplier_id values where the supplier_name is IBM or the supplier_city is Karachi..

```
SELECT supplier.suppler_name, orders.order_id
FROM supplier, orders
WHERE supplier.supplier_id = orders.supplier_id
and supplier.supplier_city = 'Karachi';
```

We can also use the WHERE clause to join multiple tables together in a single SQL statement. This SQL statement would return all supplier names and order_ids where there is a matching record in the supplier and orders tables based on supplier_id, and where the supplier_city is Karachi.

We will now see a query in which those courses, which are part of MCS, are to be displayed

Q: Display all courses of the MCS program

Select crCode, crName, prName from course

where prName = 'MCS

Now in this query whole table would be checked row by row and where program name would be MCS would be selected and displayed.'

Q List the course names offered to programs other than MCS

SELECT crCode, crName, prName

FROM course

WHERE not (prName = 'MCS')

Now in this query again all the rows would be checked and those courses would be selected and displayed which are not for MCS. So it reverses the output.

The BETWEEN condition allows you to retrieve values within a specific range.

The syntax for the BETWEEN condition is:

```
SELECT columns
FROM tables
WHERE column1 between value1 and value2;
```

This SQL statement will return the records where column1 is within the range of value1 and value2 (inclusive). The BETWEEN function can be used in any valid SQL statement - select, insert, update, or delete. We will now see few examples of this operator.

```
SELECT *
FROM suppliers
WHERE supplier_id between 10 AND 50;
```

This would return all rows where the supplier_id is between 10 and 50.

The BETWEEN function can also be combined with the NOT operator.

For example,

```
SELECT *
FROM suppliers
WHERE supplier_id not between 10 and 50;
```

The IN function helps reduce the need to use multiple OR conditions. It is sued to check in a list of values. The syntax for the IN function is:

```
SELECT columns
FROM tables
WHERE column1 in (value1, value2,.... value_n);
```

This SQL statement will return the records where column1 is value1, value2... or value_n. The IN function can be used in any valid SQL statement - select, insert, update, or delete. We will now see an example of IN operator.

```
SELECT crName, prName
From course
Where prName in ('MCS', 'BCS')
```

It is equal to the following SQL statement

```
SELECT crName, prName
From course
Where (prName = 'MCS') OR (prName = 'BCS')
```

Now in these two queries all the rows will be checked for MCS and BCS one by one so OR can be replaced by IN operator.

The LIKE operator allows you to use wildcards in the where clause of an SQL statement. This allows you to perform pattern matching. The LIKE condition can be used in any valid SQL statement - select, insert, update, or delete.

The patterns that you can choose from are:

% Allows you to match any string of any length (including zero length)

_ Allows you to match on a single character

We will now see an example of LIKE operator

Q: Display the names and credits of CS programs

```
SELECT crName, crCrdts, prName FROM course
WHERE prName like '%CS'
```

The ORDER BY clause allows you to sort the records in your result set. The ORDER BY clause can only be used in SELECT statements.

The syntax for the ORDER BY clause is:

```
SELECT columns
FROM tables
WHERE predicates
ORDER BY column ASC/DESC;
```

The ORDER BY clause sorts the result set based on the columns specified. If the ASC or DESC value is omitted, the system assumed ascending order.

ASC indicates ascending order. (Default)

DESC indicates descending order.

We will see the example of ORDER BY clause in our next lecture.

In today's lecture we have discussed different operators and use of WHERE clause which is the most widely used in SQL Commands. These different operators are used according to requirements of users. We will study rest of the SQL Commands in our coming lectures.

Lecture No. 30

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill

Overview of Lecture

Data Manipulation Language
Functions in SQL

In the previous lecture we have discussed different operators of SQL, which are used in different commands. By the end of previous lecture we were discussing ORDER BY clause, which is basically used to bring the output in ascending or descending order. In this lecture we will see some examples of this clause.

ORDER BY Clause

The ORDER BY clause allows you to sort the records in your result set. The ORDER BY clause can only be used in SELECT statements. The ORDER BY clause sorts the result set based on the columns specified. If the ASC or DESC value is omitted, the system assumed ascending order. We will now see few examples of this clause

```
SELECT supplier_city
FROM supplier
WHERE supplier_name = 'IBM'
ORDER BY supplier_city;
```

This would return all records sorted by the supplier_city field in ascending order.

```
SELECT supplier_city
FROM supplier
```

```
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC;
```

This would return all records sorted by the supplier_city field in descending order.

Functions in SQL

A function is a special type of command. Infact, functions are one-word command that return a single value. The value of a function can be determined by input parameters, as with a function that averages a list of database values. But many functions do not use any type of input parameter, such as the function that returns the current system time, CURRENT_TIME. There are normally two types of functions. First is Built in, which are provided by any specific tool or language. Second is user defined, which are defined by the user. The SQL supports a number of useful functions.. In addition, each database vendor maintains a long list of their own internal functions that are outside of the scope of the SQL standard.

Categories of Functions:

These categories of functions are specific to SQL Server. Depending on the arguments and the return value, functions are categorized as under:

- Mathematical (ABS, ROUND, SIN, SQRT)
- String (LOWER, UPPER, SUBSTRING, LEN)
- Date (DATEDIFF, DATEPART, GETDATE ())
- System (USER, DATALENGTH, HOST_NAME)
- Conversion (CAST, CONVERT)

We will now see an example using above-mentioned functions:

```
SELECT upper (stName), lower (stFName), stAdres, len(convert(char, stAdres)),
  FROM student
```

In this example student name will be displayed in upper case whereas father name will be displayed in lower case. The third function is of getting the length of student address. It has got nesting of functions, first address is converted into character and then its length will be displayed.

Aggregate Functions

These functions operate on a set of rows and return a single value. If used among many other expressions in the item list of a SELECT statement, the SELECT must

have a GROUP BY clause. No GROUP BY clause is required if the aggregate function is the only value retrieved by the SELECT statement. Following are some of the aggregate functions:

Function	Usage
AVG(expression)	Computes average value of a column by the expression
COUNT(expression)	Counts the rows defined by the expression
COUNT(*)	Counts all rows in the specified table or view
MIN(expression)	Finds the minimum value in a column by the expression
MAX(expression)	Finds the maximum value in a column by the expression
SUM(expression)	Computes the sum of column values by the expression

```
SELECT avg(cgpa) as 'Average CGPA', max(cgpa) as 'Maximum CGPA'
 from student
```

GROUP BY Clause

The GROUP BY clause can be used in a SELECT statement to collect data across multiple records and group the results by one or more columns. It is added to SQL because aggregate functions (like SUM) return the aggregate of all column values every time they are called, and without the GROUP BY function it is impossible to find the sum for each individual group of column values.

The syntax for the GROUP BY clause is:

```
SELECT column1, column2, ... column_n, aggregate_function
(expression)
FROM tables
WHERE predicates
GROUP BY column1, column2, ... column_n;
```

Aggregate function can be a function such as SUM, COUNT, MIN or MAX

Example using the SUM function
For example, the SUM function can be used to return the name of the department and the total sales (in the associated department).

```
SELECT department, SUM (sales) as "Total sales"
FROM order_details
GROUP BY department;
```

In this example we have listed one column in the SELECT statement that is not encapsulated in the SUM function, so we have used a GROUP BY clause. The department field must, therefore, be listed in the GROUP BY section.

Example using the COUNT function
We can also use the COUNT function to return the name of the department and the number of employees (in the associated department) that make over Rs 25,000 / year.

```
SELECT department, COUNT (*) as "Number of employees"
FROM employees
WHERE salary > 25000
GROUP BY department;
```

HAVING Clause

The HAVING clause is used in combination with the GROUP BY clause. It can be used in a SELECT statement to filter the records that a GROUP BY returns. At times we want to limit the output based on the corresponding sum (or any other aggregate functions). For example, we might want to see only the stores with sales over Rs 1,500. Instead of using the WHERE clause in the SQL statement, though, we need to use the HAVING clause, which is reserved for aggregate functions. The HAVING clause is typically placed near the end of the SQL statement, and a SQL statement with the HAVING clause may or may not include the GROUP BY clause. The syntax for the HAVING clause is:

```
SELECT column1, column2, ... column_n, aggregate_function
(expression)
FROM tables
WHERE predicates
GROUP BY column1, column2, ... column_n
HAVING condition1 ... condition_n;
```

Aggregate function can be a function such as SUM, MIN or MAX.

We will now see few examples of HAVING Clause.

Example using the SUM function

We can use the SUM function to return the name of the department and the total sales (in the associated department). The HAVING clause will filter the results so that only departments with sales greater than Rs 1000 will be returned.

```
SELECT department, SUM (sales) as "Total sales"
FROM order_details
GROUP BY department
HAVING SUM (sales) > 1000;
```

Example using the COUNT function

For example, you could use the COUNT function to return the name of the department and the number of employees (in the associated department) that make over $25,000 / year. The HAVING clause will filter the results so that only departments with at least 25 employees will be returned.

```
SELECT department, COUNT (*) as "Number of employees"
FROM employees
WHERE salary > 25000
GROUP BY department
HAVING COUNT (*) > 10;
```

Accessing Multiple Tables:

Until now we have been accessing data through one table only. But there can be occasions where we have to access the data from different tables. So depending upon different requirements data can be accessed from different tables. Referential integrity constraint plays an important role in gathering data from multiple tables. Following are the methods of accessing data from different tables:

Cartesian Product

- Inner join
- Outer Join
- Full outer join
- Semi Join
- Natural JoinWe will now discuss them one by one.

Cartesian product:

A Cartesian join gives a Cartesian product. A Cartesian join is when you join every row of one table to every row of another table. You can also get one by joining every row of a table to every row of itself. No specific command is used just Select is used to join two tables. Simply the names of the tables involved are given and Cartesian product is produced. It produces m x n rows in the resulting table. We will now see few examples of Cartesian product.

Select * from program, course

Now in this example all the attributes of program and course are selected and the total number of rows would be number of rows of program x number of rows of course.In Cartesian product certain columns can be selected, same column name needs to be

qualified. Similarly it can be applied to more than one table, and even can be applied on the same table .For Example

SELECT * from Student, class, program

Summary

In today's lecture we have seen certain important functions of SQL, which are more specific to SQL Server. We studied some mathematical, string and conversion functions, which are used in SQL Commands. We also studied Aggregate functions, which are applied on a entire table or a set of rows and return one value. We also studied Group By clause which is used in conjunction with aggregate functions. In the end we saw how to extract data from different tables and in that we studied Cartesian product. We will see rest of the methods in our coming lectures.

Lecture No. 31

Reading Material

Raghu Ramakrishnan, Johannes Gehkre, 'Database Management Systems', Second edition	Chapter 17

Overview of Lecture

- Types of Joins
- Relational Calculus
- Normalization

In the previous lecture we studied that rows from two tables can be merged with each other using the Cartesian product. In real life, we very rarely find a situation when two tables need to be merged the way Cartesian product, that is, every row of one table is merged with every row of the other table. The form of merging that is useful and used most often is 'join'. In the following, we are going to discuss different forms of join.

Inner Join

Only those rows from two tables are joined that have same value in the common attribute. For example, if we have two tables R and S with schemes

R (a, b, c, d) and S (f, r, h, a), then we have 'a' as common attribute between these twit tables. The inner join between these two tables can be performed on the basis of 'a' which is the common attribute between the two. The common attributes are not required to have the same name in both tables, however, they must have the same domain in both tables. The attributes in both tables are generally tied in a primary-foreign key relationship but that also is not required. Consider the following two tables:

	crCode	crName	crCrdts	prName
1	CS-105	Intro to Programmin...	4	BCS
2	CS-216	Data Structures	4	BCS
3	CS-316	Database Systems	4	BCS
4	CS-504	Analysis of Algorit...	4	MCS
5	CS-511	Operating System	4	MCS
6	CS-516	Data Structures and...	4	MCS
7	CS-616	Intro to Database S...	3	MCS
8	MG-103	Intro to Management...	NULL	NULL
9	MG-105	Intro to Accounting...	3	BBA
10	MG-314	Money & Capital Mar...	3	BIT
11	MG-505	Intro to Accounting...	3	MBA
12	MT-305	Linear Algebra	3	MCS

	prName	totSem	prCredits
1	BBA	8	130
2	BCS	8	134
3	BIT	8	132
4	MBA	4	64
5	MCS	4	64

Fig. 1: COURSE and PROGRAM tables with common attribute prName

The figure shows two tables, COURSE and PROGRAM. The COURSE.prName and PROGRAM. prName are the common attributes between the two tables; incidentally the attributes have the same names and definitely the same domains. If we apply inner join on these tables, the rows from both tables will be merged based on the values of common attribute, that is, the prName. Like, row one of COURSE has the value 'BCS' in attribute prName. On the other hand, row number 2 in PROGRAM table has the value 'BCS'. So these two rows will merge and form one row of the resultant table of the inner join operation. As has been said before, the participating tables of inner join are generally tied in a primary-foreign key link, so the common attribute is PK in one of the tables. It means the table in which the common attribute is FK, the rows from this table will not be merged with more that one row from the other table. Like in the above example, each row from COURSE table will find exactly one match in PROGRAM table, since the prName is the PK in PROGRAM table.

The inner join can be implemented using different techniques. One possibility is that we may find 'inner join' operation as such, like:

- **SELECT * FROM course INNER JOIN program ON course.prName = program.prName**

or

- **Select * FROM Course c INNER JOIN program p ON c.prName = p.prName**

The output after applying inner join on tables of figure 1 will be as follows:

	crCode	crName	crCrdts	prName	prName	totSem	prCredits
1	CS-105	Intro to Programming	4	BCS	BCS	8	134
2	CS-216	Data Structures	4	BCS	BCS	8	134
3	CS-316	Database Systems	4	BCS	BCS	8	134
4	CS-504	Analysis of Algorithm	4	MCS	MCS	4	64
5	CS-511	Operating System	4	MCS	MCS	4	64
6	CS-516	Data Structures and Algos	4	MCS	MCS	4	64
7	CS-616	Intro to Database Systems	3	MCS	MCS	4	64
8	MG-105	Intro to Accounting	3	BBA	BBA	8	130
9	MG-314	Money & Capital Mark	3	BIT	BIT	8	132
10	MG-505	Intro to Accounting	3	MBA	MBA	4	64
11	MT-305	Linear Algebra	3	MCS	MCS	4	64

Fig. 2: Output of inner join on tables of figure 1

As can be seen in the figure, the common attribute appears twice in the output of inner join; that is, from both the tables. Another possible approach to implement inner join can be as follows:

SELECT * FROM course, program WHERE course.prName = program.prName

The output of this statement will be exactly the same as is given in figure 2.

Outer Join

SQL supports some interesting variants of the join operation that rely on null values, called outer joins. Consider the two tables COURSE and PROGRAM given in figure 1 and their inner join given in figure 2. Tuples of COURSE that do not match some row in PROGRAM according to the inner join condition (COURSE.prName = PROGRAM.prName) do not appear in the result. In an outer join, on the other hand, COURSE rows without a matching PROGRAM row appear exactly once in the result, with the result columns inherited from PROGRAM assigned null values.

In fact, there are several variants of the outer join idea. In a right outer join, COURSE rows without a matching PROGRAM row appear in the result, but not vice versa. In a left outer join, PROGRAM rows without a matching COURSE row appear in the result, but not vice versa. In a full outer join, both COURSE and PROGRAM rows without a match appear in the result. (Of course, rows with a match always appear in the result, for all these variants, just like the usual joins or inner joins).

SQL-92 allows the desired type of join to be specified in the FROM clause. For example,

- **Select * from COURSE c RIGHT OUTER JOIN**

PROGRAM p on c.prName = p.prName

	crCode	crName	crCrdts	prName	prName	totSem	prCredits
1	CS-105	Intro to Programming	4	BCS	BCS	8	134
2	CS-216	Data Structures	4	BCS	BCS	8	134
3	CS-316	Database Systems	4	BCS	BCS	8	134
4	CS-504	Analysis of Algorithm	4	MCS	MCS	4	64
5	CS-511	Operating System	4	MCS	MCS	4	64
6	CS-516	Data Structures and Algos	4	MCS	MCS	4	64
7	CS-616	Intro to Database Systems	3	MCS	MCS	4	64
8	MG-103	Intro to Management	NULL	NULL	NULL	NULL	NULL
9	MG-105	Intro to Accounting	3	BBA	BBA	8	130
10	MG-314	Money & Capital Mark	3	BIT	BIT	8	132
11	MG-505	Intro to Accounting	3	MBA	MBA	4	64
12	MT-305	Linear Algebra	3	MCS	MCS	4	64

Fig. 3: Right outer join of the tables in figure 1

In figure 3 above, the row number 8 is the non matching row of COURSE that contains nulls in the attributes corresponding to PROGRAM table, rest of the rows are the same as in inner join of figure 2.

- **Select * from COURSE c LEFT OUTER JOIN**

PROGRAM p on c.prName = p.prName

	crCode	crName	crCrdts	prName	prName	totSem	prCredits
1	MG-105	Intro to Accounting...	3	BBA	BBA	8	130
2	CS-105	Intro to Programmin...	4	BCS	BCS	8	134
3	CS-216	Data Structures	4	BCS	BCS	8	134
4	CS-316	Database Systems	4	BCS	BCS	8	134
5	MG-314	Money & Capital Mar...	3	BIT	BIT	8	132
6	MG-505	Intro to Accounting...	3	MBA	MBA	4	64
7	CS-504	Analysis of Algorit...	4	MCS	MCS	4	64
8	CS-511	Operating System	4	MCS	MCS	4	64
9	CS-516	Data Structures and...	4	MCS	MCS	4	64
10	CS-616	Intro to Database S...	3	MCS	MCS	4	64
11	MT-305	Linear Algebra	3	MCS	MCS	4	64
12	NULL	NULL	NULL	NULL	MIT	4	62

Fig. 4: Left outer join of the tables in figure 1

In figure 4 above, the row number 12 is the non matching row of PROGRAM that contains nulls in the attributes corresponding to COURSE table, rest of the rows are the same as in inner join of figure 2.

- **Select * from COURSE c FULL OUTER JOIN**

  PROGRAM p on c.prName = p.prName

	prName	totSem	prCredits	crCode	crName	crCrdts	prName
1	NULL	NULL	NULL	MG-103	Intro to Management	NULL	NULL
2	BBA	8	130	MG-105	Intro to Accounting	3	BBA
3	BCS	8	134	CS-105	Intro to Programming	4	BCS
4	BCS	8	134	CS-216	Data Structures	4	BCS
5	BCS	8	134	CS-316	Database Systems	4	BCS
6	BIT	8	132	MG-314	Money & Capital Mark	3	BIT
7	MBA	4	64	MG-505	Intro to Accounting	3	MBA
8	MCS	4	64	MT-305	Linear Algebra	3	MCS
9	MCS	4	64	CS-504	Analysis of Algorithm	4	MCS
10	MCS	4	64	CS-616	Intro to Database Systems	3	MCS
11	MCS	4	64	CS-511	Operating System	4	MCS
12	MCS	4	64	CS-516	Data Structures and Algos	4	MCS
13	MIT	4	62	NULL	NULL	NULL	NULL

Fig. 5: Full outer join of the tables in figure 1

In figure 5 above, the row number 1 and 13 are the non matching rows from both tables, rest of the rows are the same as in inner join of figure 2.

Semi Join

Another form of join that involves two operations. First inner join is performed on the participating tables and then resulting table is projected on the attributes of one table. The advantage of this operation is that we can know the particular rows of one table that are involved in inner join. For example, through semi join of COURSE and PROGRAM tables we will get the rows of COURSE that have matching rows in PROGRAM, or in other words, the courses that are part of any program. Same can be performed other way round. SQL does not provide any operator as such, but can be implemented by select and inner join operations, for example.

- **SELECT distinct p.prName, totsem, prCredits FROM program p inner JOIN course c ON p.prName = c.prName**

	prName	totsem	prCredits
1	BBA	8	130
2	BCS	8	134
3	BIT	8	132
4	MBA	4	64
5	MCS	4	64

Fig. 6: Semi join of tables in figure 1

Self Join

In self join a table is joined with itself. This operation is used when a table contains the reference of itself through PK, that is, the PK and the FK are both contained in the same table supported by the referential integrity constraint. For example, consider STUDENT table having an attribute 'cr' storing the id of the student who is the class representative of a particular class. The example table is shown in figure 7, where a CR has been specified for the MCS class, rest of the class students contain a null in the 'cr' attribute.

	stId	stname	prName	cr
1	S0123	Amjad	MCS	NULL
2	S1012	Amjad	MCS	S0123
3	S1015	Tahira Ejaz	MCS	S0123
4	S1018	Arif Zia	BIT	NULL
5	S1020	Suhai Dar	MCS	S0123
6	S1021	M. Ali	MBA	NULL
7	S1034	Sadia Zia	BIT	NULL

Fig. 7: Example STUDENT table

Applying self join on this table:

- **SELECTa.stId, a.stName, b.stId, b.stName FROM student a, student b**
  WHERE a.cr = b.stId

Since same table is involved two times in the join, we have to use the alias. The above statement displays the names of the students and of the CR.

	stId	stName	stId	stName
1	S1012	Amjad	S0123	Amjad
2	S1015	Tahira Ejaz	S0123	Amjad
3	S1020	Suhai Dar	S0123	Amjad

Fig. 8: Self join of STUDENT table of figure 7

Subquery

Subquery is also called nested query and is one of the most powerful features of SQL. A nested query is a query that has another query embedded within it; the embedded query is called a subquery. When writing a query, we sometimes need to express a condition that refers to a table that must itself be computed. The query used to compute this subsidiary table is a subquery and appears as part of the main query. A subquery typically appears within the WHERE clause of a query. Subqueries can sometimes appear in the FROM clause or the HAVING clause. Here we have discussed only subqueries that appear in the WHERE clause. The treatment of subqueries appearing elsewhere is quite similar. Examples of subqueries that appear in the FROM clause are discussed in following section.

