Database Modeling and Design
(CS312)

Contents

Module-01: Introduction to Data & Information

A. Definition of Data

Data can be referred to raw material from which we can draw conclusion after analysis or the facts from which we can infer new facts. Data has nothing to do with decision making but the data lays down foundation for decision making. Data is collected and further analyzed to convert it into information suitable for decision making. Data and information are two separate things and are not to be confused with each other.

B. Example of Data

The followings are the examples of data:

- *Telephone Directory*: it's a huge collection of Data, with an internal structure of representation i.e. name, telephone number, city or address. Facts with structures are also termed as data.
- *Student Record on Computer*: a file on computer containing data of ten thousand students is also a data. This data too have internal structure in the form of Student ID, Student Name, CGPA and likes.

C. Definition of Information

Information can be defined as processed or organized data presented in a given context so as to make it useful related to the problem in hand. When a specific set of data is analyzed or interpreted, it becomes information that is more suitable for decision making. Simply stating, data becomes information when it becomes relevant to the decision problem.

D. Examples of Information

The followings are the examples of information:

- *Telephone Directory: as stated above, telephone directory is a huge set of data and when we process this set of data to get the telephone number of a specific dentist or a colleague, it becomes information.*
- *Student Record on Computer: a file on computer containing data of ten thousand students is again a huge set of data, converts in to information when processed to get list of a students with CGPA more than three.*

E. Comparison of Data & Information

Data and information are often considered to be a synonym of each other but the truth is that there is nothing in common between these two. Both have different meaning and characteristics. Information is a subset of data, typically required set of data. Data leads to information so information is considered to be dependent of data.

F. Example to elaborate difference between Data and Information

Consider the following example to understand the difference between Data and Information:

Consider a file storing flight routes from one city to another. Data is structured in the form of Flight ID, Origin, Destination, Date of Departure, Time of Departure etc. The file contains two thousand numbers of routes. This file qualifies to be called Data because it's a huge set of structured data.

This huge data will turn into information when processed or filtered to get information about flight for a specific destination, let's say from Lahore to Karachi on April 15th.

Sample Data				
Flight ID	Origin	Destination	Data of Flight	Time of Flight
PK-725	Islamabad	Lahore	7-Apr-15	8:00
PK-702	Lahore	Karachi	15-Apr-15	22:00
PK-702	Lahore	Karachi	15-Apr-15	10:00
SI-829	Karachi	Dubai	12-Apr-15	13:00
GI-390	Lahore	Istanbul	28-Apr-15	17:00
DATA				
Question: Flight Details from Lahore to Karachi on 15-April-15				
PK-702	Lahore	Karachi	15-Apr-15	22:00
PK-702	Lahore	Karachi	15-Apr-15	10:00
INFORMATION				

G. Life With& Without Database

Database is organized data with digital procedures, which is able to handle different set of data with a very large capacity. Database is a huge repository of data and it comes with ease to manage and therefore database has become a substitute for the manual handling of lots of file with very high chances and risk of data loss. Access to data has become easy and less time-consuming activity due to the databases.

H. Benefits of Using Databases

As mentioned above, advantages of using databases are enormous. Benefits of using databases vary from reducing redundancy in data to getting rid of unnecessary set of data. Without databases, loads of data is to be handled manually with many potential risks and difficult to access. With databases, it's easy to access large set of data when required. Example of Driving License and Passport can serve as a good example in which the profile data is same but it's repeated over again.

I. Why Database not Information Base

Databases deals with the storage or organization of data and the main focus of databases are to optimize data storage. On the other hand, information – as it is a subset of data - is all about the retrieval of specific data for different purposes. As information is dependent on data so databases are basic requirements for information which can then be used for multiple purposes.

J. Difference between Database and Database Management System (DBMS)

Database is all about design and structure of the data and DBMS works as a tool to manipulate or analyze the data inside the databases. DBMS is a whole system to manage a digital database, its storage, creation and retrieval (information). Differentiating between these two terms, Database is a huge set of data while DBMS is a tool to manage or control that data in a database with major function of data retrieval.

K. Example of Database and Database Management System (DBMS)

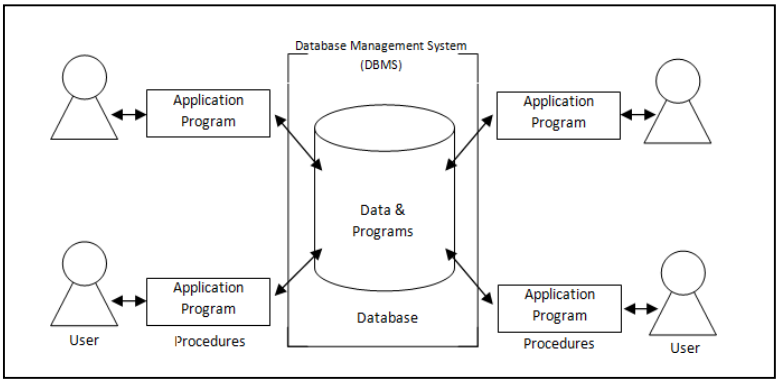
Example 01:

Database: Employee Records Column with Employee ID, Name, Qualification, Date of Joining etc.
DBMS: Oracle to create and load employee records

Example 02:
Database: Individual files on Hard Disk
DBMS: Operating System to read the file from Hard Disk

L. Components of Database Management System

Database Management System DBMS is the aggregate of the followings:



- Data inside the database which is to be processed, organized, retrieve or update.
- User: The users are the people who manage the databases and perform different operations on the databases in the database system. They can be end users, data base administrator or application programmer.
- Hardware: Hardware consists of a set of physical electronic devices such as computers (together with associated I/O devices like disk drives), storage devices, I/O channels, that make interface between computers and the real world systems etc.
- Software: It is the set of programs used to handle the database and to control and manage the overall computerized database/
- Procedures: Procedures refer to the instructions and rules that help to design the database and to use the DBMS.

Module 02: Data Storage Mechanism

A. Data Model Basics

Data model is the basic architectural unit of the databases which determines the structure of the database and determines the way in which data will be stored or organized. It also defines the connection, processing and storage of data in a system. It organize the data and determine how the data elements relate to each other The data model is the map or the design of the database and it is difficult to change when data is inserted.

M. Flat File Basics

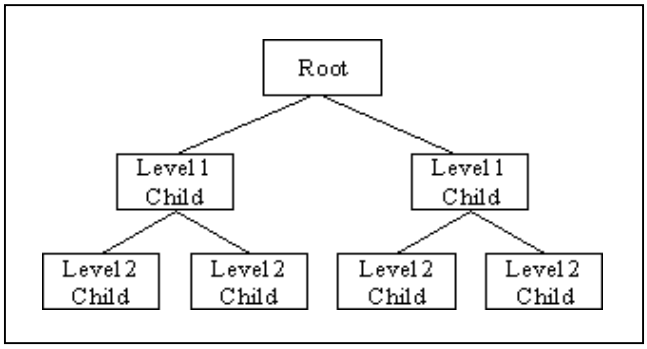
Flat File is a database which is stored on its host computer system as an ordinary file and the complete file has to be read into the memory to process or perform operations. This database contains one record per line and prefers plain text (word or notepad) or ASCII File. The flat file database design put all the data around a single table and data has no interrelationships.