Lets suppose we want to get the data of the student with the maximum cgpa, we cannot get them within a same query since to get the maximum cgpa we have to apply the group function and with group function we cannot list the individual attributes. So we use nested query here, the outer query displays the attributes with the condition on cgpa whereas the subquery finds the maximum cgpa as shown below:

- SELECT * from student where cgpa >
  (select max(cgpa) from student where prName = 'BCS‘)

	stId	stName	stFName	stAdres	stPhone	prName	curSem	cgpa
1	S0123	Amjad	Hussain	8-SD Lahore	234322	MCS	1	NULL
2	S1012	Amjad	Rehan	I8 Ibd	5456754	MCS	2	2.3
3	S1018	Arif Zia	Zia Khan	GM Rawalpindi	4356488	BIT	2	2.8
4	S1021	M. Ali	M. Saad	JT Lahore	544325	MBA	2	2.8
5	S1034	Sadia Zia	NULL	NULL	NULL	BIT	1	NULL
6	S1038	Shoaib Ali	Rahmat Ali	G-6 Islamabad	5343240	BCS	2	3.2
7	S1020	Suhai Dar	Nek Muhammad	I-8 Islamabad	5523240	MCS	2	3.2
8	S1015	Tahira Ejaz	Nek Muhammad	AJ Road	4323456	MCS	3	3.2

	stId	stName	stFName	stAdres	stPhone	prName	curSem	cgpa	cr
1	S1015	Tahira Ejaz	Nek Muhammad	AJ Road	4323456	MCS	3	3.25	S012

Fig. 9: STUDENT table and nested query applied on it

We have to take care of the operator being applied in case of subquery in the where clause. The type of operator depends on the result set being returned by the subquery. If the output expected from the subquery is a single value, as is the case in the above example, then we can use operators like =, <, >, etc. However, if the subquery returns multiple values then we can use operators like IN, LIKE etc. The IN operator allows us to test whether a value is in a given set of elements; an SQL query is used to generate the set to be tested. We can also use the NOT IN operator where required.

The subquery can be nested to any level, the queries are evaluated in the reverse order, and that is, the inner most is evaluated first, then the outer one and finally the outer most.

ACCESS CONTROL

SQL-92 supports access control through the GRANT and REVOKE commands. The GRANT command gives users privileges to base tables and views. The syntax of this command is as follows:

GRANT privileges ON object TO users [WITH GRANT OPTION]

For our purposes object is either a base table or a view. Several privileges can be specified, including these:

SELECT: The right to access (read) all columns of the table specified as the object, including columns added later through ALTER TABLE commands.

INSERT(*column-name*): The right to insert rows with (non-*null* or nondefault) values in the named column of the table named as object. If this right is to be granted with

respect to all columns, including columns that might be added later, we can simply use INSERT. The privileges UPDATE(*column-name*) and UPDATE are similar.

DELETE: The right to delete rows from the table named as object.

REFERENCES(*column-name*): The right to define foreign keys (in other tables) that refer to the speci_ed column of the table object. REFERENCES without a column name speci_ed denotes this right with respect to all columns, including any that are added later.

If a user has a privilege with the grant option, he or she can pass it to another user (with or without the grant option) by using the GRANT command. A user who creates a base table automatically has all applicable privileges on it, along with the right to grant these privileges to other users. A user who creates a view has precisely those privileges on the view that he or she has on *every* one of the view or base tables used to define the view. The user creating the view must have the SELECT privilege on each underlying table, of course, and so is always granted the SELECT privilege on the view. The creator of the view has the SELECT privilege with the grant option only if he or she has the SELECT privilege with the grant option on every underlying table.

In addition, if the view is updatable and the user holds INSERT, DELETE, or UPDATE privileges (with or without the grant option) on the (single) underlying table, the user automatically gets the same privileges on the view.

Only the owner of a schema can execute the data definition statements CREATE, ALTER, and DROP on that schema. The right to execute these statements cannot be granted or revoked.

In conjunction with the GRANT and REVOKE commands, views are an important component of the security mechanisms provided by a relational DBMS. We will discuss the views later in detail. Suppose that user Javed has created the tables COURSE, PROGRAM and STUDENT. Some examples of the GRANT command that Javed can now execute are listed below:

- GRANT INSERT, DELETE ON COURSE TO Puppoo WITH GRANT OPTION
- GRANT SELECT ON COURSE TO Mina
- GRANT SELECT ON PROGRAM TO Mina WITH GRANT OPTION

There is a complementary command to GRANT that allows the withdrawal of privileges. The syntax of the REVOKE command is as follows:

```
REVOKE [GRANT OPTION FOR] privileges ON object FROM users
{RESTRICT | CASCADE}
```

The command can be used to revoke either a privilege or just the grant option on a privilege (by using the optional GRANT OPTION FOR clause). One of the two alternatives, RESTRICT or CASCADE, must be specified; we will see what this choice means shortly. The intuition behind the GRANT command is clear: The creator of a base table or a view is given all the appropriate privileges with respect to it and is allowed to pass these privileges including the right to pass along a privilege to other users. The REVOKE command is, as expected, intended to achieve the reverse: A user who has granted a privilege to another user may change his mind and want to withdraw the granted privilege. The intuition behind exactly what effect a REVOKE command has is complicated by the fact that a user may be granted the same privilege multiple times, possibly by different users.

When a user executes a REVOKE command with the CASCADE keyword, the effect is to withdraw the named privileges or grant option from all users who currently hold these privileges solely through a GRANT command that was previously executed by the same user who is now executing the REVOKE command. If these users received the privileges with the grant option and passed it along, those recipients will also lose their privileges as a consequence of the REVOKE command unless they also received these privileges independently. Consider what happens after the following sequence of commands, where Javed is the creator of COURSE.

GRANT SELECT ON COURSE TO Alia WITH GRANT OPTION (executed by Javed)

GRANT SELECT ON COURSE TO Bobby WITH GRANT OPTION (executed by Alia)

REVOKE SELECT ON COURSSE FROM Alia CASCADE (executed by Javed)

Alia loses the SELECT privilege on COURSE, of course. Then Bobby, who received this privilege from Alia, and only Alia, also loses this privilege. Bobby's privilege is said to be abandoned when the privilege that it was derived from (Alia's SELECT privilege with grant option, in this example) is revoked. When the CASCADE

keyword is specified, all abandoned privileges are also revoked (possibly causing privileges held by other users to become abandoned and thereby revoked recursively). If the RESTRICT keyword is specified in the REVOKE command, the command is rejected if revoking the privileges *just* from the users specified in the command would result in other privileges becoming abandoned.

Consider the following sequence, as another example:

GRANT SELECT ON COURSE TO Alia WITH GRANT OPTION *(executed by Javed)*
GRANT SELECT ON COURSE TO Bobby WITH GRANT OPTION *(executed by Javed)*
GRANT SELECT ON COURSE TO Bobby WITH GRANT OPTION *(executed by Alia)*
REVOKE SELECT ON COURSE FROM Alia CASCADE *(executed by Javed)*

As before, Alia loses the SELECT privilege on COURSE. But what about Bobby? Bobby received this privilege from Alia, but he also received it independently (coincidentally, directly from Javed). Thus Bobby retains this privilege. Consider a third example:

GRANT SELECT ON COURSE TO Alia WITH GRANT OPTION *(executed by Javed)*
GRANT SELECT ON COURSE TO Alia WITH GRANT OPTION *(executed by Javed)*
REVOKE SELECT ON COURSE FROM Alia CASCADE *(executed by Javed)*

Since Javed granted the privilege to Alia twice and only revoked it once, does Alia get to keep the privilege? As per the SQL-92 standard, no. Even if Javed absentmindedly granted the same privilege to Alia several times, he can revoke it with a single REVOKE command. It is possible to revoke just the grant option on a privilege:

GRANT SELECT ON COURSE TO Alia WITH GRANT OPTION *(executed by Javed)*
REVOKE GRANT OPTION FOR SELECT ON COURSE FROM Alia CASCADE *(executed by Javed)*

This command would leave Alia with the SELECT privilege on COURSE, but Alia no longer has the grant option on this privilege and therefore cannot pass it on to other users.

Summary

In this lecture we have studied the different types of joins, with the help of which we can join different tables. We also discussed two major commands of access control that are also considered the part of Data Control Language component of SQL. The last part of this lecture handout is taken from chapter 17 of the reference given. The

interested users are recommended to see the book for detailed discussion on the topic. There is a lot left on SQL, for our course purposes however we have studied enough. Through extensive practice you will have clear understanding that will help you in further learning.

Exercise:

Practice for all various types of Joins and Grant and Revoke commands.

Lecture No. 32

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill
Database Management, Jeffery A Hoffer

Overview of Lecture

- Application Programs
- User Interface
- Designing Forms

Until now we have studied conceptual database design, whose goal is to analyze the application and do a conceptual design on the application. The goal is to provide a conceptual design and description of reality. It is independent of data model. Then we discussed the relational database design. A relational database stores all its data inside tables, and nothing more. All operations on data are done on the tables themselves or produce another tables as the result. Here we had discussed the selection of data model. Thereafter we had discussed data manipulation language through SQL. We are using SQL Server as a tool. In this lecture we will discuss application program.

Application Programs

Programs written to perform different requirement posed by the users/organization are the application programs. Application programs can be developed in parallel or after the construction of database design. Tool selection is also critical, but it depends upon developer in which he feels comfortable.

General Activities in Application Programs
Following are the general activities, which are performed during the development of application programs:

- Data input programmes
- Editing
- Display

- Processing related to activities
- Reports

User Interface

In the minds of users, a system's user interface is the system; everything else is just stuff they're happy to ignore. The design of the user interface is therefore critical to the success or failure of a project. Get it right, and your users will forgive the occasional infelicity in implementation. Get it wrong, and it won't really matter how efficient your code is.

The irony here is that if you do get it right, hardly anyone will notice. Really elegant interfaces are invisible. Even if you get it wrong, no one might notice. The interfaces of many computer systems, particularly database systems, are so awkward that your system will be just one more of the mediocre, mildly abusive computer systems people have come to expect.

Effective interfaces can also require more work to implement, although this isn't necessarily the case. In addition, the payoffs can be huge, and they're not all of the "virtue is its own reward" variety. An effective user interface minimizes the time users require to learn and implement the system. Once the system is implemented, productivity gains are higher if users don't have to struggle to bend it to their will. Chances are good that both of these issues were addressed in the project goals. They certainly impact the infamous bottom line.

Effective interfaces that closely match the users' expectations and work processes also minimize the need for external documentation, which is always expensive. And while users might not consciously notice how wonderful your user interface is, they'll certainly notice that your system seems to just work better.

So, what constitutes an effective interface? To my mind, it's one that helps users accomplish their tasks and otherwise gets out of the way. An effective interface doesn't impose its requirements on users. It never forces users to play by its rules; it plays by the users' rules. An effective interface doesn't force users to learn a bunch of

uninteresting stuff just to use it. And finally, it doesn't behave in unexpected ways. Following are the two types of user interfaces:

- Text based
- Graphical User Interface (GUI) most commonly called as Forms

Text Based User Interface

In text based user interface certain keyboard numbers are designated for any action but it is used very rarely nowadays. For example

Adding a Record	1
Deleting a Record	2
Enrollment	3
Result Calculation	4
Exit	5

Forms

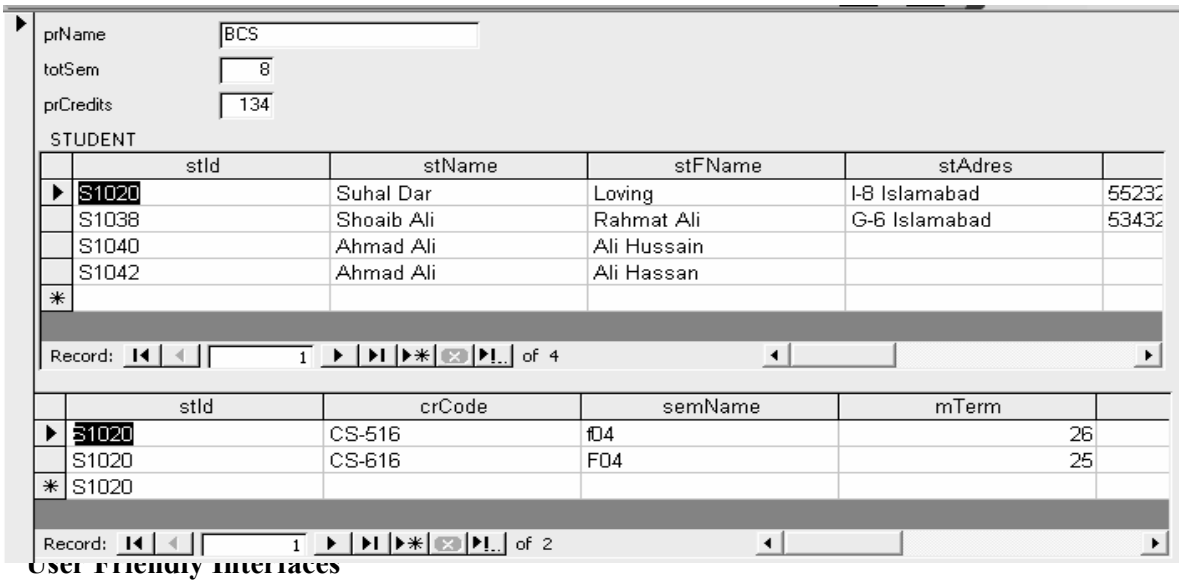
Forms are now days used extensively in the application programs. Following are the different types of forms

Browser Based

These are web-based forms. They are developed in HTML, scripting language or Front Page.

Non-Browser/Simple

Graphical User Interface



User Friendly Interfaces

Two definitions of user-friendly that are often mooted are "easy to learn" and "easy to use." If we put aside for the moment the question of what, exactly, "easy" means, we still have to ask ourselves, "Easy for whom?" The system that's easy for a beginner to learn is not necessarily easy for an expert to use. Your best approach is to consider the needs of each level of user and accommodate each with different facets of the interface. Following are the different kinds of users:

Beginners

Everyone is a beginner at some point. Very few people remain that way—they will either pass through the stage to intermediacy, or they'll discard your system entirely in favor of someone else's. For this reason, you must be careful not to build in support for beginners that will get in the way of more advanced users. Beginners need to know what your system does before they start to learn how to use it. The best way to present this information is outside the main system itself. For simple systems, an introductory dialog box that describes the system can be sufficient. (Just be sure you always include a means of dismissing the dialog box permanently.) For more complex

systems, a guided tour might be more appropriate. Online help isn't a good option for beginners. They might not know it exists or, if they do, how to use it. I have had some success using an online user's guide, however, by including a link to it from the introductory dialog box and from the Help menu. To be successful with beginners, these guides must be task-oriented. Beginners don't want to know what "menu item" means; they want to know how to create an invoice.

Intermediate

For most systems, the majority of users fall into the intermediate category. Intermediate users know what the system does, but they often forget the details of how. This is the group you must support directly in the user interface. Fortunately, the Microsoft Windows interface provides a lot of tools for helping these users.A well-designed menu system is one of the best tools for reminding intermediate users of the system capabilities. A quick scan of the available menu items will immediately remind them of the functions available and at the same time allow them to initiate the appropriate task.

An excellent second level of support for intermediate users is online help. Writing online help is outside the scope of this book. In this context, however, I will mention that most intermediate users will use the index as their primary access mechanism. The index should therefore be as complete as you can possibly make it.

Experts

Expert users know what to do and how to do it. They're primarily interested in doing things quickly. The more shortcuts you can build into your system, the happier you will make this group of users. In my experience, expert users tend to be keyboard-oriented, so make sure that you provide a way to move around the system using the keyboard if you're catering to this group.Expert users also appreciate the ability to customize their working environment. Providing this functionality can be an expensive exercise, however, so you will want to carefully evaluate the benefit before including it. If you do decide to include some level of interface customization, even if it's only a matter of arranging windows on the screen, be certain to maintain the changes between sessions. Nothing is as irritating as having to rearrange everything every time you load a program

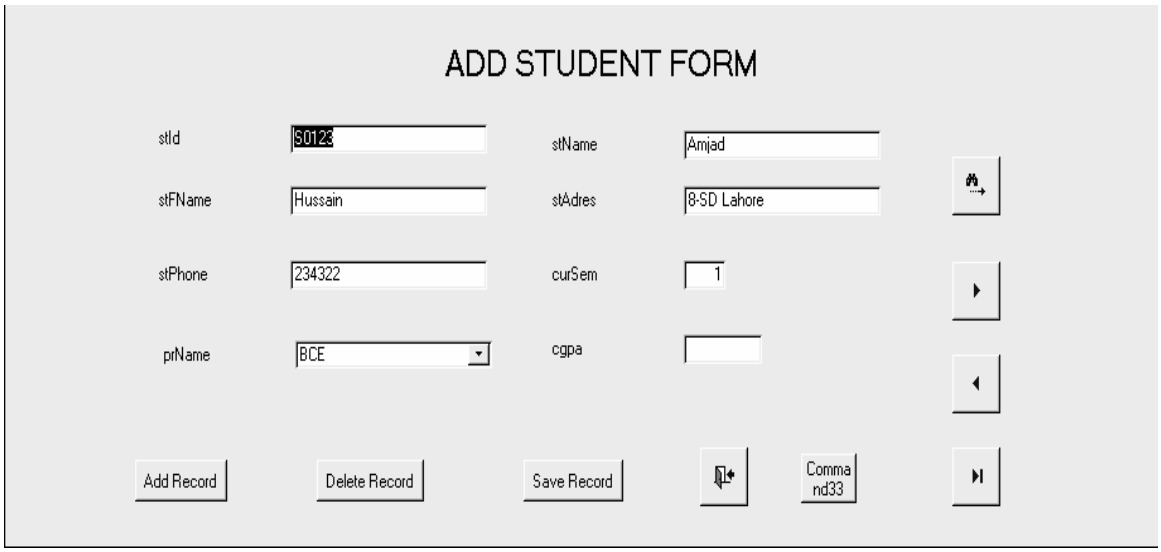
Tips for User Friendly Interface

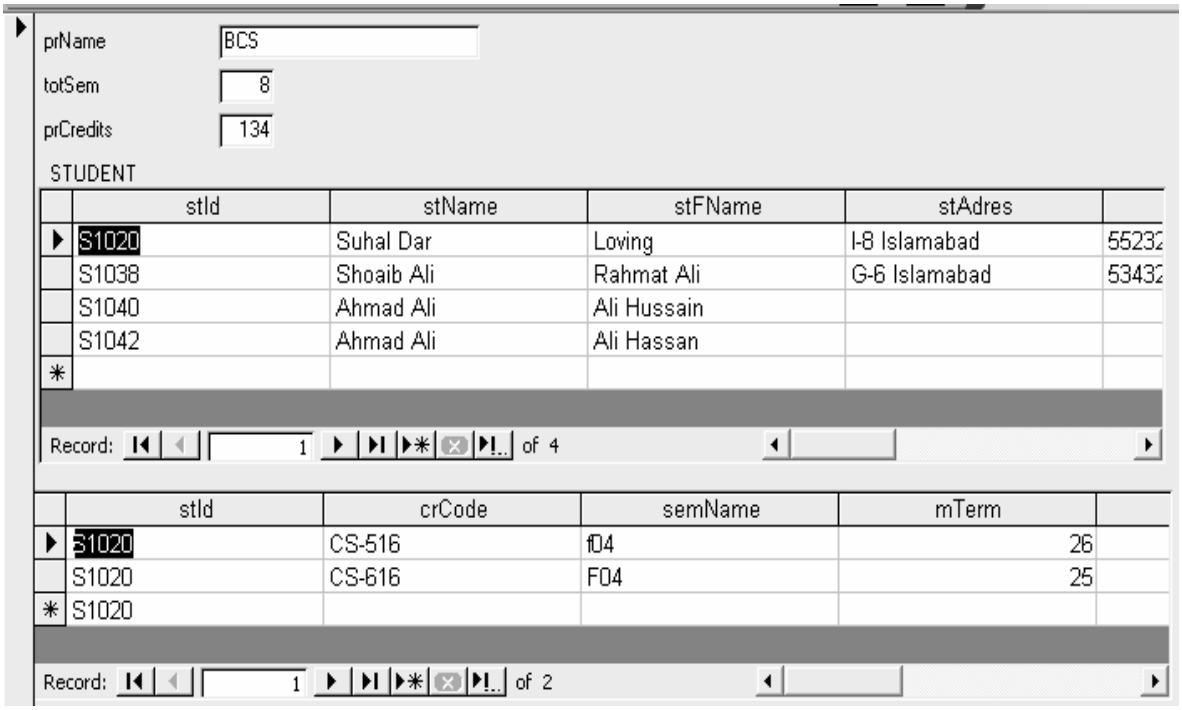
Following are some of the important tips, which should be adhered for interfaces:

- It should be user friendly and user must not search for required buttons or text boxes.
- Interface should be designed in such a manner that user is in charge of the form.
- It should be designed in manner that user memory is always tested
- You should be consistent in your approach while designing interface.
- It should be processes based rather than the data structure based.

Entities and Relationships

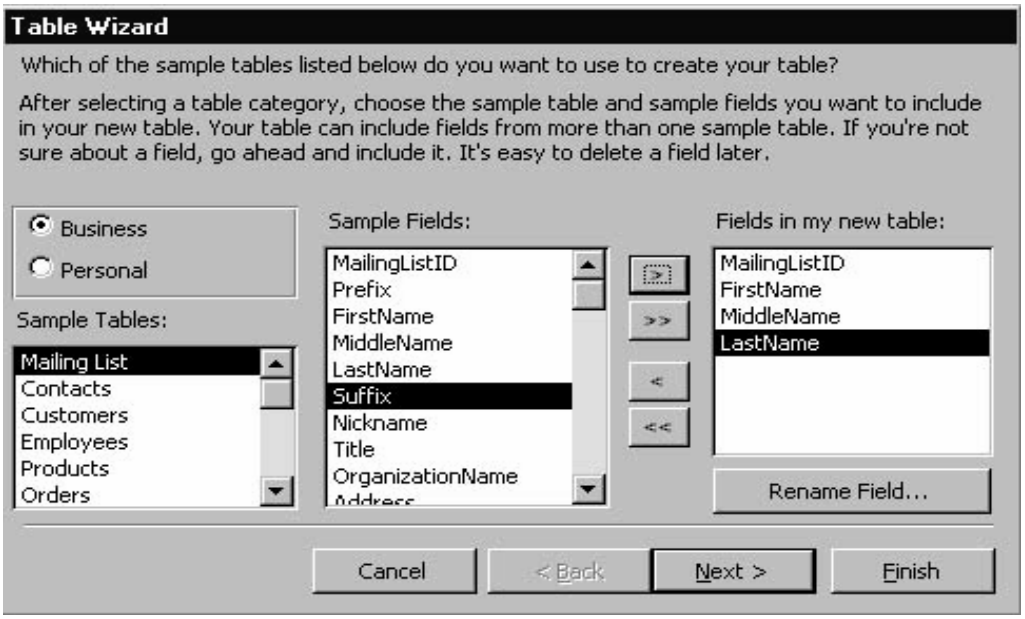
First type is the simple entity on a page.





Windows Controls

There are number of controls which are used to take input and display output like buttons, checkboxes etc. Following are the examples



Numbers, Dates and Text
Normally text boxes are used for the display of dates

Summary

In today's lecture we have studied the application programs and designing user interface and forms. We have studied different techniques and practices for the user interface and form designing. We will discuss an example of form designing in our next lecture.

Lecture No. 33

Reading Material

Programming Microsoft Access
Mastering MS Access

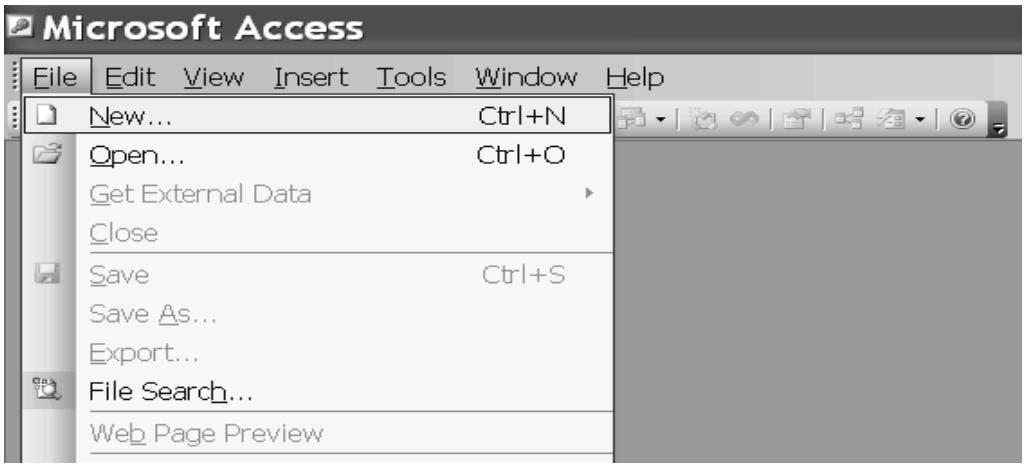
Overview of Lecture

- Designing Input Form
- Arranging Form
- Adding Command Buttons

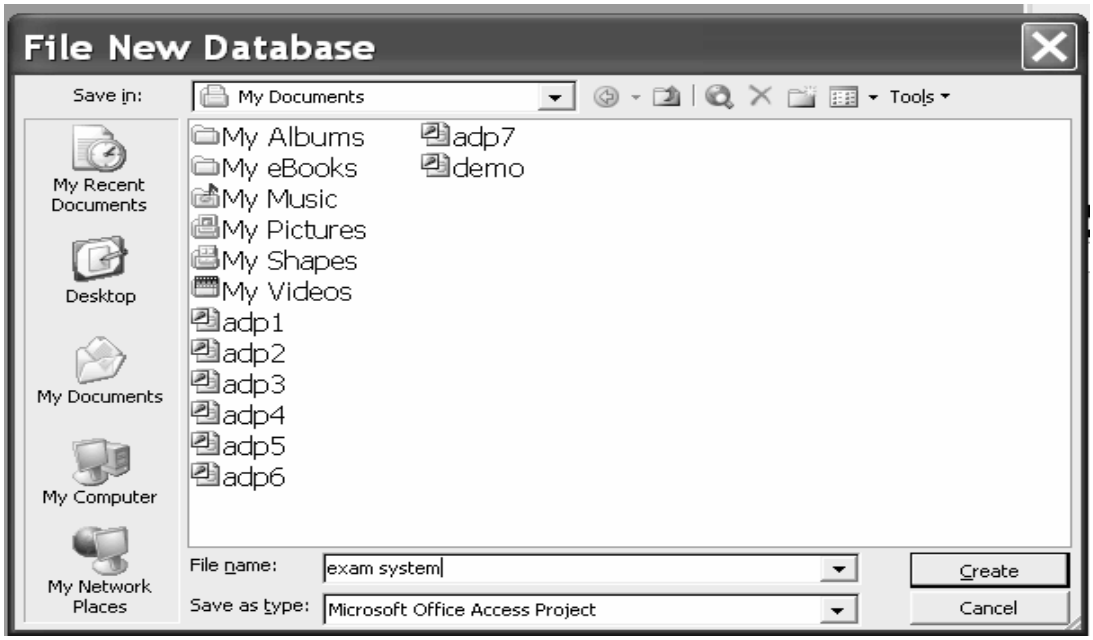
In the previous lecture we have discussed the importance of user interface. It plays an important role in the development of any application. User will take interest in the application if user interface is friendly. We then discussed different tools, which are used in the development of any application. In this lecture we will see the designing of input forms.

An input form is an easy, effective, efficient way to enter data into a table. Input forms are especially useful when the person entering the data is not familiar with the inner workings of Microsoft Access and needs to have a guide in order to input data accurately into the appropriate fields. Microsoft Access provides several predefined forms and provides a forms wizard that walks you through the process of creating a form. One of these predefined forms will be used in the example below. You can also create your own customized forms by using Microsoft Access form design tools. Following things must be ensured for input forms:

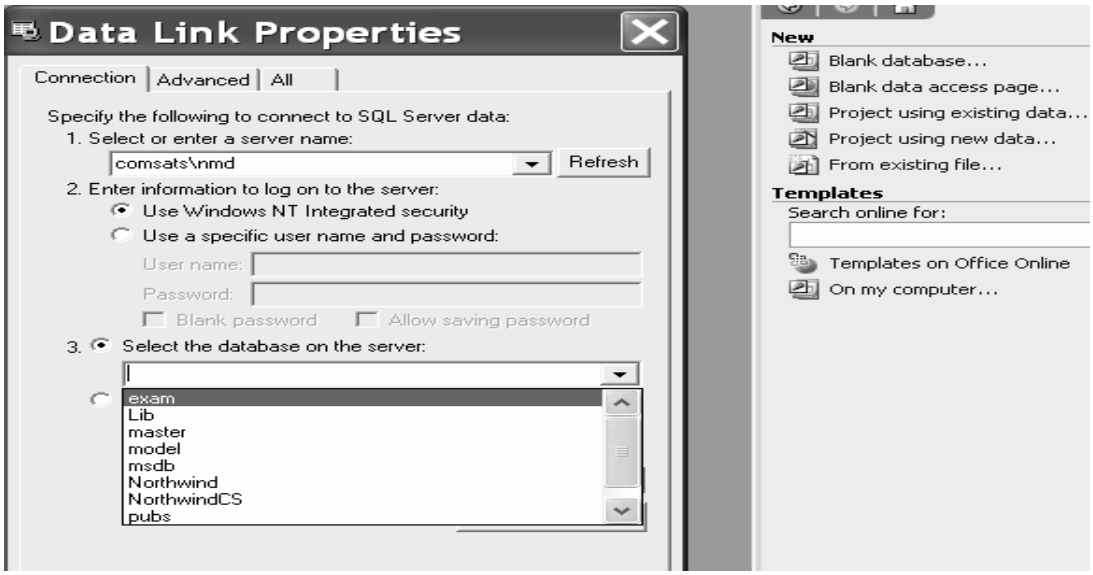
- Forms should be user friendly
- Data integrity must be ensured, which means that database must represent the true picture of real system.
- Checks can be applied within the tables definition or through input forms



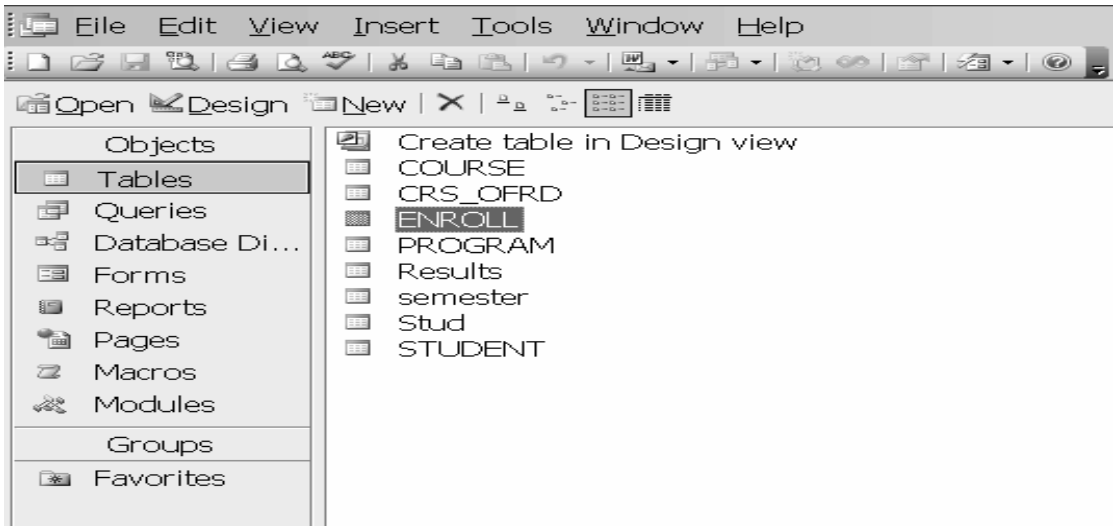
So first we will run MS Access and select New option from the file. Next it will ask the name of database as follows.



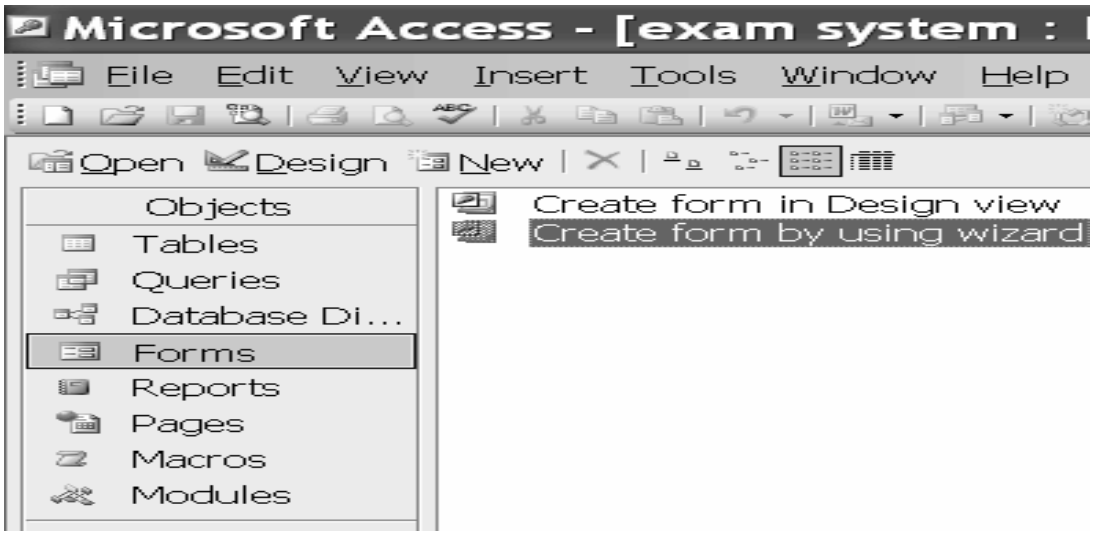
We have given the name as Exam System, we will then press the create button and following screen would be available.



In this screen we will first select project using existing data from New templates, as we are using data of SQL Server that is why we are using this option. Next move on to Data Link Properties dialog, which is adjacent one. So first select the connection Tab in which first select the server name then is the security setting after selecting that option then is the selection of database on which forms are to be constructed.



Now these are the tables of Exam Data base, with which connectivity has been established. Now we will select the Forms option as under:



So here are two options in design and wizard both we are selecting wizard view after selecting this next screen would be as under:

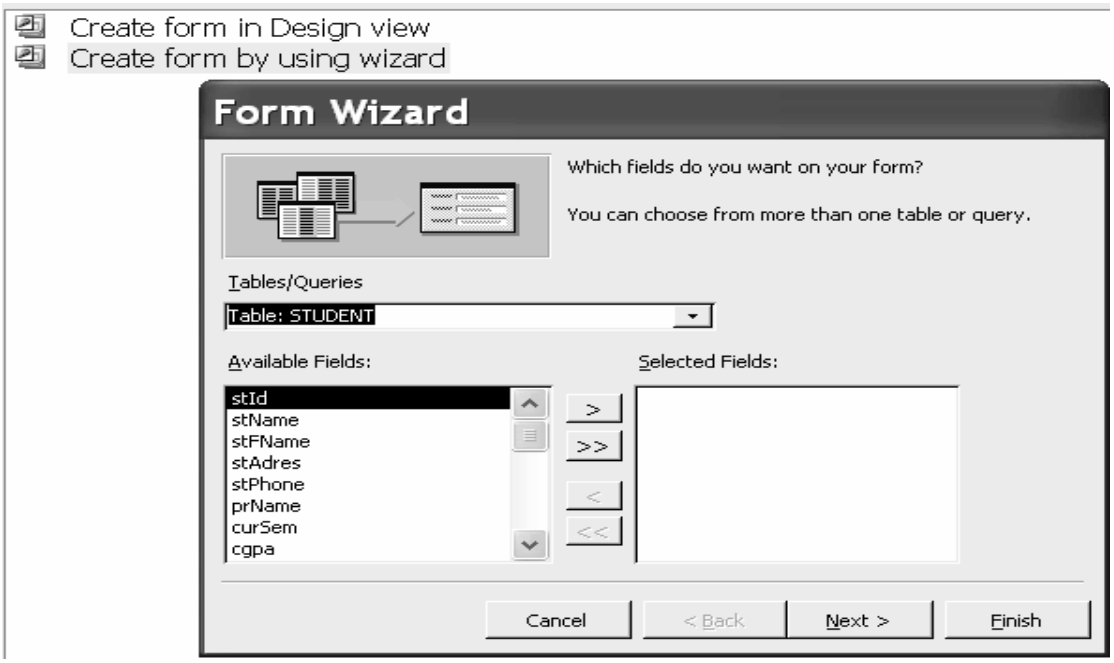
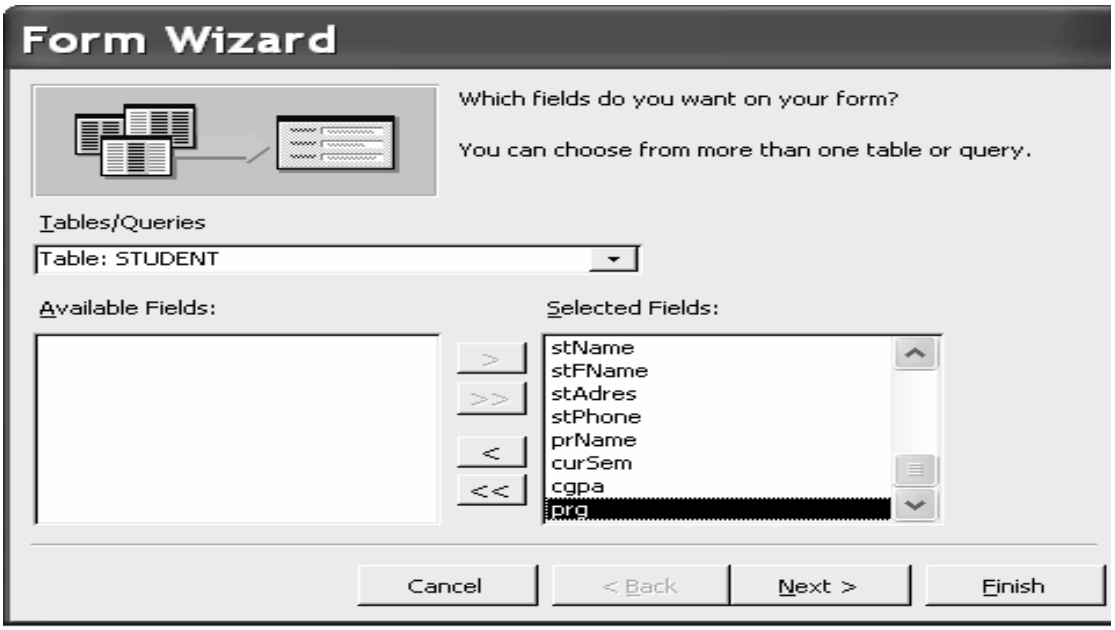
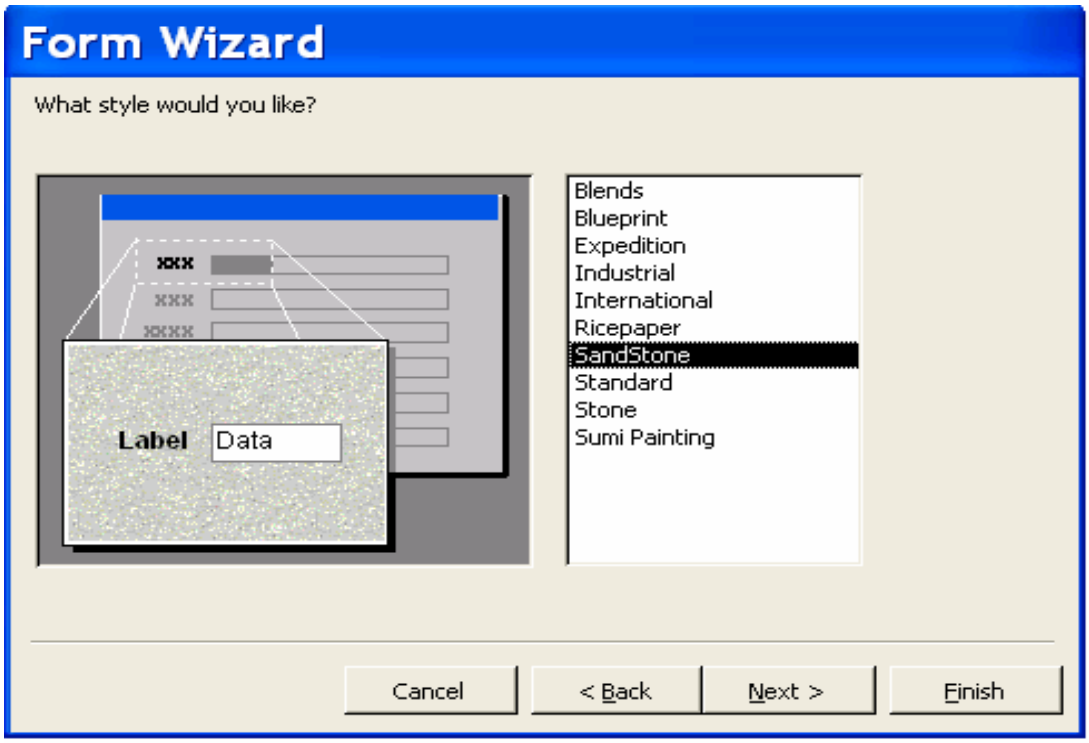


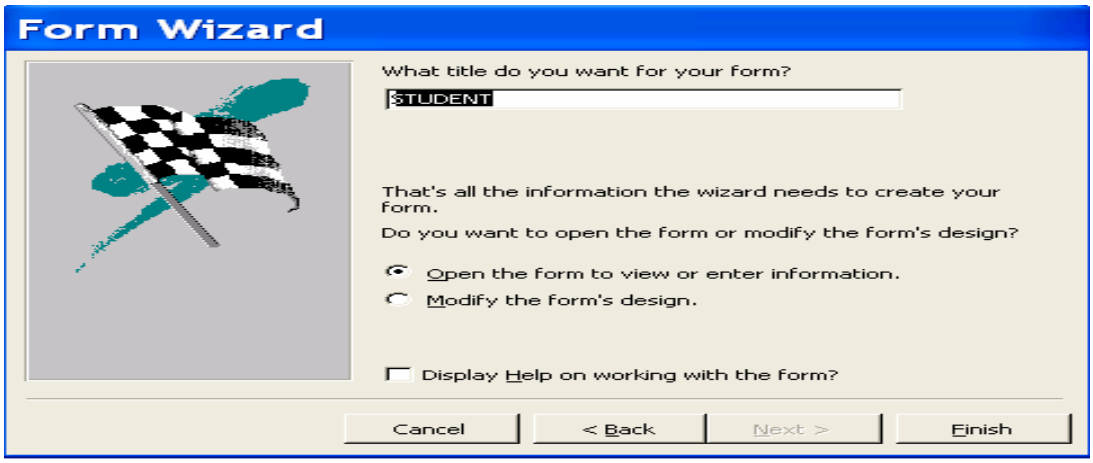
table as under.