N. How Flat File Works

Data is stored in columns and rows and each row is separated by a comma or tab. Each line of text holds one record. It contains many fields often with duplicate data.There is no relation between data and as more and more records are added to the database, it increases the chances of the data duplication. This redundancy results in reduced integrity of data and it becomes more unreliable. Along with this, the duplication makes it difficult to update the records as there are non-unique records in the database.

ID	Title	First name	Surname	Address	City	Postcode	Telephone
1	Mr	Tom	Smith	42 Mill Street	London	WE13GW	010344044
2	Mrs	Sandra	Jones	10 Low Lane	Hull	HU237HJ	022344033
2	Mr	John	Jones	10 Low Lane	Hull	HU237HJ	022344033

O. Hierarchical Database Basics



Hierarchical database is one of the two main data model proposed by IBM's IMS System. It's a data model in which data is stored in a tree like structure. The data is stored as records (combination of rows and columns) and these records are connected through links. The Hierarchical Data Model is a way of organizing a database with multiple one-to-many relationships. The structure is based on the rule that one parent can have many children but children are allowed only one parent. In order to retrieve data from a hierarchical database the whole tree needs to be traversed starting from the root node.

P. How Hierarchical Database Works

In hierarchical data model, data is retrieved using traversal mean visiting the root node and navigating through hits children. There is one-to-many relationship between root and its child node. Each parent record store the address of its child records and every child records will act as a parent record for its next level.

Q. Network Database Basics

Opposite to the hierarchical model, network model allows each record to have multiple parents and child records. This structure forms a generalized graph structure and this model can be seen as upside down tree. This model is considered to be a very flexible way or representing objects and their relationships with each other. Network database model is easy to navigate because of the linkages between the information and many-to-many relationships.

R. How Network Database Work

There is no concept of parent-child relationship in this type of data model. Each record will store the addresses of all the referencing records. As it support many-to-many relationships between root and child nodes, any record can be linked to any record and thus there is no hierarchy in this type of data model.

Module 03: Three-Tier Architecture

A. Introduction to 3-Tier Architecture

Tiers can be defined as the collection of logically related but physically independent components of an application. In this architecture, the application is divided into 3 tiers in which each tier has its own boundary. Each tier contributes towards the whole and several tiers join to make something whole. By segregating an application into various tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application.

S. Rationale for 3-Tier Architecture

3-Tier architecture came as a solution to the complexity of the application and enterprise scalability, the new architecture purpose three layers and different teams can work on the different layers at the same time thus making the development process easy. Dividing application into 3-Tiers helps avoiding single point of failure in the application and failure of one tier doesn't mean the crash of whole application.

T. External Level (Client Interface)

It is the top most layer of 3-Tier architecture and it is also known as "Presentation Layer". This tier managers the input/output data and their display. In other words it displays what clients view. This tier is responsible for formatting the input/output data into a presentable form. This tier is able to communicate one tier below with the application tier to display information. In simple words, it's a tier which users can access directly such as mobile applications, browsers and forms etc.

U. Conceptual Level (Business Logic)

This tier – also referred to as the application tier – is the middle tier that bridges the gap between the user interface and the underlying database. This tier consists of business and data rules. In this tier the logic of the business is encapsulated (Programming/Class Diagram/ERD). This tier receives the requests for the presentation tier only and interprets or processes the requests as per the defined rules and sends back the results to the presentation tier only. It can receive the request and send the response to the presentation tier only, and it can send request to only the below tier i.e. Internal Level (Data Tier)

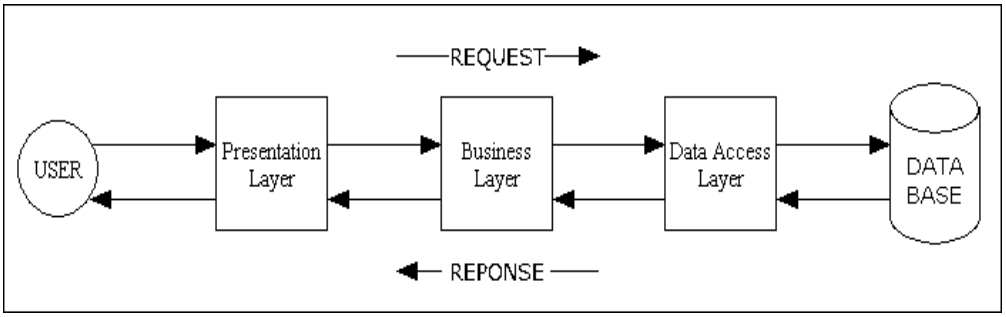
V. Internal Level (Data Tier)

This tier is responsible for storing information and data needed for the system and for optimizing the data access. Data needed by the application logic layer are retrieved from the database, and then the computation results produced by the application logic layer are stored back in the database. This tier can receive requests from Conceptual Level or logic tier only and can send the responses to the said tier only. All the decisions related to the data storage are stored here and these storage mechanisms are independent of other tiers.

W. Layer Independence

The tiers mentioned above have no interdependence and work exclusively. Every tier is totally unaware of the internal working or the logics of other tiers and the decision made by the tiers are also independent. In simple words:

- Data layer is not aware of the internal working of logic layer
- Logic layer is not concerned about the workings of data layer
- Presentation layer works in total independence for other layers

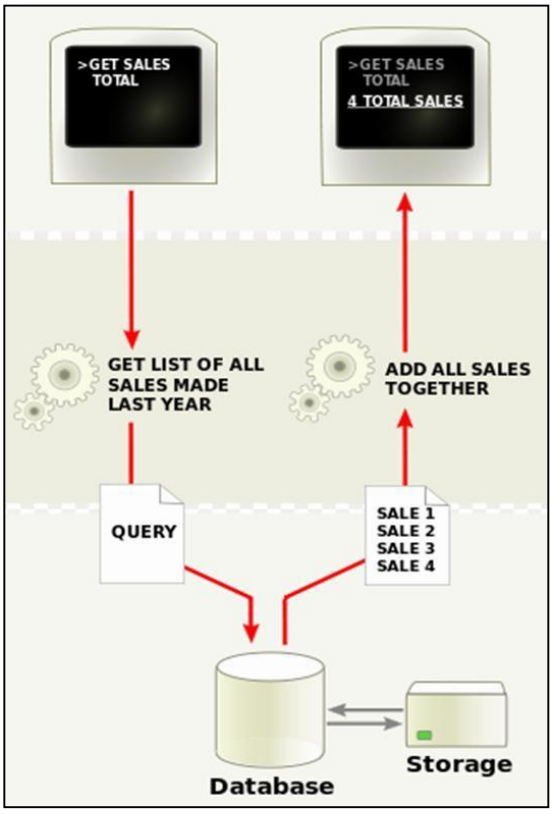


X. Example – Presentation Layer Independence

Consider the following situation:

Query: Get Total Sales

A manager of a company need to run sales query to find out the list of all sales made last year. He will use the presentation tier or the client interface to send this request to the application tier. Suppose this is done by using a form or any other way. This form, browser or mobile app through which the request is being made cannot affect the way the data is retrieved. All other tiers will work in total independence. Same way, all other tiers can't affect the way data is presented in the presentation tier.



Y. Example – Logical Layer Independence

Continuing with the same example:

Query: Get List of all Sales made last year

The structure or the workings of this tier is also exclusive to the logical tier. Using if-then, while or any other structure (i.e. how the query is made) is internal to this tier only and it can't affect the way data is displayed.

Z. Example – Data Layer Independence

As shown in the figure 6 above, the application tier will send query to the data tier and it will respond to the application tier only with the requested results. What logic the application has used or it will use is not the concern of the data tier; it will just provide the required results. The application tier will perform the activity as it is programmed to do and will forward the results to the presentation layer which will display the results to the end user. None of the mentioned tiers can affect the working or processes of the other tiers.

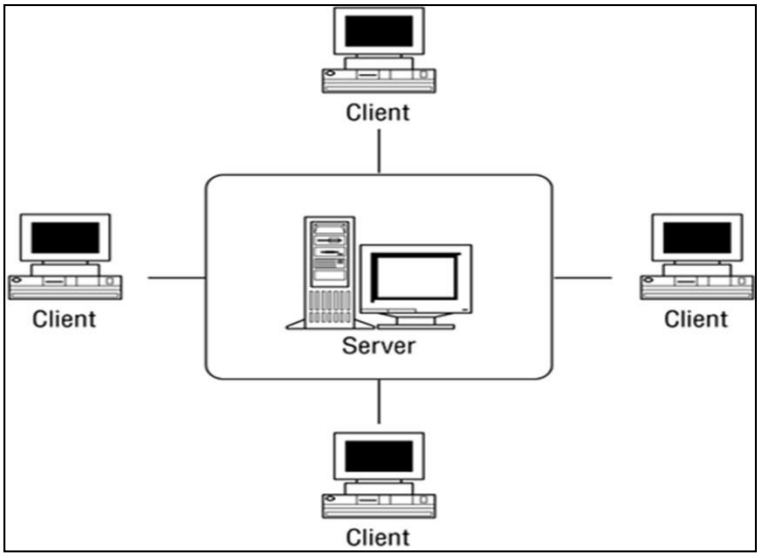
AA. Roles in Database Management System (DBMS)

There are following roles in a database environment. The details of them are also given.

- ***Application Programmer*:** application programmer develops the application that provide required the functionality by the end user.
- ***End User:*** The end-users are the 'clients' for the database, which has been designed and implemented, and is being maintained to serve their information needs.
- ***Database Administrator:*** Database Administrator (DBA) is responsible for the physical realization of the database, including physical database design and implementation, security and integrity control, maintenance of the operational system, and ensuring satisfactory performance of the applications for users.
- ***System Analyst:*** systems analyst is a person who uses analysis and design techniques to solve business problems using information technology.

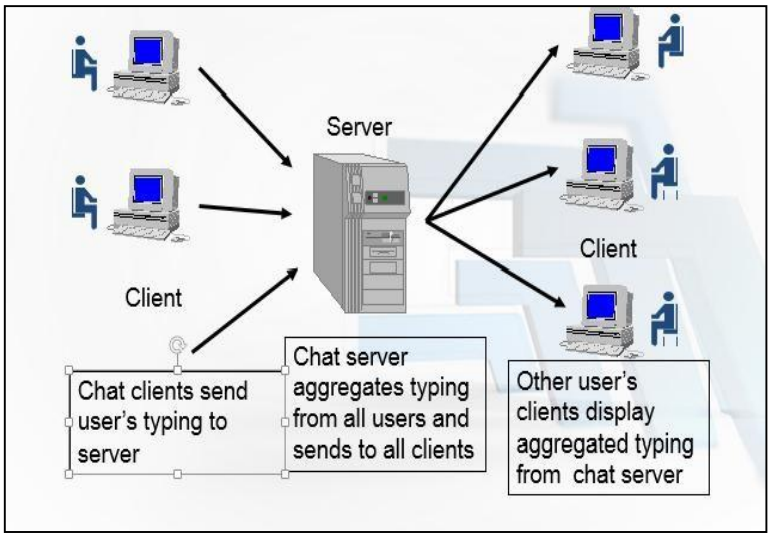
BB. Client Server Architecture

Client-server architecture is architecture of a computer network in which many clients (remote processors) request and receive service from a centralized server (host computer). Client computers provide an interface to allow a computer user to request services of the server and to display the results the server returns. Servers wait for requests to arrive from clients and then respond to them. Server is powerful in terms of processing and the file server, printer server or the email server. The client is connected to the server via communication link. The client interacts with the server when it requires access to any additional functionality that does not exists in its own system client system provides interfaces to access and utilize server resources.



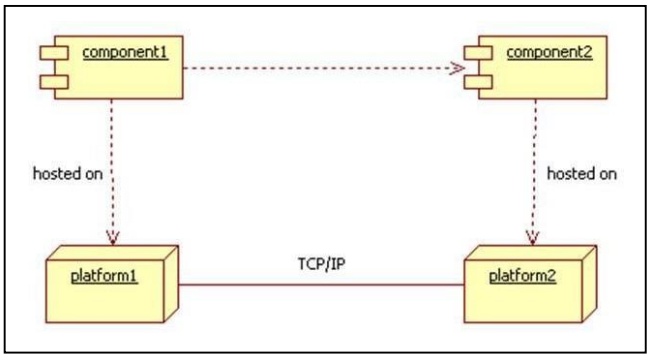
CC. Example – Client Server Architecture

The client server architecture is best elaborate with the example of chatting over the internet where the communication takes places between multiple clients with multiple clients. The figure 8 shown at the right side explains further. The chat clients send the user typed message to the server, the server then aggregate typing from all the clients and forwards it to all the clients, user's clients on the other side display the aggregated typing from the chat server.



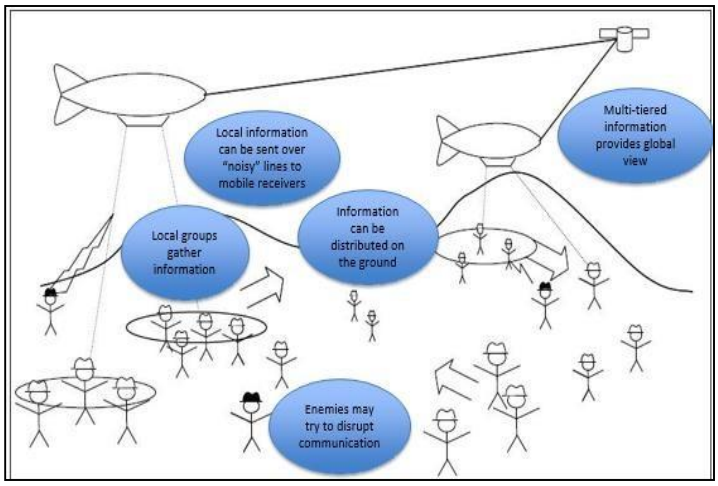
DD. Distributed Processing

Distributed system is combination of multiple components located on network and these components communicate and coordinate their action by passing messages. These components interact with each other to achieve a common goal. The processing is divided into multiple systems increasing the processing power. In distributed computing, a problem is divided into many tasks, each of which is solved by one or more computers, which communicate with each other by message passing as mentioned earlier.