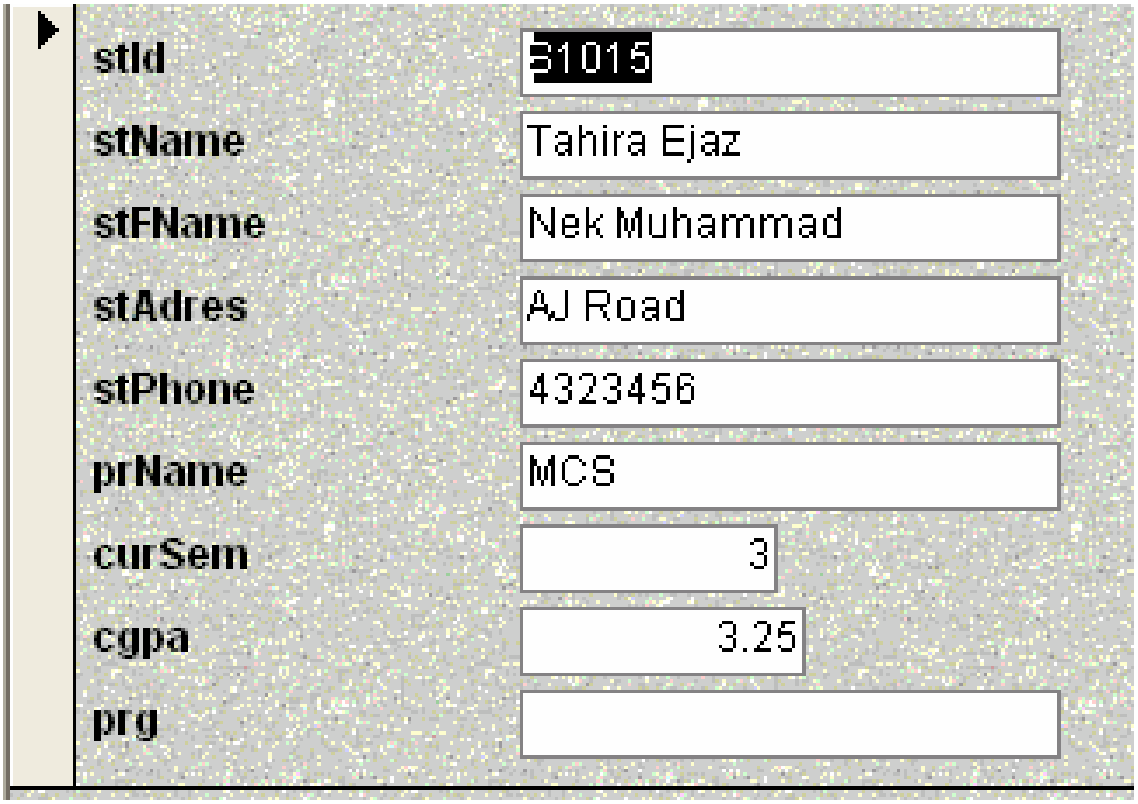
Then is the selection of required layout for the form. There are different options available in this option. We can select the required option. We have selected Column option for the layout of forms. Then is the selection of background or style for our forms as under. We have selected the SandStone option.



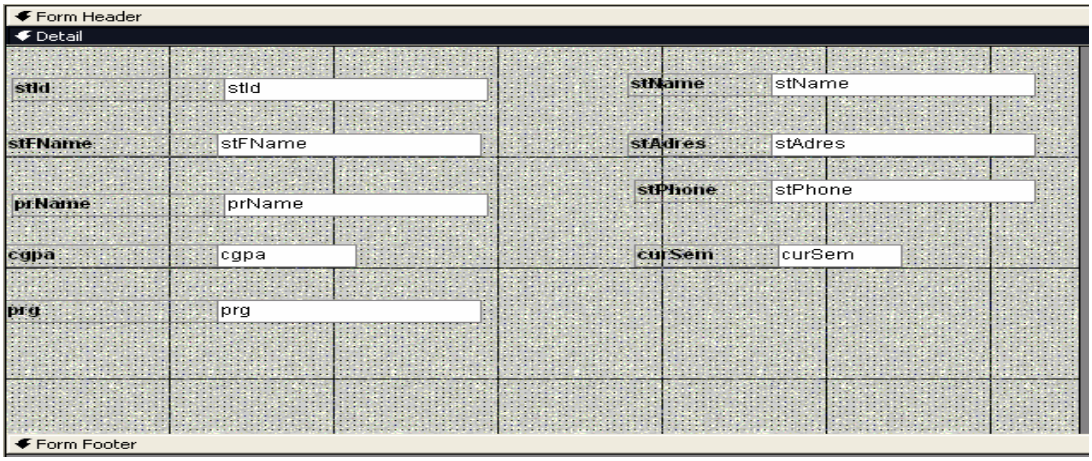
Next is the selection of title for the form as under.



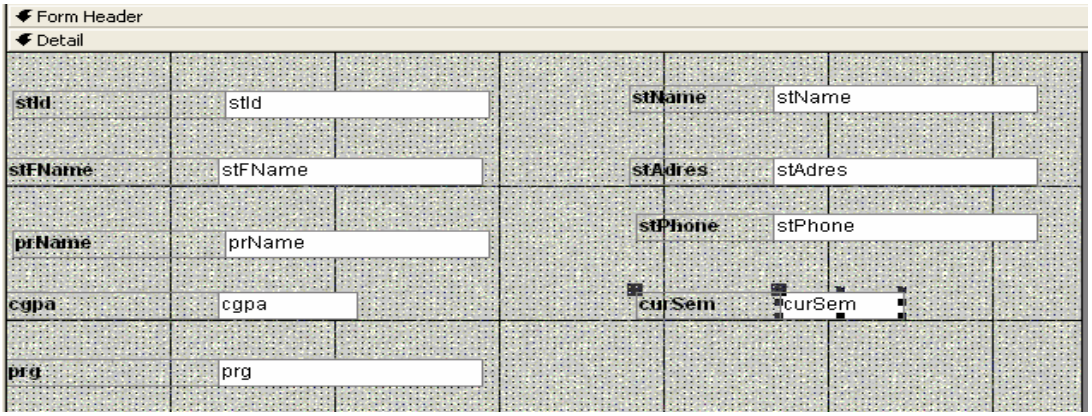
Next is the form view as under in which we can view our data.



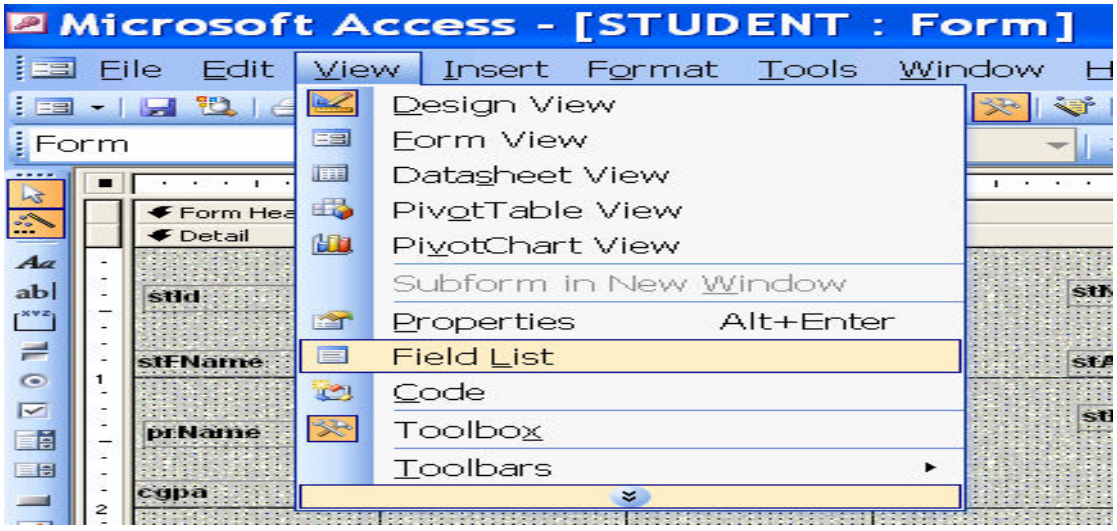
Forms must be designed and arranged in a systematic manner now



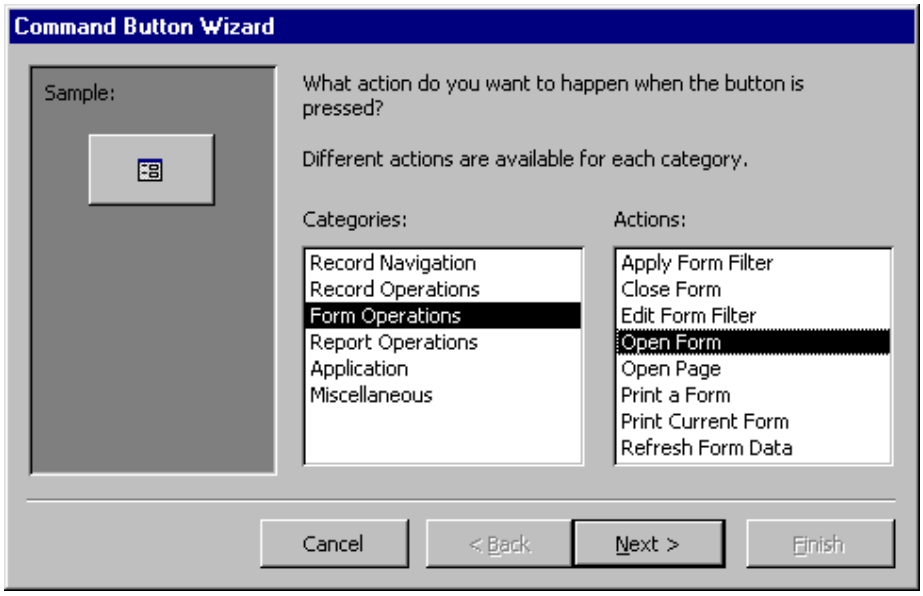
Next is deleting a field from the form.



If we want to add an attribute in the list.



- Open the form in Design View and ensure that the Control Wizard button on the toolbox is pressed in.
- Click the command button icon on the toolbox and draw the button on the form. The Command Button Wizard will then appear.
- On the first dialog window, action categories are displayed in the left list while the right list displays the actions in each category. Select an action for the command button and click Next .



- The next few pages of options will vary based on the action you selected. Continue selecting options for the command button.
- Choose the appearance of the button by entering caption text or selecting a picture. Check the Show All Pictures box to view the full list of available

images. Click **Next >**.



- Enter a name for the command button and click **Finish** to create the button.

In today's lecture we have studied the designing of forms. Forms are used as an alternative way to enter data into a database table. It can be made more perfect with lots of practice and designing number of forms. So it needs practice. We have designed a simple form through wizard. A more complex form can also be designed with some practice.

Lecture No. 34

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill
"Modern Database Management", Fred McFadden, Jeffrey Hoffer, Benjamin/Cummings

Overview of Lecture

- Data Storage Concepts
- Physical Storage Media
- Memory Hierarchy

In the previous lecture we have discussed the forms and their designing. From this lecture we will discuss the storage media.

Classification of Physical Storage MediaStorage media are classified according to following characteristics:

Speed of access

Cost per unit of data

Reliability

We can also differentiate storage as either

Volatile storage

Non-volatile storage

Computer storage that is lost when the power is turned off is called as volatile storage. Computer storage that is not lost when the power is turned off is called as non – volatile storage, pronounced cashe is a special high-speed storage mechanism. It can be either a reserved section of main memory or an independent high-speed storage device. Two types of caching are commonly used in personal computers: memory caching and disk caching.

A memory cache, sometimes called a cache store or RAM. Cache is a portion of memory made of high-speed static RAM (SRAM) instead of the slower and cheaper DRAM used for main memory. Memory caching is effective because most programs access the same data or instructions over and over. By keeping as much of this information as possible in SRAM, the computer avoids accessing the slower DRAM.

Some memory caches are built into the architecture of microprocessors.. The Intel 80486 microprocessor, for example, contains an 8K memory cache, and the Pentium has a 16K cache. Such internal caches are often called Level 1 (L1) caches. Most modern PCs also come with external cache memory, called Level 2 (L2) caches. These caches sit between the CPU and the DRAM. Like L1 caches, L2 caches are composed of SRAM but they are much larger. Disk caching works under the same principle as memory caching, but instead of using high-speed SRAM, a disk cache uses conventional main memory. The most recently accessed data from the disk (as

well as adjacent sectors) is stored in a memory buffer. When a program needs to access data from the disk, it first checks the disk cache to see if the data is there. Disk caching can dramatically improve the performance of applications, because accessing a byte of data in RAM can be thousands of times faster than accessing a byte on a hard disk.

When data is found in the cache, it is called a *cache hit,* and the effectiveness of a cache is judged by its hit rate. Many cache systems use a technique known as smart caching, in which the system can recognize certain types of frequently used data. The strategies for determining which information should be kept in the cache constitute some of the more interesting problems in computer science.

The main memory of the computer is also known as **RAM**, standing for Random Access Memory. It is constructed from integrated circuits and needs to have electrical power in order to maintain its information. When power is lost, the information is lost too! The CPU can directly access it. The access time to read or write any particular byte are independent of whereabouts in the memory that byte is, and currently is approximately 50 **nanoseconds** (a thousand millionth of a second). This is broadly comparable with the speed at which the CPU will need to access data. Main memory is expensive compared to external memory so it has limited capacity. The capacity available for a given price is increasing all the time. For example many home Personal Computers now have a capacity of 16 **megabytes** (million bytes), while 64 megabytes is commonplace on commercial workstations. The CPU will normally transfer data to and from the main memory in groups of two, four or eight bytes, even if the operation it is undertaking only requires a single byte.

Flash memory is a form of EEPROM that allows multiple memory locations to be erased or written in one programming operation. Normal EEPROM only allows one location at a time to be erased or written, meaning that flash can operate at higher effective speeds when the system uses it to read and write to different locations at the same time. All types of flash memory and EEPROM wear out after a certain number of erase operations, due to wear on the insulating oxide layer around the charge storage mechanism used to store data.

Flash memory is non-volatile, which means that it stores information on a silicon chip in a way that does not need power to maintain the information in the chip. This means that if you turn off the power to the chip, the information is retained without consuming any power. In addition, flash offers fast read access times and solid-state shock resistance. These characteristics are why flash is popular for applications such as storage on battery-powered devices like cellular phones and PDAs.Flash memory is based on the Floating-Gate Avalanche-Injection Metal Oxide Semiconductor (FAMOS transistor) which is essentially an NMOS transistor with an additional conductor suspended between the gate and source/drain terminals.

Magnetic disk is round plate on which data can be encoded. There are two basic types of disks: magnetic disks and optical disks on magnetic disks, data is encoded as microscopic magnetized *needles* on the disk's surface. You can record and erase data on a magnetic disk any number of times, just as you can with a cassette tape. Magnetic disks come in a number of different forms:

Floppy Disk: A typical 5¼-inch floppy disk can hold 360K or 1.2MB (megabytes). 3½-inch floppies normally store 720K, 1.2MB or 1.44MB of data.

Hard Disk: Hard disks can store anywhere from 20MB to more than 10GB. Hard disks are also from 10 to 100 times faster than floppy disks.

Optical disks record data by burning microscopic holes in the surface of the disk with a laser. To read the disk, another laser beam shines on the disk and detects the holes by changes in the reflection pattern.
Optical disks come in three basic forms:
CD-ROM: Most optical disks are read-only. When you purchase them, they are already filled with data. You can read the data from a CD-ROM, but you cannot modify, delete, or write new data.
WORM: Stands for **w**rite-**o**nce, **r**ead-**m**any. WORM disks can be written on once and then read any number of times; however, you need a special WORM disk drive to write data onto a WORM disk.
Erasable optical (EO**):** EO disks can be read to, written to, and erased just like magnetic disks.
The machine that spins a disk is called a disk drive. Within each disk drive is one or more *heads* (often called read/write heads) that actually read and write data. Accessing data from a disk is not as fast as accessing data from main memory, but disks are much cheaper. And unlike RAM, disks hold on to data even when the computer is turned off. Consequently, disks are the storage medium of choice for most types of data. Another storage medium is magnetic tape. But tapes are used only for backup and archiving because they are sequential-access devices (to access data in the middle of a tape, the tape drive must pass through all the preceding data).
Short for Redundant Array of Independent (or **I**nexpensive) **D**isks, a category of disk drives that employ two or more drives in combination for fault tolerance and performance. RAID disk drives are used frequently on servers but aren't generally necessary for personal computers.
Fundamental to RAID is "striping", a method of concatenating multiple drives into one logical storage unit. Striping involves partitioning each drive's storage space into stripes which may be as small as one sector (512 bytes) or as large as several megabytes. These stripes are then interleaved round-robin, so that the combined space is composed alternately of stripes from each drive. In effect, the storage space of the drives is shuffled like a deck of cards. The type of application environment, I/O or data intensive, determines whether large or small stripes should be used.
Most multi-user operating systems today, like NT, UNIX and Netware, support overlapped disk I/O operations across multiple drives. However, in order to maximize throughput for the disk subsystem, the I/O load must be balanced across all the drives so that each drive can be kept busy as much as possible. In a multiple drive system without striping, the disk I/O load is never perfectly balanced. Some drives will contain data files which are frequently accessed and some drives will only rarely be accessed. In I/O intensive environments, performance is optimized by striping the drives in the array with stripes large enough so that each record potentially falls entirely within one stripe. This ensures that the data and I/O will be evenly distributed across the array, allowing each drive to work on a different I/O operation, and thus maximize the number of simultaneous I/O operations, which can be performed by the array.
In data intensive environments and single-user systems which access large records, small stripes (typically one 512-byte sector in length) can be used so that each record will span across all the drives in the array, each drive storing part of the data from the record. This causes long record accesses to be performed faster, since the data transfer occurs in parallel on multiple drives. Unfortunately, small stripes rule out multiple overlapped I/O operations, since each I/O will typically involve all drives. However, operating systems like DOS which do not allow overlapped disk I/O, will not be

negatively impacted. Applications such as on-demand video/audio, medical imaging and data acquisition, which utilize long record accesses, will achieve optimum performance with small stripe arrays.

RAID-0
RAID Level 0 is not redundant, hence does not truly fit the "RAID" acronym. In level 0, data is split across drives, resulting in higher data throughput. Since no redundant information is stored, performance is very good, but the failure of any disk in the array results in data loss. This level is commonly referred to as striping.
RAID-1
RAID Level 1 provides redundancy by writing all data to two or more drives. The performance of a level 1 array tends to be faster on reads and slower on writes compared to a single drive, but if either drive fails, no data is lost. This is a good entry-level redundant system, since only two drives are required; however, since one drive is used to store a duplicate of the data, the cost per megabyte is high. This level is commonly referred to as mirroring.
RAID-2
RAID Level 2, which uses Hamming error correction codes, is intended for use with drives which do not have built-in error detection. All SCSI drives support built-in error detection, so this level is of little use when using SCSI drives.

RAID-3
RAID Level 3 stripes data at a byte level across several drives, with parity stored on one drive. It is otherwise similar to level 4. Byte-level striping requires hardware support for efficient use.

RAID-4
RAID Level 4 stripes data at a block level across several drives, with parity stored on one drive. The parity information allows recovery from the failure of any single drive. The performance of a level 4 array is very good for reads (the same as level 0). Writes, however, require that parity data be updated each time. This slows small random writes, in particular, though large writes or sequential writes are fairly fast. Because only one drive in the array stores redundant data, the cost per megabyte of a level 4 array can be fairly low.

RAID-5
RAID Level 5 is similar to level 4, but distributes parity among the drives. This can speed small writes in multiprocessing systems, since the parity disk does not become a bottleneck. Because parity data must be skipped on each drive during reads, however, the performance for reads tends to be considerably lower than a level 4 array. The cost per megabyte is the same as for level 4.
The manner data records are stored and retrieved on physical devices .The technique used to find and retrieve store records are called access methods.

Sequential File Organization
Records are arranged on storage devices in some sequence based on the value of some field, called sequence field. Sequence field is often the key field that identifies the record.

Simply, easy to understand and manage, best for providing sequential access. It is not feasible for direct or random access; inserting/deleting a record in/from the middle of the sequence involves cumbersome record searches and rewriting of the file.
RAID-0 is the fastest and most efficient array type but offers no fault-tolerance.
RAID-1 is the array of choice for performance-critical, fault-tolerant environments. In addition, RAID-1 is the only choice for fault-tolerance if no more than two drives are desired.
RAID-2 is seldom used today since ECC is embedded in almost all modern disk drives.
RAID-3 can be used in data intensive or single-user environments which access long sequential records to speed up data transfer. However, RAID-3 does not allow multiple I/O operations to be overlapped and requires synchronized-spindle drives in order to avoid performance degradation with short records.
RAID-4 offers no advantages over RAID-5 and does not support multiple simultaneous write operations.
RAID-5 is the best choices in multi-user environments which are not write performance sensitive. However, at least three and more typically five drives are required for RAID-5 arrays.

Lecture No. 35

Reading Material

“Database Systems Principles, Design and Implementation” written by Catherine Ricardo, Maxwell Macmillan.
“Database Management System” by Jeffery A Hoffer

Overview of Lecture

- Hashing
- Hashing Algorithm
- Collision Handling

In the previous lecture we have studied about different storage media and the RAID levels and we started with file organization. In this lecture we will study in length about different types of file organizations.

File Organizations

This is the most common structure for large files that are typically processed in their entirety, and it's at the heart of the more complex schemes. In this scheme, all the records have the same size and the same field format, with the fields having fixed size as well. The records are sorted in the file according to the content of a field of a scalar type, called ``key''. The key must identify uniquely a records, hence different record have diferent keys. This organization is well suited for batch processing of the entire file, without adding or deleting items: this kind of operation can take advantage of the fixed size of records and file; moreover, this organization is easily stored both on disk and tape. The key ordering, along with the fixed record size, makes this organization amenable to dicotomic search. However, adding and deleting records to this kind of file is a tricky process: the logical sequence of records tipycally matches their physical layout on the media storage, so to ease file navigation, hence adding a record and maintaining the key order requires a reorganization of the whole file. The usual solution is to make use of a “log file'' (also called “transaction file''), structured as a

pile, to perform this kind of modification, and periodically perform a batch update on the master file.