EE. Example of Distributed Processing

TO BE DONE BY SHERAZ SB



FF.

Module 04: Fact Finding Technique

A. Examine Document Basics

Examine documents is one of the fact finding techniques and it is the first step in database development. This technique is critical to the design and development of database as it put some light on how the need for a database arose and what symptoms and problems in the current system lead to this decision. It is also helpful in studying the current system and the in the identification of the effected part of the organization. It is also valuable tool for the identification of the data to be collected and reported by the system.

Purpose of documentation	Examples of useful sources
Describes problem and need for database	Internal memos, e-mails, and minutes of meetings Employee/customer complaints, and documents that describe the problem Performance reviews/reports
Describes the part of the enterprise affected by problem	Organizational chart, mission statement, and strategic plan of the enterprise Objectives for the part of the enterprise being studied Task/job descriptions Samples of completed manual forms and reports Samples of completed computerized forms and reports
Describes current system	Various types of flowcharts and diagrams Data dictionary Database system design Program documentation User/training manuals

GG. Documenting the Examine Document

Documenting the existing systems of the documents leads to the identification of the type of data coming in and out of the system. It lays down the foundation of the requirements for the development of databases. Documenting the existing system and documentation identifies the data being handled and the relationship between the data. This relationship explains how the data flow from one part to another thus making a better and efficient system. In the figure on the right there is data about customer, and services requested (problems are categorized into two categories). The purpose of the figure is to show that by examining the documents within organization difference stakeholder holders, information flows are identified within the system which are difficult to understand by listening..

CUSTOMER SERVICE REQUEST

CUSTOMER INFORMATION
Date
Customer Name
Address
Phone

SERVICE REQUEST INFORMATION			
MERCHANDISE PROBLEMS:		DELIVERY PROBLEMS:	
Order not filled		Bad address	
Defective merchandise		Customer not in	
Repair problem		Delayed/lost in transit	
Wrong merchandise sold		Damaged in transit	
Amount charged in error		Mdse. Missing in package	
Credit/refund not issued		Other	
Other			

OTHER INFORMATION
REMARKS:

HH. Conducting Interviews Basics

Interviewing is the most commonly used, and normally most useful, technique for fact finding. It is a technique where the business analyst gathers information by having face-to-face interaction. There can be multiple objectives to using this technique such as fact finding or verifying the

facts, getting more diverse ideas and information, gathering the requirements, involving the end user to generate enthusiasm and to make them feel engaged in the whole project.

II. Types of Interview

There are two types of interview: unstructured and structured. The details of them are as follows:

- *Unstructured Interviews:* These interviews are conducted with a general goal in mind with very few specific questions. The interviewer let the interviewee to drive the discussion and provide direction to the interview.
- *Structured Interviews:* in this type of interviews, the interviewer has specific set of questions. Those questions can be further divided into two categories: Open-ended and Close-ended Question. Open-ended questions let the interviewee respond in any way that seems appropriate with no specific options to choose. Close-ended question restrict the answers to either specific choice or to a very short answers.

JJ. Example of Type of Interviews

Example 01: Unstructured Interview: Consider this interview between a researcher and a caregiver

- Researcher: "What kinds of illnesses do children in this village get?"
- Caregiver: "Well, diarrhea is the most common; also cough and cold, measles, fever, typhoid."
- Researcher: Are there any other during cold and hot weather?
- Caregiver: Pneumonia and whooping cough are very common in winter
- Researcher: Can you tell me something about measles?
- Caregiver: During measles it is very good to give cold water because it is very hot. The child should be kept away from other children
- Researcher: Why the child should be kept away from other children's?
- Caregiver: because measles can spread to other children very easily. Once a child get measles all the family get it
- Researcher: How does measles spread from child to child?
- Caregiver: I think it must happen through smell and sweat. May a child touches a thing which a sick child touches, I am not sure

Example 02: Structured Interviews: Consider the following questions:

- Could you tell me the tasks you do to prepare salaries per month?
- How do you know person monthly salary?

KK. Observing the Enterprise in Operations

Observation is one of the most effective fact finding technique as it give analyst a chance to participate in or watch the person perform the activity which results in deep understanding of the system. This technique is particularly useful when the validity of data collected through others methods is a serious concern or the system is too complex that the person or the end use in not able to explain it properly. To make sure the success of observation, it is necessary to know as

much about the individual and activity to perform as possible. Here the question like, when low, normal and peak periods for the activity being observed.

LL. Have and Have Not's for Observing

The observation should be recorded immediately or simply in parallel with the observation. The observer must not wait the observation to end and then write the complete process; rather the observer must take notes in the form or bullets during the observation in order to avoid losing any point. The observer must not interrupt or ask too many question from the end user while he is performing the activity otherwise the activity will get tempered and the observer will get the real information.

MM. Research

It is very much valued to research the solution for the problem in hand. Internet, journals, reference books are good sources of information about how other have solves related problems or what solutions are available. It can save a lot of time as we can get ready to deploy solutions to fulfill the requirements. It also a helpful tool in validating the information gathered thorough observation as the chances are there that the actual execution of the activity is different from what is observed.

NN. Questionnaire

Another useful method for fact finding is to conduct surveys through questionnaires which are specific purpose document to gather information from a large number of people while maintaining some control over the responses. There are two types of questions that can be asked in a questionnaire and the details are given below:

- *Free- Format Questions*: it offers greater freedom to the respondent in answering question as the respondents have to answer the question in the space provided after the question. The responses are more subjective in nature and difficult to tabulate.
- *Fixed-Format Questions*: These kinds of questions require specific responses and the individuals have to choose from the options available. The responses are more specific which make the responses much easier to tabulate. The disadvantage of this kind of questions is that the researcher doesn't get additional information that might be worth knowing.

OO. Example of Questionnaire

Consider the following requirement and observe the question in the questionnaire shown in the figure 12 below:

Requirement: What are current trends among students in terms of interest in media programs?

1. Are You: a. Male b. Female	2. When was the last time you saw a file, what was it?
3. How old are you? Years……. Months……	4. How many hours a day you spend watching, reading or listening to the following a. TV b. Radio c. Internet d. Print (Magazines / Newspaper)
5. Who do you live with at home? (be Specific)	6. Please list your top three TV programs a. b. c.
7. Do you have a part-time job? a. Yes b. No c. Paid d. Voluntary If so, please describe what you do	8. List in order of preference (1 being more preferred, 5 being least) which genres (types of programs) you watch: a. Sports b. Soap c. Sitcom d. Documentaries e. Film
9. Please list you top 3 favorite food? a. b. c.	10. You are going to help organize some kind of music even for your year group: what type of music / bands would you want to play?

Figure 1: Sample Questionnaire for Fact-Finding

Module 05: Implementing Fact Finding Techniques

A. From Fact-finding to Requirement Writing

Requirement writing is the very next step after fact finding in the cycle of database design and development. The requirements defining the systems are to be converted in written format. Along with the system definition it also contains and explains the data flow in the system and the relationships between the data. The requirement or the specification document describes in details the data to be held in database and how the data is to be used (i.e. the processes and the relationships). These requirements are specific and measureable in nature. The system requirement like how fact the transaction must run or how many transactions must be processed in specific interval of time form the basis of number of decision in database design and development.

PP. Example of Translating Interview to Requirements

Consider the following example in which the interview asks questions to get the requirements for high level database for rental system of a certain plaza. Following the interview are the requirements gathered through the interview.

Interview:

1. Interviewer: How many building types are there for rent?
   Interviewee: Primarily Apartments, some Offices and few Shops
2. Interviewer: How many are total apartments, total offices and total shops
   Interviewee: Ahhhh… yes 15 Apartment, 5 offices and 3 shops
3. Interviewer: How many floors are there in the plaza?
   Interviewee: 6 floors but we have apartments only on 3 floors and on remaining floors we have shops and offices
4. Interviewer: Do we have fix rent for each apartments or does it varies from Floor to Floor
   Interviewee: Rent varies from Floor to Floor of apartment.
5. Interviewer: Did you send rent bill / receipt at the end of month?
   Interviewee: We usually send bill for two-months but receive installments on monthly basis also
6. Interviewer:, can you tell me specific data you store about customer into the system
   Interviewee: We usually ask for name, address and phone no
7. Interviewer: Did you assign any number to customer
   Interviewee: Yes, I miss out
8. Interviewer: If multiple apartments / shops / offices are rented to same customer, then do we have same contract no and customer no
   Interviewee: Good question, let me think….pause, we usually assign different contract no but one bill to such case.

Requirements:

In a plaza renting system, there are three types of buildings primarily apartment, offices and shops. There are multiple floors in a plaza and for 3 floors apartments are allocated but for remaining there can be mix of shops and offices. When customer is registered usually customer id, name, phone and address is stored as profile and bill is generated for the rent on bi-monthly basis but customer has an opportunity to pay the bill on monthly basis. Contract is signed and uploaded against every building type. Contract id assigned to every contract. It is possible that a customer may rent multiple building types, in this case different contract no is assigned to same customer. One Bill is generated for customers with multiple building types are generated.

QQ. Example of Translating Questionnaire in Requirements

Following is the sample questionnaire to elaborate the translation of data gathered through questionnaires into requirements.

Questionnaire:

1. What is price (person person) range of food offered?
    - 500-1000
    - 700-1200 **(checked)**
    - 1000-1500
2. Is cost of dish calculated on the basis of?
    - Cost of ingredient only**(checked)**
    - Cost of ingredient and electricity also
    - Cost of ingredient plus % of profit
3. Is % of Profit?
    - 20%**(checked)**
    - 15%
    - 10%
    - None of the Above
4. Are ingredients and Quantity mentioned in?
    - Menu
    - Recipe **(checked)**
    - Restaurant Manual

Requirements: the requirements can be concluded as the following:

In a restaurant management system there are many dishes offered either as individual serving or buffet. Price range between 700-1200 is targeted which will include drinks and taxes also. Cost of every dish is calculated individually depending on ingredients involved only. Each ingredient is listed in recipe and quantity is also mention. Cost is calculated as per market rate of ingredient with 20% profit to cater for management and miscellaneous cost.

RR. Defining System Definition from Requirements

System definition is defined by listing down the details of the whole system. These details when properly concluded form the basis of decision regarding the scope and the boundaries, the

architecture and the design of the system to be developed. Concept definition artifact lays down the foundation of system definition; concept definitions are activities in which the needs and the requirements of all the stakeholders are closely examined before defining the system. The concept definition is the articulation of System-of-Interest (Sol) – the model of system on the basis of needs and requirement – which is a collective set of all elements of any system being considered by the lifecycle.

SS.Process of Extracting System Definition

System definition is based on the concept definition, a deep understanding of concepts, needs, expectations and stakeholders' requirements. System is further divided into subsystems through top-down decomposition of system-of-interest in order to reach the exact destination. The same model is shown in the Figure 13 on the right side. The needs and the concepts along with the stakeholder requirements are gathered to form a system-of-interest and this is further divided into subsystem to fulfill the desired requirements.

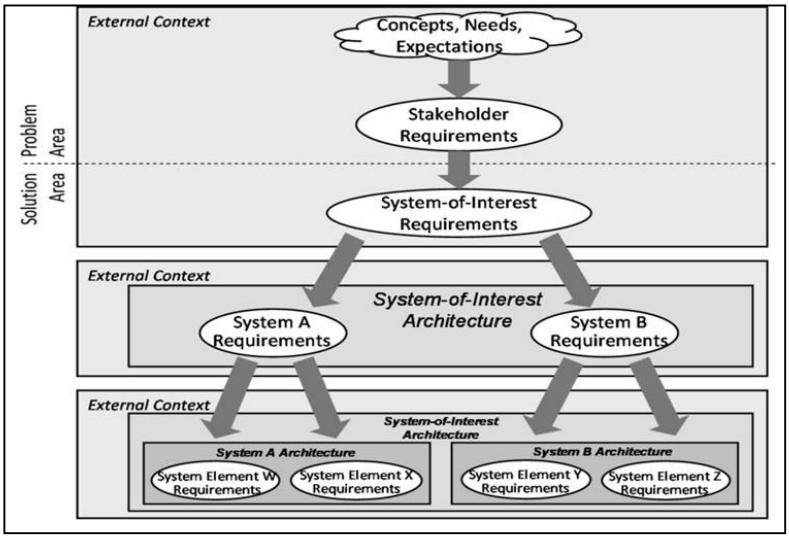


Figure 2: Process of Extracting System Definition

The system should list the entire important feature for a database system and these should include the followings:

- Initial database size
- Database rate of growth
- The types and average number of record searches;
- Performance
- Security;
- Backup and recovery
- Legal issues etc.

Module 06: Process to Database Design

A. Motivation for Requirements

The requirement gathering process identify the actual need, purpose or the potential update need of a system and these requirement are then base the foundation of system design and development or system update. The requirement gathering process is a non-technical process and it is just the communication of the requirement. This communication is important because this is the thing on which the development of the required system or software is dependent.

The requirement which can be defined as the conditions or the capability, to which a system must conform, can be related to the functionality, usability, reliability, performance or the supportability of the system. There are few challenges in gathering or documenting requirements, discussed in the following lines.