Sequential files provide access only in a particular sequence. That does not suit many applications since it involves too much time. Some mechanism for direct access is required

Direct Access File Organization:
For most users, the file system is the most visible aspect of an operating system. Files store data and programs. The operating system implements the abstract concept of a file by managing mass storage devices, such as tapes and disks. Also files are normally organized into directories to ease their use, so we look at a variety of directory structures. Finally, when multiple users have access to files, it may be desirable to control by whom and in what ways files may be accessed. This control is known as file protection. Following are the two types:

- Indexed Sequential
- Direct File Organization

Indexed sequential file:

An index file can be used to effectively overcome the above-mentioned problem, and to speed up the key search as well. The simplest indexing structure is the single-level one: a file whose records are pair's key-pointer, where the pointer is the position in the data file of the record with the given key. Only a subset of data records, evenly spaced along the data file, are indexed, so to mark intervals of data records. A key search then proceeds as follows: the search key is compared with the index ones to find the highest index key preceding the search one, and a linear search is performed from the record the index key points onward, until the search key is matched or until the record pointed by the next index entry is reached. In spite of the double file access (index + data) needed by this kind of search, the decrease in access time with respect to a sequential file is significant.

A type of file access in which an index is used to obtain the address of the block which contains the required record. An index file can be used to effectively to speed up the key search as well. The simplest indexing structure is the single-level one: a

file whose records are pairs key-pointer, where the pointer is the position in the data file of the record with the given key. Only a subset of data records, evenly spaced along the data file, are indexed, so to mark intervals of data records.

A key search then proceeds as follows: the search key is compared with the index ones to find the highest index key preceding the search one, and a linear search is performed from the record the index key points onward, until the search key is matched or until the record pointed by the next index entry is reached. In spite of the double file access (index + data) needed by this kind of search, the decrease in access time with respect to a sequential file is significant.

Consider, for example, the case of simple linear search on a file with 1,000 records. With the sequential organization, an average of 500 key comparisons are necessary (assuming uniformly distributed search key among the data ones). However, using and evenly spaced index with 100 entries, the number of comparisons is reduced to 50 in the index file plus 50 in the data file: a 5:1 reduction in the number of operations.

This scheme can obviously be hyerarchically extended: an index is a sequential file in itself, amenable to be indexed in turn by a second-level index, and so on, thus exploiting more and more the hyerarchical decomposition of the searches to decrease the access time. Obviously, if the layering of indexes is pushed too far, a point is reached when the advantages of indexing are hampered by the increased storage costs, and by the index access times as well.

Indexed:
Why using a single index for a certain key field of a data record? Indexes can be obviously built for each field that uniquely identifies a record (or set of records within the file), and whose type is amenable to ordering. Multiple indexes hence provide a high degree of flexibility for accessing the data via search on various attributes; this organization also allows the use of variable length records (containing different fields).
It should be noted that when multiple indexes are are used the concept of sequentiality of the records within the file is useless: each attribute (field) used to construct an index typically imposes an ordering of its own. For this very reason is typicaly not possible to use the ``sparse'' (or ``spaced'') type of indexing previously described. Two

types of indexes are usually found in the applications: the exhaustive type, which contains an entry for each record in the main file, in the order given by the indexed key, and the partial type, which contain an entry for all those records that contain the chosen key field (for variable records only).

Defining Keys:

An indexed sequential file must have at least one key. The first (primary) key is always numbered 0. An indexed sequential file can have up to 255 keys; however, for file-processing efficiency it is recommended that you define no more than 7 or 8 keys. (The time required to insert a new record or update an existing record is directly related to the number of keys defined; the retrieval time for an existing record, however, is unaffected by the number of keys.)

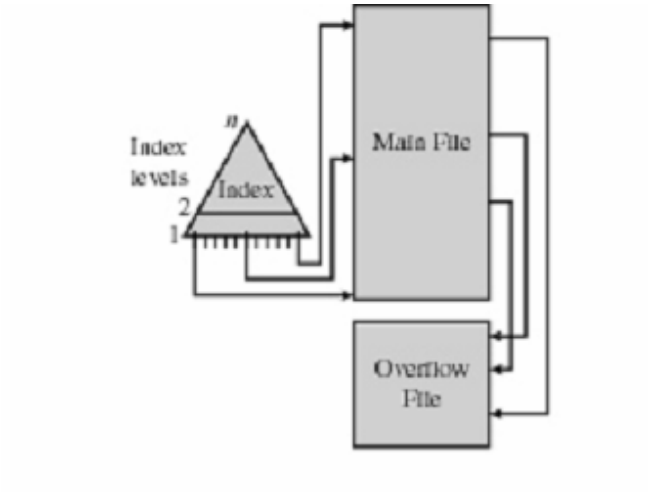
When you design an indexed sequential file, you must define each key in the following terms:

- Position and size
- Data type
- Index number
- Options selected

When you want to define more than one key, or to define keys of different data types, you must be careful when you specify the key fields. The next few subsections describe some considerations for specifying keys. In Indexed sequential files following are ensured:

- New records are added to an overflow file
- Record in main file that precedes it is updated to contain a pointer to the new record
- The overflow is merged with the main file during a batch update
- Multiple indexes for the same key field can be set up to increase efficiency

The diagram of Index sequential file is as under:



Indexed Sequential Summary:
Following are salient features of Indexed sequential file structure:
Records are stored in sequence and index is maintained.
Dense and nondense types of indexes are maintained.
Track overflows and file overflow areas are ensured.
Cylinder index increases the efficiency .

Lecture No. 36

Reading Material

"Database Management Systems", 2nd edition, Raghu Ramakrishnan, Johannes Gehrke, McGraw-Hill
Database Management, Jeffery A Hoffer

Overview of Lecture

- Hashing
- Hashing Algorithm
- Collision Handling

In the previous lecture we have discussed file organization and techniques of file handling. In today's lecture we will study hashing techniques, its algorithms and collision handling.

Hashing

A hash function is computed on some attribute of each record. The result of the function specifies in which block of the file the record should be placed .Hashing provides rapid, non-sequential, direct access to records. A key record field is used to calculate the record address by subjecting it to some calculation; a process called hashing. For numeric ascending order a sequential key record fields this might involve simply using relative address indexes from a base storage address to access records. Most of the time, key field does not have the values in sequence that can directly be used as relative record number. It has to be transformed. Hashing involves computing the address of a data item by computing a function on the search key value.

A hash function h is a function from the set of all search key values *K* to the set of all bucket addresses *B*.

- We choose a number of buckets to correspond to the number of search key values we will have stored in the database.
- To perform a lookup on a search key value K_i, we compute $h(K_i)$, and search the bucket with that address.
- If two search keys *i* and *j* map to the same address, because $h(K_i) = h(K_j)$, then the bucket at the address obtained will contain records with both search key values.
- In this case we will have to check the search key value of every record in the bucket to get the ones we want.
- Insertion and deletion are simple.

Hash Functions

A good hash function gives an average-case lookup that is a small constant, independent of the number of search keys. We hope records are distributed uniformly among the buckets. The worst hash function maps all keys to the same bucket. The best hash function maps all keys to distinct addresses. Ideally, distribution of keys to addresses is uniform and random.

Hashed Access Characteristics

Following are the major characteristics:

- No indexes to search or maintain
- Very fast direct access
- Inefficient sequential access
- Use when direct access is needed, but sequential access is not.

Mapping functions

The direct address approach requires that the function, h(k), is a one-to-one mapping from each k to integers in (1,m). Such a function is known as a perfect hashing function: it maps each key to a distinct integer within some manageable range and enables us to trivially build an O(1) search time table.

Unfortunately, finding a perfect hashing function is not always possible. Let's say that we can find a hash function, h(k), which maps most of the keys onto unique integers, but maps a small number of keys on to the same integer. If the number of collisions

(cases where multiple keys map onto the same integer), is sufficiently small, then hash tables work quite well and give O(1) search times.

Handling the Collisions

In the small number of cases, where multiple keys map to the same integer, then elements with different keys may be stored in the same "slot" of the hash table. It is clear that when the hash function is used to locate a potential match, it will be necessary to compare the key of that element with the search key. But there may be more than one element, which should be stored in a single slot of the table. Various techniques are used to manage this problem:

- Chaining,
- Overflow areas,
- Re-hashing,
- Using neighboring slots (linear probing),
- Quadratic probing,
- Random probing, ...

Chaining:

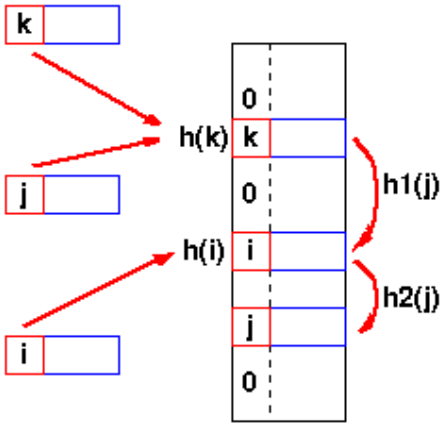
One simple scheme is to chain all collisions in lists attached to the appropriate slot. This allows an unlimited number of collisions to be handled and doesn't require a priori knowledge of how many elements are contained in the collection. The tradeoff is the same as with linked lists versus array implementations of collections: linked list overhead in space and, to a lesser extent, in time.

Re-hashing:

Re-hashing schemes use a second hashing operation when there is a collision. If there is a further collision, we re-hash until an empty "slot" in the table is found. The re-hashing function can either be a new function or a re-application of the original one. As long as the functions are applied to a key in the same order, then a sought key can always be located.

Linear probing:

One of the simplest re-hashing functions is +1 (or -1), ie on a collision; look in the neighboring slot in the table. It calculates the new address extremely quickly and may be extremely efficient on a modern RISC processor due to efficient cache utilization. The animation gives you a practical demonstration of the effect of linear probing: it also implements a quadratic re-hash function so that you can compare the difference.



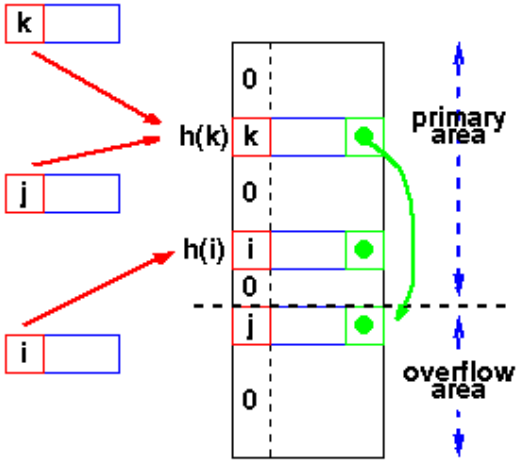
h(j)=h(k), so the next hash function, h1 is used. A second collision occurs, so h2 is used.

Clustering:

Linear probing is subject to a clustering phenomenon. Re-hashes from one location occupy a block of slots in the table which "grows" towards slots to which other keys hash. This exacerbates the collision problem and the number of re-hashed can become large.

Overflow area:

Another scheme will divide the pre-allocated table into two sections: the primary area to which keys are mapped and an area for collisions, normally termed the overflow area.



When a collision occurs, a slot in the overflow area is used for the new element and a link from the primary slot established as in a chained system. This is essentially the same as chaining, except that the overflow area is pre-allocated and thus possibly faster to access. As with re-hashing, the maximum number of elements must be known in advance, but in this case, two parameters must be estimated: the optimum size of the primary and overflow areas.

Of course, it is possible to design systems with multiple overflow tables, or with a mechanism for handling overflow out of the overflow area, which provide flexibility without losing the advantages of the overflow scheme.

Collision handling

A well-chosen hash function can avoid anomalies which are result of an excessive number of collisions, but does not eliminate collisions. We need some method for handling collisions when they occur. We'll consider the following techniques:

- Open addressing
- Chaining

Open addressing:

The hash table is an array of (key, value) pairs. The basic idea is that when a (key, value) pair is inserted into the array, and a collision occurs, the entry is simply inserted at an alternative location in the array. Linear probing, double hashing, and rehashing are all different ways of choosing an alternative location. The simplest probing method is called linear probing. In linear probing, the probe sequence is simply the sequence of consecutive locations, beginning with the hash value of the key. If the end of the table is reached, the probe sequence wraps around and continues at location 0. Only if the table is completely full will the search fail.

Summary Hash Table Organization:

Organization	Advantages	Disadvantages
Chaining	Unlimited number of elements Unlimited number of collisions	Overhead of multiple linked lists
Re-hashing	Fast re-hashing Fast access through use of main table space	Maximum number of elements must be known Multiple collisions may become probable
Overflow area	Fast access Collisions don't use primary table space	Two parameters which govern performance need to be estimated

Lecture No. 37

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management System" by Jeffery A Hoffer

Overview of Lecture:

- Indexes
- Index Classification

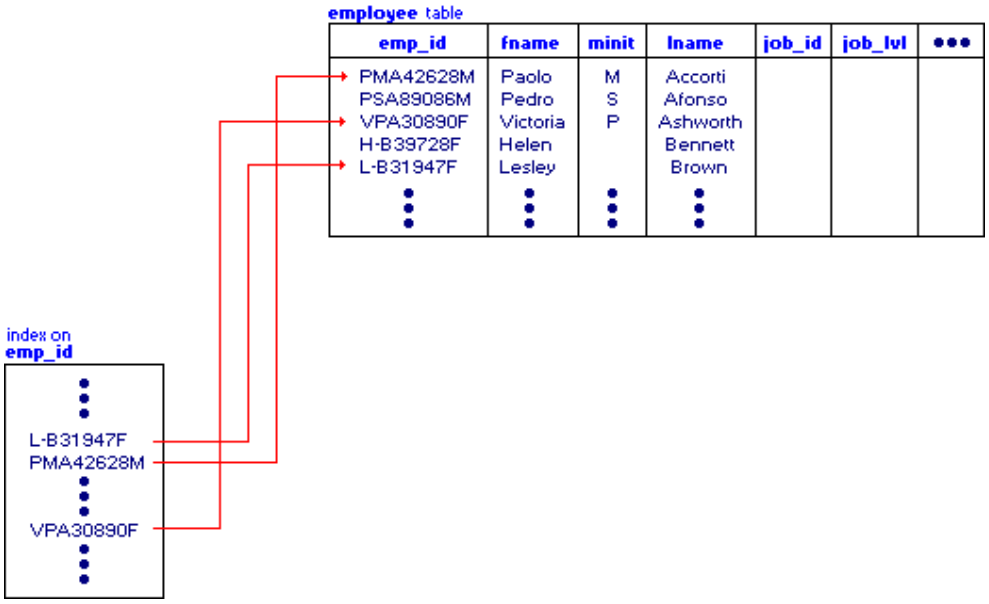
In the previous lecture we have discussed hashing and collision handling. In today's lecture we will discuss indexes, their properties and classification.

Index

In a book, the index is an alphabetical listing of topics, along with the page number where the topic appears. The idea of an INDEX in a Database is similar. We will consider two popular types of indexes, and see how they work, and why they are useful. Any subset of the fields of a relation can be the search key for an index on the relation. Search key is not the same as k*ey* (e.g. doesn't have to be unique ID). An index contains a collection of data entries, and supports efficient retrieval of all records with a given search key value k. typically, index also contains auxiliary information that directs searches to the desired data entries. There can be multiple (different) indexes per file. Indexes on primary key and on attribute(s) in the unique constraint are automatically created. Indexes in databases are similar to indexes in books. In a book, an index allows you to find information quickly without reading the entire book. In a database, an index allows the database program to find data in a table without scanning the entire table. An index in a book is a list of words with the page numbers that contain each word. An index in a database is a list of values in a table with the storage locations of rows in the table that contain each value. Indexes can be

created on either a single column or a combination of columns in a table and are implemented in the form of B-trees. An index contains an entry with one or more columns (the search key) from each row in a table. A B-tree is sorted on the search key, and can be searched efficiently on any leading subset of the search key. For example, an index on columns A, B, C can be searched efficiently on A, on A, B, and A, B, C. Most books contain one general index of words, names, places, and so on. Databases contain individual indexes for selected types or columns of data: this is similar to a book that contains one index for names of people and another index for places. When you create a database and tune it for performance, you should create indexes for the columns used in queries to find data.

In the pubs sample database provided with Microsoft® SQL Server™ 2000, the employee table has an index on the emp_id column. The following illustration shows that how the index stores each emp_id value and points to the rows of data in the table with each value.



When SQL Server executes a statement to find data in the employee table based on a specified emp_id value, it recognizes the index for the emp_id column and uses the index to find the data. If the index is not present, it performs a full table scan starting at the beginning of the table and stepping through each row, searching for the specified emp_id value. SQL Server automatically creates indexes for certain types of

constraints (for example, PRIMARY KEY and UNIQUE constraints). You can further customize the table definitions by creating indexes that are independent of constraints.

The performance benefits of indexes, however, do come with a cost. Tables with indexes require more storage space in the database. Also, commands that insert, update, or delete data can take longer and require more processing time to maintain the indexes. When you design and create indexes, you should ensure that the performance benefits outweigh the extra cost in storage space and processing resources.

Index Classification

Indexes are classified as under:

- Clustered vs. Un-clustered Indexes
- Single Key vs. Composite Indexes
- Tree-based, inverted files, pointers

Primary Indexes:

Consider a table, with a Primary Key Attribute being used to store it as an ordered array (that is, the records of the table are stored in order of increasing value of the Primary Key attribute.)As we know, each BLOCK of memory will store a few records of this table. Since all search operations require transfers of complete blocks, to search for a particular record, we must first need to know which block it is stored in. If we know the address of the block where the record is stored, searching for the record is very fast. Notice also that we can order the records using the Primary Key attribute values. Thus, if we just know the primary key attribute value of the first record in each block, we can determine quite quickly whether a given record can be found in some block or not. This is the idea used to generate the Primary Index file for the table.

Secondary Indexes:

Users often need to access data on the basis of non-key or non-unique attribute; secondary key. Like student name, program name, students enrolled in a particular program .Records are stored on the basis of key attribute; three possibilities

- Sequential search
- Sorted on the basis of name

- Sort for command execution

A part from primary indexes, one can also create an index based on some other attribute of the table. We describe the concept of Secondary Indexes for a Key attribute that is, for an attribute which is not the Primary Key, but still has unique values for each record of the table The idea is similar to the primary index. However, we have already ordered the records of our table using the Primary key. We cannot order the records again using the secondary key (since that will destroy the utility of the Primary Index Therefore, the Secondary Index is a two column file, which stores the address of every tuple of the table.

Following are the three-implementation approaches of Indexes:

- Inverted files or inversions
- Linked lists
- B+ Trees

Creating Index

CREATE [UNIQUE]

[CLUSTERED | NONCLUSTERED]

INDEX *index_name*

ON { *table* | *view* } **(** *column* [ASC | DESC] [**,***...n*]

We will now see an example of creating index as under:

CREATE UNIQUE INDEX pr_prName

ON program(prName)

It can also be created on composite attributes. We will see it with an example.

CREATE UNIQUE INDEX

St_Name ON

Student(stName ASC, stFName DESC)

Properties of Indexes:

Following are the major properties of indexes:

- Indexes can be defined even when there is no data in the table
- Existing values are checked on execution of this command
- It support selections of form as under:

  field <operator> constant

- It support equality selections as under:
  Either “tree” or “hash” indexes help here.
- It support Range selections (operator is one among <, >, <=, >=, BETWEEN)

Summary

In today’s lecture we have discussed the indexes, its classification and creating the indexes as well. The absence or presence of an index does not require a change in the wording of any SQL statement. An index merely offers a fast access path to the data; it affects only the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value. Indexes are logically and physically independent of the data in the associated table. You can create or drop an index at anytime without affecting the base tables or other indexes. If you drop an index, all applications continue to work; however, access to previously indexed data might be slower. Indexes, being independent structures, require storage space.

Lecture No. 38

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management System" by Jeffery A Hoffer

Overview of Lecture

Indexes
Clustered Versus Un-clustered Indexes
Dense Verses Sparse Indexes
Primary and Secondary Indexes
Indexes Using Composite Search Keys

In the previous lecture we studied about what the indexes are and what is the need for creating an index. There exist a number of index types which are important and are helpful for the file organization in database environments for the storage of files on disks.

Ordered Indices

In order to allow fast **random** access, an index structure may be used.
A file may have several indices on different search keys.
If the file containing the records is sequentially ordered, the index whose search key specifies the sequential order of the file is the **primary index**, or **clustering index**.
Note: The search key of a primary index is usually the primary key, but it is not necessarily so.
Indices whose search key specifies an order different from the sequential order of the file are called the **secondary indices**, or **nonclustering indices**.

Clustered Indexes

A **clustered index** determines the storage order of data in a table. A clustered index is analogous to a telephone directory, which arranges data by last name. Because the clustered index dictates the physical storage order of the data in the table, a table can contain only one clustered index. However, the index can comprise multiple columns (a composite index), like the way a telephone directory is organized by last name and first name.
A clustered index is particularly efficient on columns often searched for ranges of values. Once the row with the first value is found using the clustered index, rows with subsequent indexed values are guaranteed to be physically adjacent. For example, if an application frequently executes a query to retrieve records between a range of dates, a clustered index can quickly locate the row containing the beginning date, and then retrieve all adjacent rows in the table until the last date is reached. This can help increase the performance of this type of query. Also, if there is a column(s) which is used frequently to sort the data retrieved from a table, it can be advantageous to

cluster (physically sort) the table on that column(s) to save the cost of a sort each time the column(s) is queried.
Clustered indexes are also efficient for finding a specific row when the indexed value is unique. For example, the fastest way to find a particular employee using the unique employee ID column **emp_id** would be to create a clustered index or PRIMARY KEY constraint on the **emp_id** column.
Note PRIMARY KEY constraints create clustered indexes automatically if no clustered index already exists on the table and a nonclustered index is not specified when you create the PRIMARY KEY constraint.
Alternatively, a clustered index could be created on **lname, fname** (last name, first name), because employee records are often grouped and queried that way rather than by employee ID.