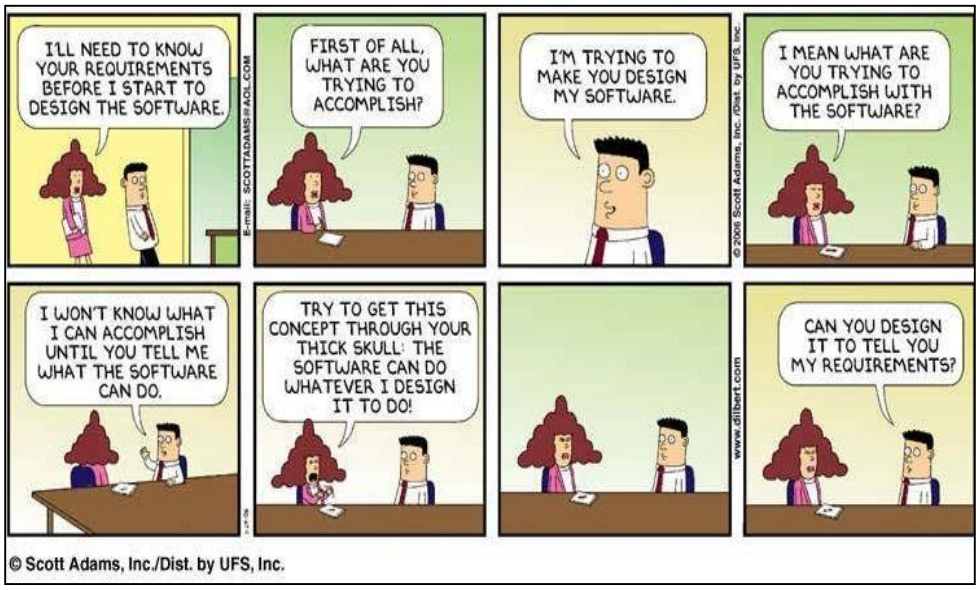


Figure 3: Motivation for Requirement Gathering

TT. Major Challenges in Requirement

The major challenges can in requirement documentation or gathering are explained below:

Missing Requirement: the person gathering or documenting the requirements can miss some important requirements because of human error or any other issues like poor observation, poor analysis of the requirements, the explaining the facts during fact-finding phase (the interviewee, the performer etc.) is unable to explain the fact properly and likes.

Wrong Requirements: there are chances that the requirements are wrongly communicated, exhibited or misinterpreted during requirement gathering and documenting phase.

Changing Requirements: the requirement can change over the period of time. The change can occur in data that is to be processed by the system, the flow of data, and the relationships, the

features required by the end users or the delivery schedule disturbing the development process. The attributes of the requirements are explained in the following parts.

UU. Attributes of Software Requirements: - Verifiable

The requirement should be communicated in such so the testing team should be able to verify it. The requirement will be considered as verifiable if there is some process through which a person or a machine can check that the software product meets the requirement. The following example explains the concept:

- Requirement: Loading time should be as minimum as possible?
- Question: What Qualifies to be minimum Loading Time?
- Implication: Not Verifiable!!
- Solution: Loading time should be 2 sec

VV. Attributes of Software Requirements: - Completeness

A complete requirements specification must precisely define all the real world situations that will be encountered and the system's responses to them. It must not include situations that are subjective in nature, non-verifiable or non-specific. In simple words, there is no room for guess work in documenting the requirements. Consider the following example:

- Requirement: On Power Loss, backup power should support Normal Operations
- Question: What is definition of Normal Operation?
- Implication: Subjective, Incomplete
- Solution: On Power Loss, backup power should support Normal Operations for 30 min

WW. Attributes of Software Requirements: - Consistent

The requirement is consistent if it is not in conflict with any other existing requirements. Conflict may arise due to variation in language patterns, use of different vocabulary or multiple descriptions of a single requirement. The requirement must not oppose other in terms of physical or functional requirements. The vagueness in the requirement is the quality destroyer. For example: the requirement says perform function X after both A and B has occurred, or perform function X after A or B has occurred. The requirement stated in such way may leads to failure. Another example:

- Requirement 01: The customer support should be ITIL compliant
- Requirement 02: The network support should be ISO – ITIL 3.0 compliant
- Question: Is ITIL and ITIL-3.0 same or different?
- Implications: No Consistency
- Solution 01: The customer support should be ITIL 3.0 compliant
- Solution 02:The network support should be ISO – ITIL 3.0 compliant

XX. Attributes of Software Requirements: - Traceability

Consider the following example:

- Requirement: System must generate a batch report when batch is completed and a discrepancy report when batch is aborted

- Issue 01: How uniquely would you be able to identify the unique batch report?
- Issue 02: What if requirement has changed and discrepancy report is not required?
- Implication: Zero Traceability
- Solution 01: System must generate a batch report when batch is completed and aborted – Assign Unique Requirement Number
- Solution 02: System must generate a discrepancy report when batch is aborted or completed – Assign Unique Requirement Number

Now understand the concept. A requirement is traceable if both the origins and the references of the requirements are available. Traceability of the origin or a requirement can help understand what modifications have been made to the requirement to bring the requirement to its current state. Traceability of references is used to aid the modification of future documents by stating where a requirement has been referenced. By having foreword traceability, consistency can be more easily controlled. Making requirements traceable is also useful in coding and testing of the system.

YY. Requirement Engineering Discipline

Requirements engineering emphasizes the use of systematic and repeatable techniques that ensure the completeness, consistency, and relevance of the system requirements [Sommerville 1997a]. Requirement engineering is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, Requirement Engineering acts as the bridge between the real-world needs of users or customers and the capabilities and opportunities afforded by software or system. Specifically, requirements engineering is divided into requirement management and requirement development which is further divided into requirements elicitation, analysis, specification and verification.

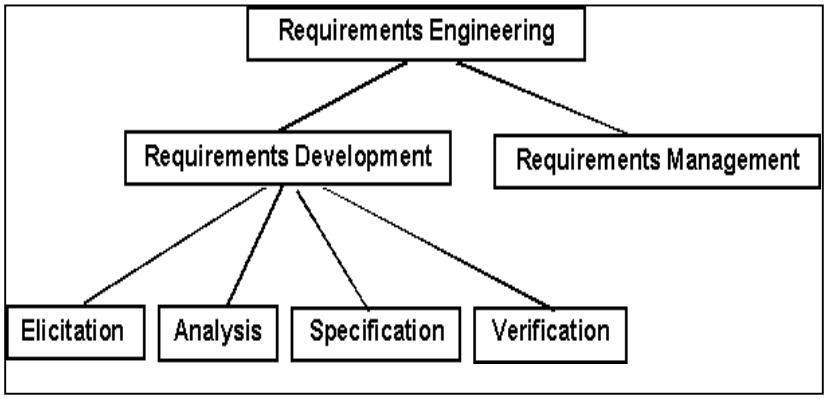


Figure 4: Sub Disciplines of Requirement Gathering

ZZ. Requirement Management

Requirements management is the process of scheduling, coordinating, and documenting the requirements engineering activities (that is, elicitation, analysis, specification, and verification) [Dorfman 1997a]. This discipline takes into account different strategies to keep the project scope within the baseline. Besides, it also monitors and controls the scope of the project and provides a mechanism to handle the proposed changes in the project scope. This discipline of the requirement engineering reviews and analyzes the changes to know the impact of them on the product line (impact on delivery schedule etc.) while communicating it to the relevant stakeholders.