Non-clustered Indexes

Nonclustered indexes have the same B-tree structure as clustered indexes, with two significant differences:
The data rows are not sorted and stored in order based on their nonclustered keys.
The leaf layer of a nonclustered index does not consist of the data pages.
Instead, the leaf nodes contain index rows. Each index row contains the nonclustered key value and one or more row locators that point to the data row (or rows if the index is not unique) having the key value.
Nonclustered indexes can be defined on either a table with a clustered index or a heap.
In Microsoft® SQL Server™ version 7.0, the row locators in nonclustered index rows have two forms:
If the table is a heap (does not have a clustered index), the row locator is a pointer to the row. The pointer is built from the file ID, page number, and number of the row on the page. The entire pointer is known as a Row ID.
If the table does have a clustered index, the row locator is the clustered index key for the row. If the clustered index is not a unique index, SQL Server 7.0 adds an internal value to duplicate keys to make them unique. This value is not visible to users; it is used to make the key unique for use in nonclustered indexes. SQL Server retrieves the data row by searching the clustered index using the clustered index key stored in the leaf row of the nonclustered index.
Because nonclustered indexes store clustered index keys as their row locators, it is important to keep clustered index keys as small as possible. Do not choose large columns as the keys to clustered indexes if a table also has nonclustered indexes.

Dense and Sparse Indices

There are Two types of ordered indices:

Dense Index:

An index record appears for every search key value in file.
This record contains search key value and a pointer to the actual record.

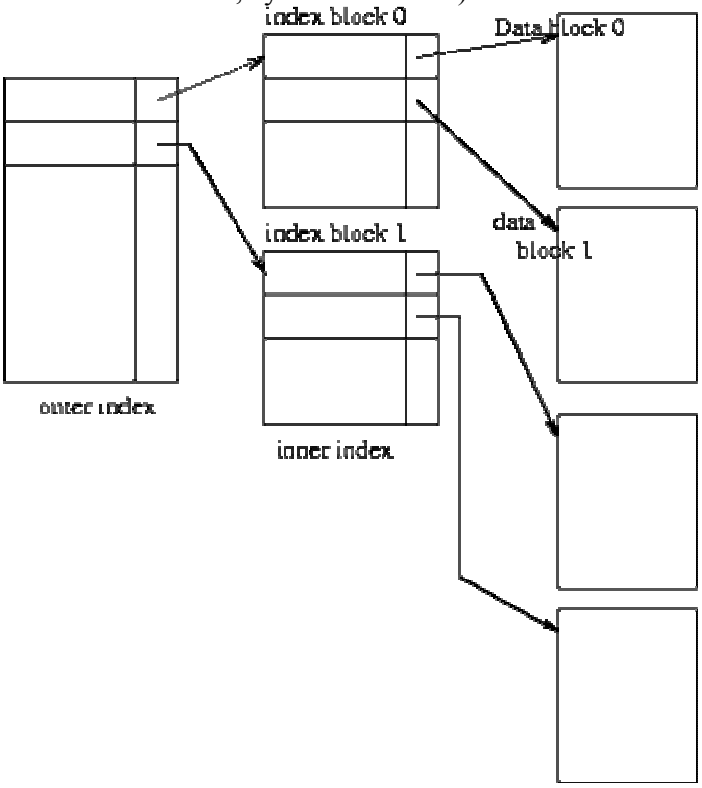
Sparse Index:

Index records are created only for some of the records.
To locate a record, we find the index record with the largest search key value less than or equal to the search key value we are looking for.
We start at that record pointed to by the index record, and proceed along the pointers in the file (that is, sequentially) until we find the desired record.
Dense indices are faster in general, but sparse indices require less space and impose less maintenance for insertions and deletions.
We can have a good compromise by having a sparse index with one entry per block. It has several advantages.

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

Multi-Level Indices

Even with a sparse index, index size may still grow too large. For 100,000 records, 10 per block, at one index record per block, that's 10,000 index records! Even if we can fit 100 index records per block, this is 100 blocks.
If index is too large to be kept in main memory, a search results in several disk reads.
If there are no overflow blocks in the index, we can use binary search.
This will read as many as $1 + \log_2(b)$ blocks (as many as 7 for our 100 blocks).
If index has overflow blocks, then sequential search typically used, reading **all** *b* index blocks.
Use binary search on outer index. Scan index block found until correct index record found. Use index record as before - scan block pointed to for desired record.
For very large files, additional levels of indexing may be required.
Indices must be updated at all levels when insertions or deletions require it.
Frequently, each level of index corresponds to a unit of physical storage (e.g. indices at the level of track, cylinder and disk).



Two-level sparse index.

Index Update

Regardless of what form of index is used, every index must be updated whenever a record is either inserted into or deleted from the file.

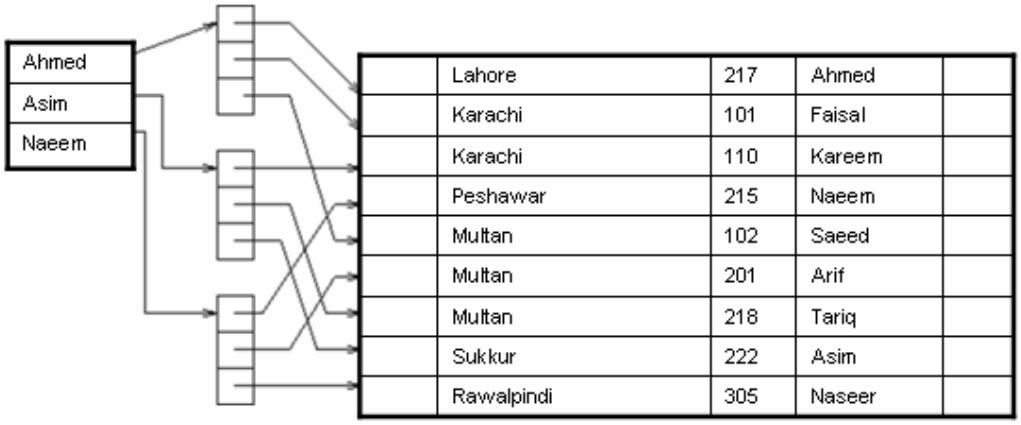
Deletion:

Find (look up) the record
If the last record with a particular search key value, delete that search key value from index.

For dense indices, this is like deleting a record in a file.
For sparse indices, delete a key value by replacing key value's entry in index by next search key value. If that value already has an index entry, delete the entry.

Insertion:

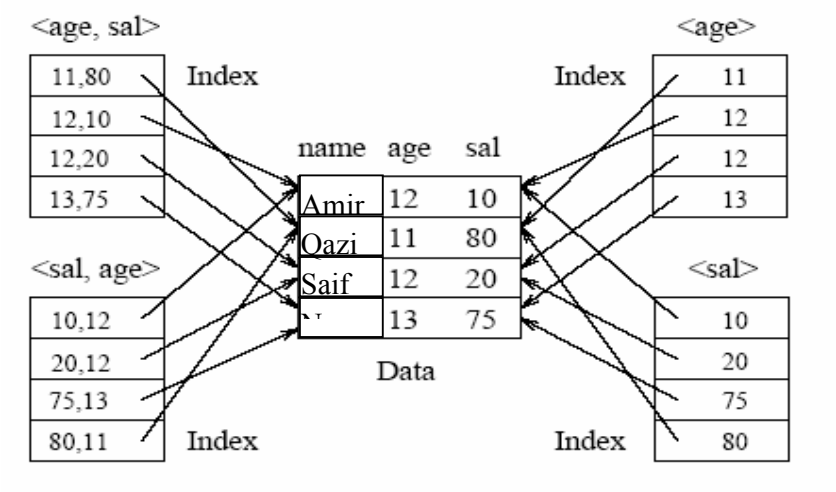
Find place to insert.
Dense index: insert search key value if not present.
Sparse index: no change unless new block is created. (In this case, the first search key value appearing in the new block is inserted into the index).
Secondary Indices
If the search key of a secondary index is not a candidate key, it is not enough to point to just the first record with each search-key value because the remaining records with the same search-key value could be anywhere in the file. Therefore, a secondary index must contain pointers to all the records.



Sparse secondary index on *sname*.
We can use an extra-level of indirection to implement secondary indices on search keys that are not candidate keys. A pointer does not point directly to the file but to a bucket that contains pointers to the file.
See Figure above on secondary key *sname*.
To perform a lookup on Peterson, we must read all three records pointed to by entries in bucket 2.
Only one entry points to a Peterson record, but three records need to be read.
As file is not ordered physically by *cname*, this may take 3 block accesses.
Secondary indices must be **dense**, with an index entry for every search-key value, and a pointer to every record in the file.
Secondary indices improve the performance of queries on non-primary keys.
They also impose serious overhead on database modification: whenever a file is updated, every index must be updated.
Designer must decide whether to use secondary indices or not.

Indexes Using Composite Search Keys
The search key for an index can contain several fields; such keys are called **composite search keys** or **concatenated keys**. As an example, consider a collection of employee records, with fields *name, age*, and *sal*, stored in sorted order by *name*. Figure below illustrates the difference between a composite index with key (*age, sal) and* a

composite index with key (*sal, age)*, an index with key *age*, and an index with key *sal*. All indexes which are shown in the figure use Alternative (2) for data entries.



If the search key is composite, an equality query is one in which each field in the search key is bound to a constant. For example, we can ask to retrieve all data entries with age = 20 and sal = 10. The hashed file organization supports only equality queries, since a hash function identifies the bucket containing desired records only if a value is specified for each field in the search key. A range query is one in which not all fields in the search key are bound to constants. For example, we can ask to retrieve all data entries with age = 20; this query implies that any value is acceptable for the sal field. As another example of a range query, we can ask to retrieve all data entries with age < 30 and sal > 40.

Lecture No. 39 and 40

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management System" by Jeffery A Hoffer

Overview of Lecture

- Introduction to Views
- Views, Data Independence, Security
- Choosing a Vertical and Horizontal Subset of a Table
- A View Using Two Tables
- A View of a View
- A View Using a Function
- Updates on Views

Views

Views are generally used to focus, simplify, and customize the perception each user has of the database. Views can be used as security mechanisms by allowing users to access data through the view, without granting the users permissions to directly access the underlying base tables of the view.

To Focus on Specific Data

Views allow users to focus on specific data that interests them and on the specific tasks for which they are responsible. Unnecessary data can be left out of the view. This also increases the security of the data because users

can see only the data that is defined in the view and not the data in the underlying table.

A database view displays one or more database records on the same page. A view can display some or all of the database fields. Views have filters to determine which records they show. Views can be sorted to control the record order and grouped to display records in related sets. Views have other options such as totals and subtotals.

Most users interact with the database using the database views. A key to creating a useful database is a well-chosen set of views. Luckily, while views are powerful, they are also easy to create.

A "view" is essentially a dynamically generated "result" table that is put together based upon the parameters you have defined in your query. For example, you might instruct the database to give you a list of all the employees in the EMPLOYEES table with salaries greater than 50,000 USD per year. The database would check out the EMPLOYEES table and return the requested list as a "virtual table".

Similarly, a view could be composed of the results of a query on several tables all at once (sometimes called a "join"). Thus, you might create a view of all the employees with a salary of greater than 50K from several stores by

accumulating the results from queries to the EMPLOYEES and STORES databases. The possibilities are limitless.

You can customize all aspects of a view, including:

- The name of the view
- The fields that appear in the view
- The column title for each field in the view
- The order of the fields in the view
- The width of columns in the view, as well as the overall width of the view
- The set of records that appear in the view (Filtering)
- The order in which records are displayed in the view (Sorting & Grouping)
- Column totals for numeric and currency fields (Totaling & Subtotaling)

The physical schema for a relational database describes how the relations in the conceptual schema are stored, in terms of the file organizations and indexes used. The conceptual schema is the collection of schemas of the relations stored in the database. While some relations in the conceptual schema can also be exposed to applications, i.e., be part of the external schema of the database, additional relations in the external schema can be defined using the view mechanism. The view mechanism thus provides the support for logical data independence in the relational model. That is, it can be used to define relations in the external schema that mask changes in the conceptual schema of the database from applications. For example, if the schema of a stored relation is changed, we can define a view with the old schema, and applications that expect to see the old schema can now use this view. Views are also valuable in the context of security: We can define views

that give a group of user's access to just the information they are allowed to see. For example, we can define a view that allows students to see other students' name and age but not their GPA, and allow all students to access this view, but not the underlying Students table.

There are two ways to create a new view in your database. You can:

- Create a new view from scratch.
- Or, make a copy of an existing view and then modify it.

Characteristics /Types of Views:

We have a number of views type of which some of the important views types are listed below:

- **Materialized View**
- **Simple Views**
- **Complex View**
- **Dynamic Views.**

A materialized view is a replica of a target master from a single point in time. The master can be either a master table at a master site or a master materialized view at a materialized view site. Whereas in multi-master replication tables are continuously updated by other master sites, materialized views are updated from one or more masters through individual batch updates, known as a **refreshes**, from a single master site or master materialized view site

Simple Views

As defined earlier simple views are created from tables and are used for creating secure manipulation over the tables or structures of the database. Views make the manipulations easier to perform on the database.

Complex Views

Complex views are by definition views of type which may comprise of many of elements, such as tables, views sequences and other similar objects of the database. When talking about the views we can have views of one table, views of one table and one view, views of multiple tables views of multiple views and so on…

Dynamic Views

Dynamic views are those types of views for which data is not stored and the expressions used to build the view are used to collect the data dynamically. These views are not executed only once when they are referred for the first time, rather they are created and the data contained in such views is updated every time the view is accessed or used in any other view or query.

Dynamic views generally are complex views, views of views, and views of multiple tables.

An example of a dynamic view creation is given below:

```
CREATE VIEW st_view1 AS (select stName, stFname, prName
FROM student
WHERE prName = 'MCS')
```

Views can be referred in SQL statements like tables

We can have view created on functions and other views as well. Where the function used for the view creation and the other nested view will be used as a simple table or relation.

Examples:

View Using another View

```
CREATE VIEW CLASSLOC2
AS SELECT COURSE#, ROOM
FROM CLASSLOC
```

View Using Function

```
CREATE VIEW CLASSCOUNT(COURSE#, TOTCOUNT)
    AS SELECT COURSE#,      COUNT(*)
    FROM ENROLL
    GROUP BY COURSE#;
```

Dynamic Views

SELECT * FROM st_view1

	stName	stFname	prName
1	Amjad	Hussain	MCS
2	Amjad	Rehan	MCS
3	Tahira Ejaz	Nek Muhammad	MCS
4	Suhal Dar	Loving	MCS

With Check Option

```
CREATE VIEW st_view2
AS (SELECT stName, stFname, prName FROM student WHERE prName = 'BCS')
WITH CHECK OPTION
```

```
UPDATE ST_VIEW1 set prName = 'BCS'
Where stFname = 'Loving'
```

SELECT * from ST_VIEW1

	stName	stFname	prName
1	Amjad	Hussain	MCS
2	Amjad	Rehan	MCS
3	Tahira Ejaz	Nek Muhammad	MCS

SELECT * FROM ST_VIEW2

	stName	stFname	prName
1	Suhal Dar	Loving	BCS
2	Shoaib Ali	Rahmat Ali	BCS

Update ST_VIEW2 set prName = 'MCS'
Where stFname = 'Loving'

```
Server: Msg 550, Level 16, State 1, Line 1
The attempted insert or update failed because the target view either specifies WITH CHECK OPTION or spans a view that
The statement has been terminated.
```

Characteristics of Views

- Computed attributes
- Nesting of views

CREATE VIEW enr_view AS (select * from enroll)

CREATE VIEW enr_view1 as (select stId, crcode, smrks, mterm, smrks + mterm sessional from enr_view)

Select * from enr_view1

	stId	crcode	smrks	mterm	sessional
1	S1015	CS-516	12	32	44
2	S1015	CS-616	10	30	40
3	S1018	MG-314	10	26	36
4	S1020	CS-516	13	26	39
5	S1020	CS-616	12	25	37

Deleting Views:

A view can be dropped using the DROP VIEW command, which is just like DROP TABLE.

Updates on Views

Updating a view is quite simple and is performed in the same way as we perform updates on any of the database relations. But this simplicity is limited to those views only which are created using a single relation. Those views which comprise of multiple relations the updation are hard to perform and needs additional care and precaution.

As we know that the views may contain some fields which are not the actual data fields in the relation but may also contain computed attributes. So update or insertions in this case are not performed through the views created on these tables.

Lecture No. 41

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management System" by Jeffery A Hoffer

Overview of Lecture

- Indexes
- Index Classification

In our previous lecture we were discussing the views. Views play an important role in database. At this layer database is available to the users. The user needs to know that they are dealing with views; it is infact virtual for them. It can be used to implement security. We were discussing dynamic views whose data is not stored as such.

Updating Multiple Tables

We can do this updation of multiple views by doing it one at a time. It means that while inserting values in different tables, it can only be done one at a time. We will now see an example of this as under:

CREATE VIEW st_pr_view1 (a1, a2, a3, a4) AS (select stId, stName, program.prName, prcredits from student, program WHERE student.prName = program.prName)
In this example this is a join statement

We will now enter data in the table

insert into st_pr_view1 (a3, a4) values ('MSE', 110)
We will now see the program table after writing this SQL statement as the data has been stored in the table.
Select * from program

	prName	totSem	prCredits
1	BBA	8	130
2	BCE	8	134
3	BCS	8	134
4	BIT	8	132
5	MBA	4	64
6	MCS	4	64
7	MIT	4	62
8	MSE	NULL	110

In this example the total semester is NULL as this attribute was not defined in view creation statement, so then this value will remain NULL. We will now see another example. In this we have catered for NOT NULL.
insert into st_pr_view1 (a1, a2) values ('S1043', 'Bilal Masood')

SELECT * from student

	stId	stName	stFName	stAdres	stPhone	prName	curSem
1	S0123	Amjad	Hussain	8-SD Lahore	234322	MCS	1
2	S1012	Amjad	Rehan	I8 Ibd	5456754	MCS	2
3	S1015	Tahira Ejaz	Nek Muhammad	AJ Road	4323456	MCS	3
4	S1018	Arif Zia	Zia Khan	GM Rawalpindi	4356488	BIT	2
5	S1020	Suhal Dar	Loving	I-8 Islamabad	5523240	BCS	2
6	S1021	M. Ali	M. Saad	JT Lahore	544325	MBA	2
7	S1034	Sadia Zia	NULL	NULL	NULL	BIT	1
8	S1038	Shoaib Ali	Rahmat Ali	G-6 Islamabad	5343240	BCS	2
9	S1040	Ahmad Ali	Ali Hussain	NULL	NULL	BCS	1
10	S1042	Ahmad Ali	Ali Hussain	NULL	NULL	BCS	1
11	S1044	Bilal Moasood	NULL	NULL	NULL	NULL	1

Materialized Views

A pre-computed table comprising aggregated or joined data from fact and possibly dimensions tables. Also known as summary or aggregate table.Views are virtual tables. In which query is executed every time .For complex queries involving large number of join rows and aggregate functions, so it is problematic. Its solution is materialized views also called indexed views created through clustered index. Creating a clustered index on a view stores the result set built at the time the index is created. An indexed view also automatically reflects modifications made to the data in

the base tables after the index is created, the same way an index created on a base table does. Create indexes only on views where the improved speed in retrieving results outweighs the increased overhead of making modifications.

Materialized views are schema objects that can be used to summarize, compute, replicate, and distribute data. They are suitable in various computing environments such as data warehousing, decision support, and distributed or mobile computing:

- In data warehouses, materialized views are used to compute and store aggregated data such as sums and averages. Materialized views in these environments are typically referred to as summaries because they store summarized data. They can also be used to compute joins with or without aggregations. If compatibility is set to Oracle9*i* or higher, then materialized views can be used for queries that include filter selections.

  Cost-based optimization can use materialized views to improve query performance by automatically recognizing when a materialized view can and should be used to satisfy a request. The optimizer transparently rewrites the request to use the materialized view. Queries are then directed to the materialized view and not to the underlying detail tables or views.

- In distributed environments, materialized views are used to replicate data at distributed sites and synchronize updates done at several sites with conflict resolution methods. The materialized views as replicas provide local access to data that otherwise have to be accessed from remote sites.
- In mobile computing environments, materialized views are used to download a subset of data from central servers to mobile clients, with periodic refreshes from the central servers and propagation of updates by clients back to the central servers.

Materialized views are similar to indexes in several ways:

- They consume storage space.
- They must be refreshed when the data in their master tables changes.

- They improve the performance of SQL execution when they are used for query rewrites.
- Their existence is transparent to SQL applications and users.

Unlike indexes, materialized views can be accessed directly using a SELECT statement. Depending on the types of refresh that are required, they can also be accessed directly in an INSERT, UPDATE, or DELETE statement.
A materialized view can be partitioned. You can define a materialized view on a partitioned table and one or more indexes on the materialized view.

Transaction Management

A transaction can be defined as an indivisible unit of work comprised of several operations, all or none of which must be performed in order to preserve data integrity. For example, a transfer of Rs 1000 from your checking account to your savings account would consist of two steps: debiting your checking account by Rs1000 and crediting your savings account with Rs1000. To protect data integrity and consistency and the interests of the bank and the customer these two operations must be applied together or not at all. Thus, they constitute a transaction.

Properties of a transaction

All transactions share these properties: atomicity, consistency, isolation, and durability (represented by the acronym ACID).

- Atomicity: This implies indivisibility; any indivisible operation (one which will either complete fully or not at all) is said to be atomic.
- Consistency: A transaction must transition persistent data from one consistent state to another. If a failure occurs during processing, the data must be restored to the state it was in prior to the transaction.
- Isolation: Transactions should not affect each other. A transaction in progress, not yet committed or rolled back (these terms are explained at the end of this section), must be isolated from other transactions. Although several transactions may run concurrently, it should appear to each that all the others

completed before or after it; all such concurrent transactions must effectively end in sequential order.

- Durability: Once a transaction has successfully committed, state changes committed by that transaction must be durable and persistent, despite any failures that occur afterwards.

A transaction can thus end in two ways: a commit, the successful execution of each step in the transaction, or a rollback, which guarantees that none of the steps are executed due to an error in one of those steps.

Lecture No. 42

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.
"Database Management System" by Jeffery A Hoffer

Overview of Lecture

- Transaction Management
- The Concept of a Transaction
- Transactions and Schedules
- Concurrent Execution of Transactions
- Need for Concurrency Control
- Serializability
- Locking
- Deadlock

A transaction is defined as any one execution of a user program in a DBMS and differs from an execution of a program outside the DBMS (e.g., a C program executing on Unix) in important ways. (Executing the same program several times will generate several transactions.)

For performance reasons, a DBMS has to interleave the actions of several transactions. However, to give users a simple way to understand the effect of running their programs, the interleaving is done carefully to ensure that the result of a concurrent execution of transactions is nonetheless equivalent (in its effect upon the database) to some serial, or one-at-a-time, execution of the same set of transactions.

The Concept of a Transaction

A user writes data access/update programs in terms of the high-level query and update language supported by the DBMS. To understand how the DBMS handles such requests, with respect to concurrency control and recovery, it is convenient to regard an execution of a user program, or transaction, as a series of reads and writes of database objects:

- To read a database object, it is first brought into main memory (specifically, some frame in the buffer pool) from disk, and then its value is copied into a program variable.
- To write a database object, an in-memory copy of the object is first modified and then written to disk.

Database `objects' are the units in which programs read or write information. The units could be pages, records, and so on, but this is dependent on the DBMS and is not central to the principles underlying concurrency control or recovery. In this chapter, we will consider a database to be a fixed collection of independent objects.

There are four important properties of transactions that a DBMS must ensure to maintain data in the face of concurrent access and system failures:

- Users should be able to regard the execution of each transaction as atomic: either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions (say, when a system crash occurs).
- Each transaction, run by itself with no concurrent execution of other transactions, must preserve the consistency of the database. This property is called consistency, and the DBMS assumes that it holds for each transaction. Ensuring this property of a transaction is the responsibility of the user.
- Users should be able to understand a transaction without considering the effect of other concurrently executing transactions, even if the DBMS interleaves the actions of several transactions for performance reasons. This property is sometimes referred to as isolation: Transactions are isolated, or protected, from the effects of concurrently scheduling other transactions.
- Once the DBMS informs the user that a transaction has been successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called durability.

The acronym ACID is sometimes used to refer to the four properties of transactions which are presented here: atomicity, consistency, isolation and durability.

Transactions and Schedules

A transaction is seen by the DBMS as a series, or list, of actions. The actions that can be executed by a transaction include reads and writes of database objects. A transaction can also be defined as a set of actions that are partially ordered. That is, the relative order of some of the actions may not be important. In order to concentrate on the main issues, we will treat transactions (and later, schedules) as a list of actions. Further, to keep our notation simple, we'll assume that an object O is always read into a program variable that is also named O. We can therefore denote the action of a transaction T reading an object O as RT (O); similarly, we can denote writing as WT (O). When the transaction T is clear from the context, we will omit the subscript.

A schedule is a list of actions (reading, writing, aborting, or committing) from a set of transactions, and the order in which two actions of a transaction T appear in a schedule must be the same as the order in which they appear in T. Intuitively, a schedule represents an actual or potential execution sequence. For example, the schedule in Figure below shows an execution order for actions of two transactions T1 and T2. We move forward in time as we go down from one row to the next. We emphasize that a schedule describes the actions of transactions as seen by the DBMS. In addition to these actions, a transaction may carry out other actions, such as reading or writing from operating system files, evaluating arithmetic expressions, and so on.

Concurrent Execution of Transactions

The DBMS interleaves the actions of different transactions to improve performance, in terms of increased throughput or improved response times for short transactions, but not all interleaving should be allowed.

Ensuring transaction isolation while permitting such concurrent execution is difficult but is necessary for performance reasons. First, while one transaction is waiting for a page to be read in from disk, the CPU can process another transaction. This is because I/O activity can be done in parallel with CPU activity in a computer. Overlapping I/O and CPU activity reduces the amount of time disks and processors are idle, and increases system throughput (the average number of transactions completed in a given time). Second, interleaved execution of a short transaction with a long transaction usually allows the short transaction to complete quickly. In serial execution, a short

transaction could get stuck behind a long transaction leading to unpredictable delays in response time, or average time taken to complete a transaction.

$T1$	$T2$
$R(A)$	
$W(A)$	
	$R(B)$
	$W(B)$
$R(C)$	
$W(C)$	

Figure: Transaction Scheduling

Serializability

To begin with, we assume that the database designer has defined some notion of a consistent database state. For example, we can define a consistency criterion for a university database to be that the sum of employee salaries in each department should be less than 80 percent of the budget for that department. We require that each transaction must preserve database consistency; it follows that any serial schedule that is complete will also preserve database consistency. That is, when a complete serial schedule is executed against a consistent database, the result is also a consistent database.

A serializable schedule over a set S of committed transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over S. That is, the database instance that results from executing the given schedule is identical to the database instance that results from executing the transactions in some serial order. There are some important points to note in this definition:

- Executing the transactions serially in different orders may produce different results, but all are presumed to be acceptable; the DBMS makes no guarantees about which of them will be the outcome of an interleaved execution.
- The above definition of a serializable schedule does not cover the case of schedules containing aborted transactions
- If a transaction computes a value and prints it to the screen, this is an `effect' that is not directly captured in the state of the database. We will assume that all such values are also written into the database, for simplicity.

Lock-Based Concurrency Control

A DBMS must be able to ensure that only serializable, recoverable schedules are allowed, and that no actions of committed transactions are lost while undoing aborted transactions. A DBMS typically uses a locking protocol to achieve this. A locking protocol is a set of rules to be followed by each transaction (and enforced by the DBMS), in order to ensure that even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions in some serial order.

Strict Two-Phase Locking (Strict 2PL)

The most widely used locking protocol, called Strict Two-Phase Locking, or Strict 2PL, has two rules.

The first rule is

If a transaction T wants to read (respectively, modify) an object, it first requests a shared (respectively, exclusive) lock on the object. Of course, a transaction that has an exclusive lock can also read the object; an additional shared lock is not required. A transaction that requests a lock is suspended until the DBMS is able to grant it the requested lock. The DBMS keeps track of the locks it has granted and ensures that if a transaction holds an exclusive lock on an object, no other transaction holds a shared or exclusive lock on the same object. The second rule in Strict 2PL is:

All locks held by a transaction are released when the transaction is completed. Requests to acquire and release locks can be automatically inserted into transactions by the DBMS; users need not worry about these details. In effect the locking protocol allows only `safe' interleaving of transactions. If two transactions access completely independent parts of the database, they will be able to concurrently obtain the locks that they need and proceed merrily on their ways. On the other hand, if two transactions access the same object, and one of them wants to modify it, their actions are effectively ordered serially all actions of one of these transactions (the one that gets the lock on the common object first) are completed before (this lock is released and) the other transaction can proceed.

We denote the action of a transaction T requesting a shared (respectively, exclusive) lock on object O as ST (O) (respectively, XT (O)), and omit the subscript denoting the transaction when it is clear from the context. As an example, consider the schedule shown in Figure below.

$T1$	$T2$
$R(A)$	
$W(A)$	
	$R(A)$
	$W(A)$
	$R(B)$
	$W(B)$
	Commit
$R(B)$	
$W(B)$	
Commit	

Figure reading uncommitted Data

This interleaving could result in a state that cannot result from any serial execution of the three transactions. For instance, T1 could change A from 10 to 20, then T2 (which reads the value 20 for A) could change B from 100 to 200, and then T1 would read the value 200 for B. If run serially, either T1 orT2 would execute first, and read the values 10 for A and 100 for B: Clearly, the interleaved execution is not equivalent to either serial execution. If the Strict 2PL protocol is used, the above interleaving is disallowed. Let us see why. Assuming that the transactions proceed at the same relative speed as before, T1 would obtain an exclusive lock on A first and then read and write A (figure below) Then, T2 would request a lock on A. However, this request cannot be granted until T1 releases its exclusive lock on A, and the DBMS therefore suspends T2. T1 now proceeds to obtain an exclusive lock on B, reads and writes B, then finally commits, at which time its locks are released. T2's lock request is now granted, and it proceeds. In this example the locking protocol results in a serial execution of the two transactions. In general, however, the actions of different transactions could be interleaved. As an example, consider the interleaving of two transactions shown in Figure below, which is permitted by the Strict 2PL protocol.

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	

Figure: Schedule illustrating strict 2PL.

Deadlocks

Consider the following example: transaction T1 gets an exclusive lock on object A, T2 gets an exclusive lock on B, T1 requests an exclusive lock on B and is queued, and T2 requests an exclusive lock on A and is queued. Now, T1 is waiting for T2 to release its lock and T2 is waiting for T1 to release its lock! Such a cycle of transactions waiting for locks to be released is called a deadlock. Clearly, these two transactions will make no further progress. Worse, they hold locks that may be required by other transactions. The DBMS must either prevent or detect (and resolve) such deadlock situations.

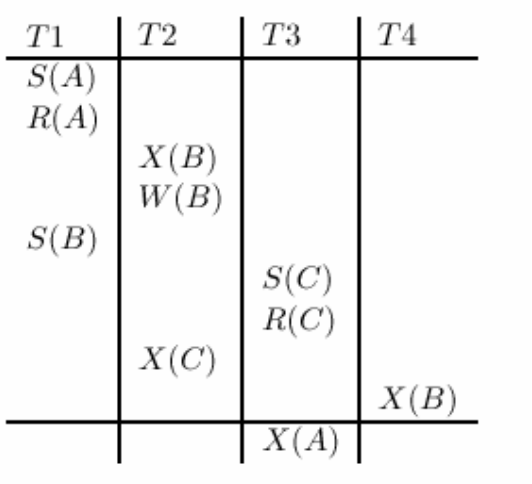


Figure :Schedule Illustrating Deadlock

Deadlock Prevention

We can prevent deadlocks by giving each transaction a priority and ensuring that lower priority transactions are not allowed to wait for higher priority transactions (or vice versa). One way to assign priorities is to give each transaction a timestamp when it starts up. The lower the timestamp, the higher the transaction's priority, that is, the oldest transaction has the highest priority.

If a transaction Ti requests a lock and transaction Tj holds a conflicting lock, the lock manager can use one of the following two policies:

- Wait-die: If Ti has higher priority, it is allowed to wait; otherwise it is aborted.
- Wound-wait: If Ti has higher priority, abort Tj; otherwise Ti waits.

In the wait-die scheme, lower priority transactions can never wait for higher priority transactions. In the wound-wait scheme, higher priority transactions never wait for lower priority transactions. In either case no deadlock cycle can develop.

A subtle point is that we must also ensure that no transaction is perennially aborted because it never has a sufficiently high priority. (Note that in both schemes, the higher priority transaction is never aborted.) When a transaction is aborted and restarted, it should be given the same timestamp that it had originally. Reissuing timestamps in this way ensures that each transaction will eventually become the oldest transaction, and thus the one with the highest priority, and will get all the locks that it requires.

The wait-die scheme is non-preemptive; only a transaction requesting a lock can be aborted. As a transaction grows older (and its priority increases), it tends to wait for more and more young transactions. A younger transaction that conflicts with an older transaction may be repeatedly aborted (a disadvantage with respect to wound wait), but on the other hand, a transaction that has all the locks it needs will never be aborted for deadlock reasons (an advantage with respect to wound-wait, which is preemptive).

Deadlock Detection

Deadlocks tend to be rare and typically involve very few transactions. This observation suggests that rather than taking measures to prevent deadlocks, it may be better to detect and resolve deadlocks as they arise. In the detection approach, the DBMS must periodically check for deadlocks. When a transaction Ti is suspended because a lock that it requests cannot be granted, it must wait until all transactions Tj that currently hold conflicting locks release them.

The lock manager maintains a structure called a waits-for graph to detect deadlock cycles. The nodes correspond to active transactions, and there is an arc from Ti to Tj if (and only if) Ti is waiting for Tj to release a lock. The lock manager adds edges to this graph when it queues lock requests and removes edges when it grants lock requests.

Detection versus Prevention

In prevention-based schemes, the abort mechanism is used preemptively in order to avoid deadlocks. On the other hand, in detection-based schemes, the transactions in a deadlock cycle hold locks that prevent other transactions from making progress. System throughput is reduced because many transactions may be blocked, waiting to obtain locks currently held by deadlocked transactions.

This is the fundamental trade-off between these prevention and detection approaches to deadlocks: loss of work due to preemptive aborts versus loss of work due to

blocked transactions in a deadlock cycle. We can increase the frequency with which we check for deadlock cycles, and thereby reduce the amount of work lost due to blocked transactions, but this entails a corresponding increase in the cost of the deadlock detection mechanism.

A variant of 2PL called Conservative 2PL can also prevent deadlocks. Under Conservative 2PL, a transaction obtains all the locks that it will ever need when it begins, or blocks waiting for these locks to become available. This scheme ensures that there will not be any deadlocks, and, perhaps more importantly, that a transaction that already holds some locks will not block waiting for other locks. The trade-o_ is that a transaction acquires locks earlier. If lock contention is low, locks are held longer under Conservative 2PL. If lock contention is heavy, on the other hand, Conservative 2PL can reduce the time that locks are held on average, because transactions that hold locks are never blocked.

Lecture No. 43

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Chapter 12
"Database Management Systems" Second edition written by Raghu Ramakrishnan, Johannes Gehkre.	Chapter 18

Overview of Lecture

- Incremental log with deferred updates
- Incremental log with immediate updates
- Introduction to concurrency control

Incremental Log with Deferred Updates

We are discussing the deferred updates approach regarding the database recovery techniques. In the previous lecture we studied the structure of log file entries for the deferred updates approach. In today's lecture we will discuss the recovery process.

Write Sequence:
First we see what the sequence of actions is when a write operation is performed. On encountering a 'write' operation, the DBMS places an entry in the log file buffer mentioning the effect of the write operation. For example, if the transaction includes the operations:

…….
X = X + 10
Write X
…….

Supposing that the value of X before the addition operation is 23 and after the execution of operation it becomes 33. Now against the write operation, the entry made in the log file will be

<Tn, X, 33>

In the entry, Tn reflects the identity of the transaction, X is the object being updated and 33 is the value that has to be placed in X. Important point to be noted here is that the log entry is made only for the write operation. The assignment operation and any other mathematical or relational operation is executed in RAM. For a 'read' operation the value may be read from the disk (database) or it may already be in RAM or cache. But the log entry results only from a 'write' operation.

The transaction proceeds in a similar fashion and entries keep on adding for the write operations. When the 'commit' statement is executed then, first, the database buffer is updated, that is the new value is placed in the database buffer. After that the log file is moved to disk log file which means that the log file entries have been made permanent. Then write is performed, that is, the data from DB buffer is moved to disk. There may be some delay between the execution of the commit statement and the actual writing of data from database buffer to disk. Now, during this time if system crashes then the problem is that the transaction has been declared as committed whereas the final result of the object was still in RAM (in database buffer) and due to system crash the contents of the RAM have been lost. The crash recovery procedure takes care of this situation.

The Recovery Procedure

When crash occurs, the recovery manager (RM), on restart, examines the log file from the disk. The transactions for which the RM finds both begin and commit operations, are redone, that is, the 'write' entries for such transactions are executed again. For example, consider the example entries in a log file

Log File Entries
<T1, begin >
<T1, X, 33>
<T2, begin>
<T1, Y, 9>
<T2, A, 43>
<T1, X, 18>
<T1, commit >
<T2, abort>
<T3, begin>
<T3, B, 12>

Here we find the begin entries for T1, T2 and T3, however, the commit entry is only for T1. On restart after the crash, the RM performs the write operations of T1 again in the forward order, that is, it writes the value 33 for object X, then writes 9 for Y and writes 19 for X again. If some or all of these operations have been performed already, writing them again will not cause any harm.

The RM does not any action for the transactions for which there is only a begin entry or for which there are begin and abort entries, like T3 and T2 respectively.
A question arises here, that how far should the RM go backward in the log file to trace the transactions affected from the crash that the RM needs to Redo or ignore. The log file may contain so many entries and going too much back or starting right from the start of the log file will be inefficient. For this purpose another concept of 'checkpoint' is used in the recovery procedure.

Checkpoint:
Checkpoint is also a record or an entry in the log file. It serves as a milestone or reference point in the log file. At certain point in time the DBMS enters a log entry/record in the log file and also performs certain operations listed below:

- **It moves modified database buffers to disk**
- **All log records from buffer to disk**
- **Writing a checkpoint record to log; log record mentions all active transactions at the time**

Now in case of crash, the RM monitors the log file up to the last checkpoint. The checkpoint guarantees that any prior commit has been reflected in the database. The RM has to decide which transactions to Redo and which to ignore and RM decides it on the following bases:

- **Transactions ended before the checkpoint, are ignored altogether.**
- **Transactions which have both begin and the commit entries, are Redone. It does not matter if they are committed again.**
- **Transactions that have begin and an abort entry are ignored.**
- **Transactions that have a begin and no end entry (commit or rollback) are ignored**

This way checkpoint makes the recovery procedure efficient. We can summarize the deferred approach as:

- **Writes are deferred until commit for transaction is found**
- **For recovery purpose, log file is maintained**
- **Log file helps to redo the actions that may be lost due to system crash**
- **Log file also contains checkpoint records**

Next, we are going to discuss the other approach of crash recovery and that is incremental log with immediate updates.

Incremental Log with Immediate Updates

Contrary to the deferred updates, the database buffers are updated immediately as the 'write' statement is executed and files are updated when convenient. The log file and database buffers are maintained as in the deferred update approach. One effect of immediate update is that the object updating process needs not to wait for the commit statement; however, there is a problem that the transaction in which a 'write' statement has been executed (and the buffer has been updated accordingly) may be aborted later. Now there needs to be some process that cancels the updation that has been performed for the aborted transaction. This is achieved by a slightly different log file entry/record. The structure of log file entry for immediate update technique is <Tr, object, old_value, new_value>, where Tr represents the transaction Id, 'object' is the object to be updated, old_value represents the value of object before the execution of 'write' statement and new_value is the new value of object. Other entries of the log file in immediate update approach are just like those of deferred update.
The sequence of steps on write statement is:

- **A log record of the form <T, X, old val, new val> is written in the log file**
- **Database buffers are updated**
- **Log record is moved from buffer to the file**
- **Database is updated when convenient**
- **On commit a log file entry is made in the log file as <T, commit>**

Recovery Sequence

In case of a crash, the RM detects the crash as before on restart and after that the recovery procedure is initiated. The transactions for which the <Tr, begin> and <Tr, commit> are found, those transactions are redone. The redo process is performed by copying the new value of the objects in the forward order. If these transactions have already been successfully executed even then this redo will not do any harm and the final state of the objects will be the same. The transactions for which a <Tr, begin> and <Tr, abort> are found or those transaction for which only <Ts, begin> is found, such transactions are undone. The undo activity is performed by executing the write statements in the reverse order till the start of the transaction. During this execution, the old value of the objects are copied/placed so all the changes made by this transaction are cancelled.

Log File Entries
<T1, begin>
<T1, X, 25, 33>
<T2, begin>
<T1, Y, 5, 9>
<T2, A, 80, 43>
<T1, X, 33, 18>
<T1, commit>
<T3 begin >
<T2, A, 43, 10>
<T3, B, 12, 9>
<T2, abort>

Here we find the begin entries for T1, T2 and T3, however, the commit entry is only for T1 and also there is an abort entry for T2. On restart after the crash, the RM performs the write operations of T1 again in the forward order, that is, it writes the value 33 for object X, then writes 9 for Y and writes 18 for X again. If some or all of these operations have been performed already, writing them again will not cause any harm.

To undo the effects of T2 (which has been aborted) the write statements of T2 are executed in the reverse order copying the old value from the entry/record. Like, here the second write statement of T2 that had possibly written the value 10 over 43, now 43 will be placed for A. Moving backward we find another write statement of T2 that places 43 replacing 80. Now we pick the old value, i.e., 80 and place this value for A. After performing these operations the effects of executing an aborted T2 are completely eliminated, that is, the object A contains value 80 that it had before the execution of transaction T2. Same process is also applied for the transaction T3. Checkpoints are used in immediate updates approach as well.

We have discussed recovery procedure both in deferred update and immediate update. Crash recovery is an important aspect of a DBMS, since in spite of all precautions and protections the crashes happen. We can minimize the frequency but they cannot be avoided altogether. Once crash happens, the purpose of recovery mechanism is to

reduce the effect of this crash as much as possible and to bring database in a consistent state and we have studied how the DBMS performs this activity.

Concurrency Control

Concurrency control (CC) is the second part of the transaction management. The objective of the CC is to control the concurrent access of database by multiple users at the same time called the concurrent access.

First questions is that why to have the concurrent access. The answer to this question refers to very basic objective of the database approach that is sharing of data. Now if we are allowing multiple users to share the data stored in the database, then how will they access this data. One user at a time? If so, it will be very inefficient, we cannot expect the users to be that patient to wait so long. At the same time we have so powerful and sophisticated computer hardware and software that it will be a very bad under-utilization of the resources if allow one user access at one time. That is why concurrent access of the data is very much required and essential.

We are convinced that concurrent access is required, but if not controlled properly the concurrent access may cause certain problems and those problems may turn the database into an inconsistent state and this is never every allowed in the database processing. The objective of CC is to allow the concurrent access in such a way that these problems due to concurrent access are not encountered or to maintain the consistency of the database in the concurrent access. It is important to understand that CC only takes care of the inconsistencies that may possibly be encountered due to concurrent access. Inconsistency due to some other reasons is not the concern of CC.

Problems due to Concurrent Access

If not controlled properly, the concurrent access may introduce following three problems in database:

- **Lost Update Problem**
- **Uncommitted Update Problem**
- **Inconsistent Analysis Problem**

We discuss these problems one by one

Lost Update Problem

This problem occurs when multiple users want to update same object at the same time. This phenomenon is shown in the table below:

TIME	TA	TB	BAL
t_1	Read (BAL)		1000
t_2	…….	Read (BAL)	1000
t_3	BAL = BAL - 50	……..	1000
t_4	Write (BAL)		950
t_5	…….	BAL = BAL + 10	950
t_6	……..	Write (BAL)	1010

Table 1: Lost update problem

This first column represents the time that acts as reference for the execution of the statements. As is shown in the table, at time t1 transaction TA reads the value of object 'BAL' that supposedly be 1000. At time t2, transaction TB reads the value of same object 'BAL' and also finds the value as 1000. Now at time t3, TA subtracts 50 of 'BAL and writes it at time t4. On the other hand TB adds 10 into value of 'BAL' (that she has already read as 1000) and writes the value of 'BAL'. Now since TB wrote the value after TA, the update made by TA is lost, it has been overwritten by value of TB. This is the situation that reflects the lost update problem.