AAA. Example of Requirement Management

Consider the following example of requirement change from the client side:

- Baseline Requirement: A - 22-May-2015
- Development as per A: 2-June-2015
- Requirement Change by Client – B: 10-June-2015
- Demo Day: 15-June-2015
- Client: This is not what I was expecting
- Reason: Client referring to B, Development to B!!

BBB. Requirement Development (RD)

Requirement development is a part of requirement engineering in which the requirements are gathered and then turned into specification of what system must do, or the scope or features of the software system. Requirement development activities are further divided in four activities namely Elicitation, Analysis, Specification and Verification (all of them are explained later). While defining the scope of the project, RD specifically identity and communicate what not to deliver. The artifact of RD process is the baseline requirements (defined schedule).

CCC. Requirement Elicitation or Gathering

Requirement elicitation is all about gathering the system's requirement and this have been discussed in detail in lesson 04 fact finding techniques. The aim is to apply all the techniques here to get the real requirements of the client. One technique is not just enough for this purpose because every requirement gathering has its own pros and cons and to eliminate the limitation of a single technique, more than one technique must be used. It could be interview, questionnaire, observation or any mixture of them as per the need.

DDD. Project Business Requirement

There must be a baseline for a scope in order to manage the scope in a proper manner. The priority of the scope is to be defined along with the baseline to deliver. This prioritizing helps the development team in scheduling their activities to deliver the project scope in time. This is all about defining and communicating the product roadmap. Let's take an example of this concept under next heading.

EEE. Example of Project Business Requirement

Product Roadmap:

- Total Scope of Project: 20 Features
- High Priority Features: 8 (Agreed by Client and PM)
- Medium Priority Features: 7
- Normal Priority: 5
- Release – 1: 8 Features with X Date
- Release – 2: 8 + 7 Features with Y date (medium priority)
- Release – 3: 8 + 7+ 5 Features with Z Date (normal priority)

FFF. Get User Involvement

Lack of user input can make a project very challenging. User involvement is of utmost important at every stage of system development. First of all, at the time of requirement gathering it is important to get the users or all stakeholders involve in order understanding the real issues in the current system and the potential solution to be developed. By understanding the skills and capabilities of the users, it would be possible to tailor the user interface to the need of particular type of users to ensure optimum usability. The concept of 'Product Champion' is worth mentioning here. The product champion is the person who is officially responsible for the delivery of the system. This person helps the stakeholders and the project manager reach a shared vision for a product, and then defines and initiates the product within that vision. The product champion have technical competence, knowledge about the company and markets and he is able to get as much knowledge as possible that if the client's representative leave, valuable knowledge doesn't leave with him.

GGG. Define Quality Metrics

A metric is a measure and combining it with the quality means the measures of quality. The quality is obviously what is defined by the customers. If the system provides what customer needs, that can be called a quality system. But there is much more than that. The focus of the quality metrics are non-functional requirements that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. Broadly, functional requirements define what a system is supposed to do and non-functional requirements define how a system is supposed to be. The few quality metrics for a system are:

- Reliability
- Robustness
- User Friendliness

HHH. Example of Quality Metric – Reliability

Reliability can be defined as the capacity of the designed or developed system to perform as it is required to perform over time. It can also be defined as resistance to failure over time. Reliability is all about stability of the developed system or its ability to perform what it is designed to perform. Following example will explain further:

- Example: We have developed a program for a Web server with the failure intensity objective of 1failure/100,000 transactions. During testing, the program runs for 50 hours, handling 10,000 transactions per hour on average with no failures occurring.

III. Example of Quality Metric – Robustness

Robustness is the ability of a computer system to cope with errors during execution. Robustness can also be defined as the ability of an algorithm to continue operating despite abnormalities in input, calculations, etc. The question or the check for this metric is the question that does the system crashes on abnormal inputs. For example: How algorithm behaves in case of abnormal input. If user input character A instead of numeric value then does system crash?

JJJ. Example of Quality Metric – User Friendliness

User Friendliness is synonymous with the usability and this term can be defined as the ease of use or the learn ability of the developed system. User friendliness is the extent to which the developed system can be used by the end user to get the desired results with effectiveness and ease of use. Most clients reject the software on the basis of the interface that is not too user friendly. This is where the requirement gathering becomes most important. The solution to this issue is to get the details on how the client wants it to be. For example the color scheme of the software, first ask the client for the color theme, get the color code or the sample image and then use the mockups of that image.

KKK. Requirement Analysis

After gathering the requirements the next step is to analyze the requirements. The basis of this analysis is the attributes of the system requirements mentioned in the same lesson above (heading C – F). This stage breaks down functional and non-functional requirements to a basic design view to provide a clear system development process framework. A thorough requirement analysis process involves various entities, including business, stakeholders and technology requirements. This stage also identifies the conflict with the base line as the prototype is made and the further development needs are clear at this stage.

LLL. Requirement Prioritization

At this stage the priority of the requirement is defined. The priority is defined in coordination with the development team and the client and the priority is based on some statistics of facts. The limitations of the project are also identified here. This prioritization helps in the stage-release of the system as the development of the system is divided into stages based on the priority. The same example mentioned in the section M of the same lesson also fits here where the requirement are prioritized and the release planning is done on the basis of it. The approach is to categories requirements into High, medium and low on some statistical basis.

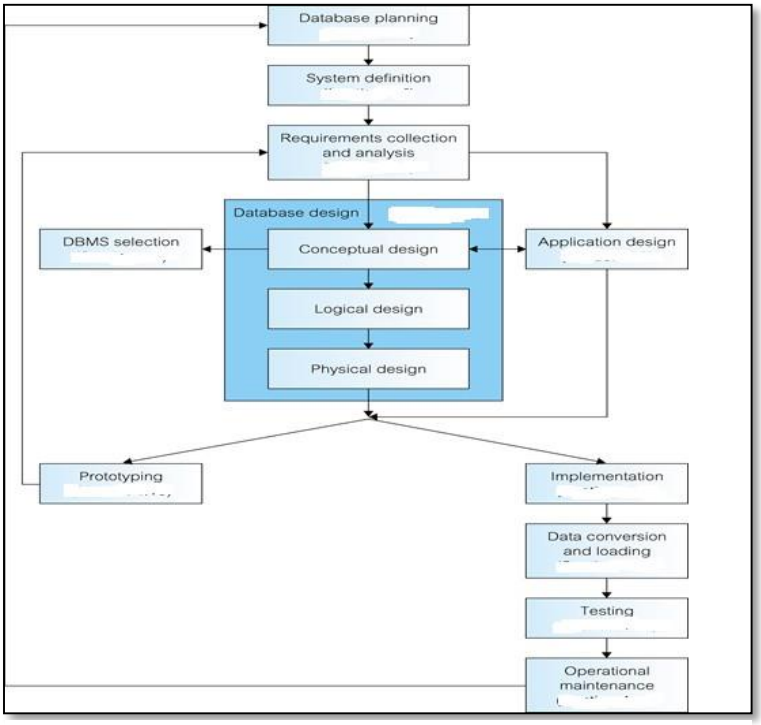


Figure 5: Process to Database Design

MMM. Requirement Specification

Requirement specification is the result of requirement analysis. Requirement specification established the basis for the agreement between clients and the development team on what the software product is to do and what not to do. The requirements are further classified into functional and

nonfunctional categories and recorded in mutually agreed document. This document is called Software Requirement Specification or SRS. This document enlists enough and necessary requirements that are required for the project development. Clear and thorough understanding of the system to be developed is making sure and this is achieved through continuous communication between client and project development team.