This problem occurred because concurrent access to the same object was not controlled properly, that is, concurrency control did not manage the things properly. There are two more situations that reflect the problems of concurrent access that we will discuss in the next lecture.

Summary

This lecture concluded the discussion on crash recovery where we studied that the main tool used for recovery is the log file. The structure of records stored in the different types of log files is almost the same, except for the record that stores the record of the change made in the database as a result of a write statement. The deferred update approach stores only the new value of the object in the log file. Whereas the immediate update approach stores the previous as well as new value of the object being updated. After the crash recovery we started the discussion on CC,

where we were studying different problems of CC; first of them is the lost update problem that we have discussed in today's lecture, rest two will be discussed in the next lecture.

Lecture No. 44

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.	Chapter 12
"Database Management Systems" Second edition written by Raghu Ramakrishnan, Johannes Gehkre.	Chapter 18

Overview of Lecture

- Concurrency control problems
- Serial and interleaved schedules
- Serializability theory
- Introduction to locking

We are studying the concurrency control (CC) mechanism. In the previous lecture we discussed that concurrent access means the simultaneous access of data from database by multiple users. The concurrency control (CC) concerns maintaining the consistency of the database during concurrent access. We also studied that if concurrent access is not controlled properly then database may encounter three different problems which may turn database into an inconsistence state. One of these problems we discussed in the previous lecture, now we will discuss the remaining two.

Uncommitted Update Problem

It reflects a situation when a transaction updates an object and another transaction reads this updated value but later the first transaction is aborted. The problem in this situation is that the second transaction reads the value updated by an aborted transaction. This situation is shown in the table below:

TIME	TA	TB	BAL
t_1	Read (BAL)	…….	1000
t_2	BAL=BAL+100	……..	1000
t_3	Write (BAL)	……..	1100
t_4	…….	Read (BAL)	1100
t_5	…….	BAL=BAL*1.5	1100
t_6	…….	Write (BAL)	2650
t_7	ROLLBACK	……..	?

Table 1: Uncommitted update problem

As is shown in the table, at time t2, transaction TA updates the value of object 'BAL' by adding 100 in its initial value 1000 and at time t3 it writes this updated value to database. So 'BAL' is written as 1100. Now at time t4, TB reads the value of 'BAL' and it gets 1100 then TB updates the value multiplying it by 1.5 and writes it at time t6. So the new value written for 'BAL' is 2650. However at time t7 transaction TA is rolled back as a result of this rollback the changes made by TA stand cancelled, but TB has already performed certain processing on the value set by TA. This is an inconsistent state of the database.

Inconsistent Analysis

This problem of concurrent access occurs when one transaction operates on many records, meanwhile another modifies some of them effecting result of first. This problem is expressed in the following table:

TIME	TA	TB
t_1	Read (BAL_A) (5000)	…….
t_2	INT = BAL_A * .05	……..
t_3	TOT = TOT + INT	……..
t_4	…….	…….
t_5	…….	Read (BAL_A)
t_6	…….	BAL_A=BAL_A – 1000
t_7	…….	Write (BAL_A)
t_8	…….	Read (BAL_E) (5000)
t_9	…….	BAL_E = BAL_E + 1000
t_{10}	…….	Write (BAL_E)
t_{11}	Read (BAL_E) (6000)	……..
t_{12}	INT = BAS_E * .05	……..
t_{13}	TOT = TOT + INT	……..

Table 2: Inconsistent analysis problem

Suppose Tr-A computes interest on all accounts balances. For this, TA reads the balance of each account multiplies it with 0.05 and adds this interest amount into a variable TOT. Now by time t5, TA has read the balance of account ‘A’ (BAL_A) and computed interest on it, it has to perform same process on all the accounts. In the meanwhile from time t5 to t10 another transaction TB subtracts a value from balance of account ‘A’ and adds this value to the balance of account ‘E’. When transaction TA reaches the account ‘E’ to compute interest on it, the value added in it by TB is also considered which as a matter of fact is being considered second time. First time from account ‘A’ and now from account ‘E’. The analysis obtained through transaction TA will be wrong at the end.

We have discussed three problems of concurrent access; the CC mechanism has to make sure that these problems do not occur in the database. In the following, we will discuss how it is done. We will start by studying some basic concepts.

Serial Execution

Serial execution is an execution where transactions are executed in a sequential order, that is, one after another. A transaction may consist of many operations. Serial execution means that all the operations of one transaction are executer first, followed by all the operations of the next transaction and like that. A Schedule or History is a list of operations from one or more transactions. A schedule represents the order of execution of operations. The order of operations in a schedule should be the same as in the transaction. Schedule for a serial execution is called a serial schedule, so in a serial schedule all operations of one transactions are listed followed by all the

operations of another transactions and so on. With a given set of transactions, we can have different serial schedules. For example, if we have two transactions, then we can have two different serial schedules as is explained in the table below:

Serial Sched 1 TA, TB	BAL	Serial Sched 2 TB, TA	BAL
Read $(BAL)_A$	50	Read $(BAL)_B$	50
$(BAL = BAL - 10)_A$	50	$(BAL=BAL * 2)_B$	50
Write $(BAL)_A$	40	Write $(BAL)_B$	100
Read $(BAL)_B$	40	Read $(BAL)_A$	100
$(BAL=BAL * 2)_B$	80	$(BAL = BAL - 10)_A$	100
Write $(BAL)_B$	80	Write $(BAL)_A$	90

Table 3: Two different serial schedules, TA, TB and TB, TA

The table shows two different schedules of two transactions TA and TB. The subscript with each operation shows the transaction to which the operation belongs. For example, Read $(BAL)_A$ means the read operation of TA to read the object 'BAL'. By looking at the table that in each serial schedule; all the operations of one transaction are executed first followed by those of the other transaction. An important point to be noted here is that different serial schedules of the same transactions may result in different final state of the database. As we can see in the table 3, final value of the object 'BAL' is 80 with the serial schedule TA, TB and it is 90 with serial schedule TB, TA. However, it is guaranteed that a serial schedule always leaves the database in a consistent state. In other words, the three problems of concurrency that we studies earlier will not occur if a serial schedule is followed.

The serial schedule ensures the consistency of the database, so should we prefer serial schedule? The answer is no. Because serial execution is badly under utilization of resources and users will not like it at all since they will have to wait a lot for the transactions' execution. Suppose we are following a serial execution and a transaction Tn has to executed completely before any other transaction may start. Lets say Tn needs an input from the user who initiated it, and by chance users gets stuck somewhere or is thinking, unless and until the users does not enter a value, transaction Tn cannot proceed and all other transactions are waiting for their turn to execute. If

allowed to happen, this will be a much disliked situation. Therefore serial schedules are not preferred for execution.

Contrary to serial schedule is an interleaved schedule in which transactions' execution is interleaved, that is, operations of different transactions are intermix with each other. However, the internal order of the operations is not changed during this intermixing. Interleave schedule provides more efficient execution since control is switched among transaction, so on one side it makes a better use of the resources and on the other hand if one transaction is stuck or halted due to some reasons, the processing can go on with other transactions. Following figure presents serial and interleaved schedules among two transactions TA and TB

TA = {Read(X), Write(X), commit}
TB = {Read(Y), Read(X), Write(Y), commit}

$S_1 = \{R_A(X), W_A(X), C_A, R_B(Y), R_B(X), W_B(Y), C_B\}$

$S_2 = \{R_B(Y), R_B(X), W_B(Y), C_B, R_A(X), W_A(X), C_A\}$

$S_3 = \{R_B(Y), R_B(X), R_A(X), W_B(Y), C_B, W_A(X), C_A\}$

$S_4 = \{R_A(X), R_B(Y), R_B(X), W_A(X), C_A, W_B(Y), C_B\}$

Fig. 1: Two transactions and different schedules

In figure 1, we have two transactions TA and TB consisting of different operations. S1, S2, S3 and S4 are four different schedules of these transactions. S1 and S2 are two serial schedules where as S3 and S4 are two interleaved schedules. Obviously we can have just two serial schedules of two transactions, but we can have many other interleaved schedules of these transactions.

Interleaved schedules are preferred but they may generate three of the concurrent access problems if not controlled properly. Before discussing the solution to this problem, lets see what is the actual problem in concurrent access.

There are different situations during concurrent access of the data:

- Different transactions accessing (reading or writing) different objects
- Different transactions reading same object

- Different transactions accessing same object and one or all writing it

The first two of these situations do not create any problem during concurrent access, so we do not need to worry in these situations. However, third situation is precisely the one that creates the concurrency problems and we have to control this situation properly. Third situation introduces the concept of conflicting operations; the operations that are accessing the same object and one of them writing the object. The transactions that contain conflicting operations are called conflicting transactions. Basically, in concurrency control we have to take care of the conflicting transactions and for that we have to study the concept of serializability.

Serializability

An interleaved schedule is said to be serializable if its final state is equivalent to some serial schedule. Serializable schedule can also be defined as a schedule in which conflicting operations are in a serial order. The serializable schedule ensures the consistency of the database. However, this is worthwhile to mention here again that serializability takes care of the inconsistencies only due to the interleaving of transactions. The serializability concerns generating serializable schedules or we can say that the purpose of the CC mechanism is to ensure serializability of the schedules. Generating serializable schedule involves taking care of only conflicting operations. We have to adopt a particular serial schedule among the conflicting operations; the non-conflicting operations can be interleaved to any order, it will not create any problem at all. There are two major approaches to implement the serializability; Locking and Timestamping. We will discuss both in detail.

Locking

The basic idea of locking is that an object is locked prior to performing any operation. When the operation has been performed the lock is released. Locking is maintained by the transaction manager or more precisely lock manager. Transactions apply locks on objects. During the time when an object is locked it is available for operations only to the transaction that has got the lock on that object through lock manager. When an item is locked and during that time another transaction needs to perform some

operation on that object, the transaction applies for the lock on the object but since the object is already locked, this transaction will have to wait. So it enters into a wait state. When first transaction finishes its job, it releases locks on items then the second item gets the lock on the object and then it can proceed.

Transactions perform two types of operations on objects; read or write. That is, a transaction either performs a read operation on an object or it writes it. Accordingly there are two types of locks as well. A read or shared lock and a write or exclusive lock a transaction applies lock according to the nature of operation that it wants to perform on a particular object. If a transaction TB applies a lock on an object O, the lock manager first checks if O is free, it is free then TB gets the desired lock on O. However, if O is already locked by another transaction say TA then lock manager checks the compatibility of the locks; the one that has already been assigned and the one that has been applied now. The compatibility of locks means that if the two locks from two different transactions may exist at the same time. The compatibility of locks is checked according to the following table:

Transaction A

Transaction B		Read	Write
	Read	*Yes*	*No*
	Write	*No*	*No*

Table shows that if a transaction A has got a read lock on an object and transaction B applies for the read lock, B will be granted this lock. It means two read locks are compatible with each other, that is, they can exist at the same time. Therefore two or even more than two transactions can read an object at the same time. Other cells contain 'No'; it means these locks are not compatible. So if a transaction has got a 'write' lock on an object and another transaction applies for a 'read' or 'write' lock, the lock will not be granted and the transaction applying for the lock later, will have to wait.

That is all for today's lecture. The locking mechanism will be discussed in detail in the next lecture.

Summary

In this lecture we discussed two of the three CC problems. Then we discussed that there are different types of schedules that determine the sequence of execution of

operations form different transactions. A serial schedule maintains the consistency of the database for sure, but serial schedule is not much useful. Interleaved schedule is preferred but if not controlled properly an interleaved schedule may cause consistency problems. So CC is based on the serializability theory that controls the order of conflicting operations. The serializability is applied using locking or Timestamping. Locking is based on two types of locks; shared and exclusive. Only two shared locks are compatible with each other, none of the remaining combinations are compatible.

Lecture No. 45

Reading Material

"Database Systems Principles, Design and Implementation" written by Catherine Ricardo, Maxwell Macmillan.

Overview of Lecture:

- Deadlock Handling
- Two Phase Locking
- Levels of Locking
- Timestamping

This is the last lecture of our course; we had started with transaction management. We were discussing the problems in transaction in which we discussed the locking mechanism. A transaction may be thought of as an interaction with the system, resulting in a change to the system state. While the interaction is in the process of changing system state, any number of events can interrupt the interaction, leaving the state change incomplete and the system state in an inconsistent, undesirable form. Any change to system state within a transaction boundary, therefore, has to ensure that the change leaves the system in a stable and consistent state.

Locking Idea

Traditionally, transaction isolation levels are achieved by taking locks on the data that they access until the transaction completes. There are two primary modes for taking locks: optimistic and pessimistic. These two modes are necessitated by the fact that when a transaction accesses data, its intention to change (or not change) the data may not be readily apparent.

Some systems take a pessimistic approach and lock the data so that other transactions may read but not update the data accessed by the first transaction until the first transaction completes. Pessimistic locking guarantees that the first transaction can always apply a change to the data it first accessed.

In an optimistic locking mode, the first transaction accesses data but does not take a lock on it. A second transaction may change the data while the first transaction is in progress. If the first transaction later decides to change the data it accessed, it has to detect the fact that the data is now changed and inform the initiator of the fact. In optimistic locking, therefore, the fact that a transaction accessed data first does not guarantee that it can, at a later stage, update it.

At the most fundamental level, locks can be classified into (in increasingly restrictive order) shared, update, and exclusive locks. A shared lock signifies that another transaction can take an update or another shared lock on the same piece of data. Shared locks are used when data is read (usually in pessimistic locking mode).

An update lock ensures that another transaction can take only a shared lock on the same data. Update locks are held by transactions that intend to change data (not just read it). If a transaction locks a piece of data with an exclusive lock, no other transaction may take a lock on the data. For example, a transaction with an isolation level of read uncommitted does not result in any locks on the data read by the transaction, and a transaction with repeatable read isolation can take only a share lock on data it has read.

DeadLock

A deadlock occurs when the first transaction has locks on the resources that the second transaction wants to modify, and the second transaction has locks on the resources that the first transaction intends to modify. So a deadlock is much like an infinite loop: If you let it go, it will continue until the end of time, until your server crashes, or until the power goes out (whichever comes first). Deadlock is a situation when two transactions are waiting for each other to release a lock. The transaction involved in deadlock keeps on waiting unless deadlock is over.

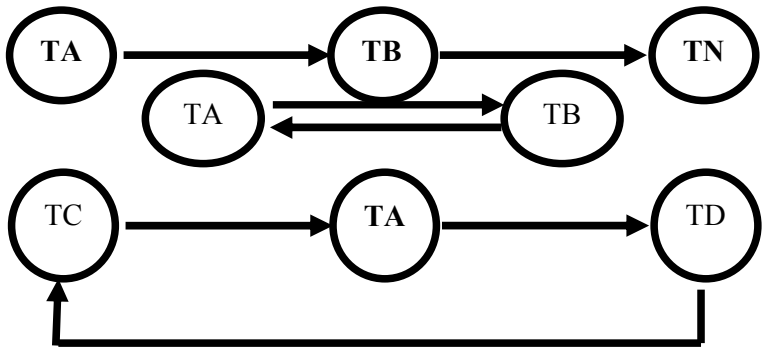
DeadLock Handling

Following are some of the approaches for the deadlock handling:

- Deadlock prevention
- Deadlock detection and resolution
- Prevention is not always possible
- Deadlock is detected by wait-for graph

Wait – for Graph:

It is used for the detection of deadlock. It consists of nodes and links. The nodes represent transaction, whereas arrowhead represents that a transaction has locked a particular data item. We will see it following example:



Now in this figure transaction A is waiting for transaction B and B is waiting for N. So it will move inversely for releasing of lock and transaction A will be the last one to execute. In the second figure there is a cycle, which represents deadlock, this kind of

cycle can be in between two or more transactions as well. The DBMS keeps on checking for the cycle.

Two Phase Locking

This approach locks data and assumes that a transaction is divided into a growing phase, in which locks are only acquired, and a shrinking phase, in which locks are only released. A transaction that tries to lock data that has been locked is forced to wait and may deadlock. During the first phase, the transaction only acquires locks; during the second phase, the transaction only releases locks. More formally, once a transaction releases a lock, it may not acquire any additional locks. Practically, this translates into a system in which locks are acquired as they are needed throughout a transaction and retained until the transaction ends, either by committing or aborting.
For applications, the implications of 2PL are that long-running transactions will hold locks for a long time. When designing applications, lock contention should be considered. In order to reduce the probability of deadlock and achieve the best level of concurrency possible, the following guidelines are helpful.

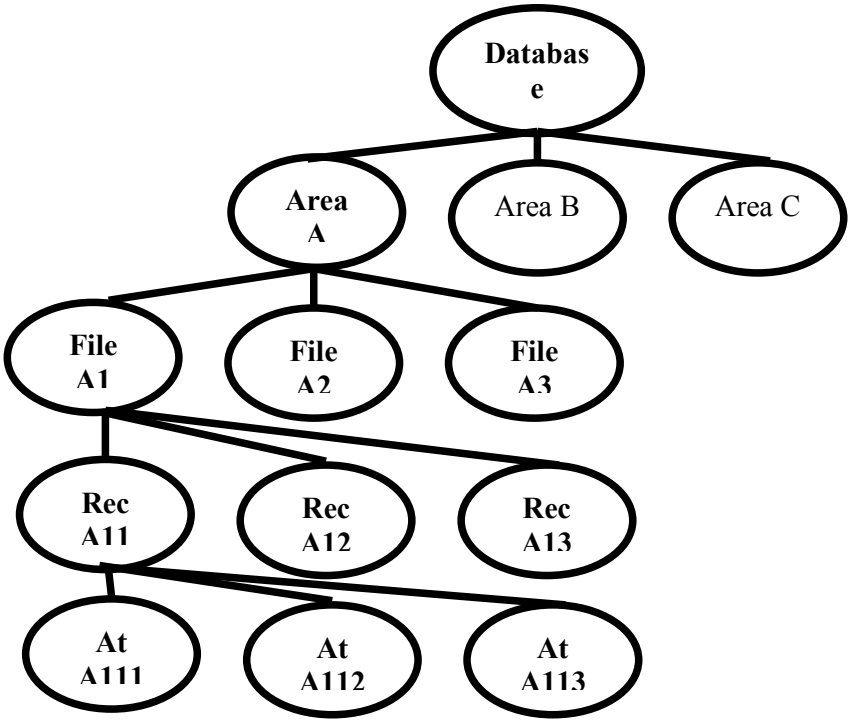
- When accessing multiple databases, design all transactions so that they access the files in the same order.
- If possible, access your most hotly contested resources last (so that their locks are held for the shortest time possible).
- If possible, use nested transactions to protect the parts of your transaction most likely to deadlock.

© Virtual University of Pakistan
thegradup.com

Levels of Locking

- Lock can be applied on attribute, record, file, group of files or even entire database
- Granularity of locks.
- Finer the granularity more concurrency but more overhead.

Greater lock granularity gives you finer sharing and concurrency but higher overhead; four separate degrees of consistency which give increasingly greater protection from inconsistencies are presented, requiring increasingly greater lock granularity.



When a lock at lower level is applied, compatibility is checked upward. It means intimation would be available in the hierarchy till the database. The granularity of locks in a database refers to how much of the data is locked at one time. A database server can lock as much as the entire database or as little as one column of data. Such extremes affect the concurrency (number of users that can access the data) and locking overhead (amount of work to process lock requests) in the server. By locking at higher levels of granularity, the amount of work required to obtain and manage locks is reduced. If a query needs to read or update many rows in a table:

- It can acquire just one table-level lock
- It can acquire a lock for each page that contained one of the required rows
- It can acquire a lock on each row

Less overall work is required to use a table-level lock, but large-scale locks can degrade performance, by making other users wait until locks are released. Decreasing the lock size makes more of the data accessible to other users. However, finer granularity locks can also degrade performance, since more work is necessary to maintain and coordinate the increased number of locks. To achieve optimum performance, a locking scheme must balance the needs of concurrency and overhead.

Deadlock Resolution

If a set of transactions is considered to be deadlocked:

- choose a victim (e.g. the shortest-lived transaction)
- Rollback `victim' transaction and restart it.
- The rollback terminates the transaction, undoing all its updates and releasing all of its locks.
- A message is passed to the victim and depending on the system the transaction may or may not be started again automatically.

Timestamping

Two-phase locking is not the only approach to enforcing database consistency. Another method used in some DMBS is timestamping. With timestamping, there are no locks to prevent transactions seeing uncommitted changes, and all physical updates are deferred to commit time.

- Locking synchronizes the interleaved execution of a set of transactions in such a way that it is equivalent to some serial execution of those transactions.
- Timestamping synchronizes that interleaved execution in such a way that it is equivalent to a particular serial order - the order of the timestamps.

Problems of Timestamping

- When a transaction wants to read an item that has been updated by a younger transaction.
- A transaction wants to write an item that has been read or written by a younger transaction.

Timestamping rules

The following rules are checked when transaction T attempts to change a data item. If the rule indicates ABORT, then transaction T is rolled back and aborted (and perhaps restarted).

- If T attempts to read a data item which has already been written to by a younger transaction then ABORT T.
- If T attempts to write a data item which has been seen or written to by a younger transaction then ABORT T.

If transaction T aborts, then all other transactions which have seen a data item written to by T must also abort. In addition, other aborting transactions can cause further aborts on other transactions. This is a `cascading rollback'.

Summary

In today's lecture we have read the locking mechanism and prevention of deadlocks. With this we are finished with our course. Locking is a natural part of any application. However, if the design of the applications and transactions is not done correctly, you can run into severe blocking issues that can manifest themselves in severe performance and scalability issues by resulting in contention on resources. Controlling blocking in an application is a matter of the right application design, the correct transaction architecture, a correct set of parameter settings, and testing your application under heavy load with volume data to make sure that the application scales well.