NNN. Requirements Verification

The correctness and the completeness of the system requirements is again verified at this stage, just to make sure that the system in the manufacturing process will ensure user need and requirements. Verification can also be seen as the process of checking that the software meets the specification. The goal of requirement verification is to ensure that requirements are good enough to proceed to design phase and development phase. This stage consists of thorough inspection and verification of system requirement as the defect and bug fixing later cost too much. Defect is defined as the wrong definition of the client's requirements. Defect is not something that is not working, it is simply working but not required part of the system and this can be detected in production phase. On the other side, bugs can be defined as the noncompliance with the agreed behavior, or software is not doing what it is supposed to do. This is a deviation from the requirements of the system.

Module 07: Relational Database

Relational database can be defined as a database whose organization is based on the relational model of data in which all the data is presented in the forms of tuple (also known as a record or a row) which are further grouped into relations. The relational model was first proposed by E. F. Codd in his seminal paper 'A relational model of data for large shared data banks' (Codd, 1970). There are some rules proposed by E. F. Codd which any database must follow to be called a relational database.

A. Codd Rule No. 01: The Information Rule

"All information in the relational database is represented in exactly one and only one way—by values in tables."

All the information (data or Metadata) must be stored in a table format. This is the only way data is to be stored. More specifically the storage mechanism should follow the row and column format. This rule is an informal definition of a relational database and indicates that every piece of data that we permanently store in a database is located in a table.

B. Codd Rule No. 02: Guaranteed Access Rule

"Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name."

This rule emphasizes the importance of the primary keys in locating the data in any database. The table name locates the correct table, the column name finds the correct column, and the primary key value finds the row containing an individual data item of interest. In other words, each piece of data is accessible by the combination of table name, primary key value, and column name.

C. Codd Rule No. 03: Systematic Handling of Null

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

Null cannot be considered as zero or empty, it is just something we just don't know and lack of information does not mean no information. The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Example:

- Question: How many Books Ali have?
- Answer: May be 5 or not known (null)
- In Database column representation not known will be using NULL not using any numeric value
- Conclusion: There will be representation of null beside numeric value in database

D. Codd Rule No. 04: Active Online Catalogue

"Data Dictionary and database description must be available in database"

The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself. The system must support an online, inline, relational catalog that is accessible to authorized users by means of their regular query language. That is, users must be able to access the database's structure (catalog) using the same query language that they use to access the database's data.

E. Codd Rule No. 05: Powerful Language

"A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations."

A linear syntax can be defined as a language which is parsed from left to right which implies the ability to write code without the use of line-feed or carriage-return characters. Line-feed and carriage-return are much closed concepts and can be lumped together and these are special character or sequence of characters signifying the end of a line of text and the start of a new line.

OOO. Example of Power Language

- Linear Syntax:

  Void main () { cout<< "Hello";}- **C++ Language**
- Language rely on semi column and parenthesis to separate code block
- Line Feed or Carriage Return (i.e. moving to next line) is not required to separate code block

F. Codd Rule No. 06: View Updating Rule

"All views that are theoretically updateable are also updateable by the system."

This rule deals with views, which are virtual tables used to give various users of a database different views of its structure. It's one of the most challenging rules to implement in practice, and no commercial product fully satisfies it today. User can be given access to selected columns as per rights and privileges. Views are created on the basis of tables. Views should be updateable without any restriction and should be updateable like tables. SQL have limitation to update views and view with single table and without group by clause are updateable only.

G. Codd Rule No. 07: Relational Level Operations

"Insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table."

The system must support set-at-a-time insert, update, and delete operators. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

H. Codd Rule No. 08: Physical Data Independence

"The way the data is stored physically must be independent of the logical manner in which it's accessed."

This rule says that Changes to the physical level (how the data is stored, whether in arrays or linked lists etc.) must not require a change to an application based on the structure. This is saying that users shouldn't be concerned about how the data is stored or how it's accessed. In fact, users of the data need only be able to get the basic data they need.

I. Codd Rule No. 09: Logical Data Independence

"Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure."

The logical data in a database must be independent of its user's view. How a user views data should not change when the logical structure (tables structure) of the database changes. This rule is particularly difficult to satisfy. The crux of this rule is that the business logic and storage mechanism works in total independence with the way user view the data and the user should be unaffected with any change in storage mechanism or business logic.

Example:

- Structure of Table – Employee (id, name, DOB)
- Application Program Access: name, id, DOB from Database
- Database Structure need no change

J. Codd Rule No. 10: Integrity Independence

"A database must be independent of the application that uses it."

All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface. Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure. Logical data independence is more difficult to achieve than physical data independence. Integrity rules should be stored in the data dictionary. Primary key constraints, foreign key constraints, check constraints, triggers, and so forth should all be stored in the data dictionary.

K. Codd Rule No. 11: Distribution Independence

"The database language must enable application programs and terminal activities (storage or data retrieval) to remain logically unimpaired whether and whenever data are physically centralized or distributed."

This rule says that the database language must be able to manipulate data located on other computer systems. The end-user must not be able to see that the data is distributed over various

locations. Users should always get the impression that the data is located at one site only.User have no concerns with the internal storage strategy. This rule has been regarded as the foundation of distributed database systems.

L. Codd Rule No. 12: Non-Subversion Rule

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

In simple words, there should be no backdoor to disable the integrity and constraints defined. This rule requires that alternate methods of accessing the data are not able to bypass integrity constraints, which means that users can't violate the rules of the database in any way. If low-level access is allowed, it must not bypass security nor integrity rules. For example, a backup or load utility should not be able to bypass authentication, constraints, and locks etc. in Oracle there is violation of this rule.

PPP. Database Schema

Database schema is better explained as the group of database objects like tables, indexes, triggers and constraints that are related to each other. It can also be referred as a blueprint of how the database is constructed or the structure of the database. Schema is a logical way of grouping database objects like tables, indexes and views. Usually the schema belongs to a single user or application but a single database can hold multiple schemas belonging to different users or applications. Schema groups the tables by owner thus implementing first level of security as the user can see only the tables that belong to him. Schemas have privileges to have authorized access to Objects and users can be assigned a username and password to have authorized access. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.

QQQ. Database Schema Example

Consider the following example in which an admin creates a schema, create a user and sets a password. This user named test in the example can create database objects and can assign rights to other database users.

- Admin Create Schema: Test (username)
- Admin assign Password: 123
- Admin grant privileges to Test
- Test: Login with password
- Test: create table, create constraints, create functions, create triggers – Database Objects
- Test can grant Privileges to other users Database Users

RRR. Introduction to Keys

Each table in a database contains multiple numbers of rows and each row is referred to as one database record. Each record or the row is assigned a unique number to manage the record

reference which is known as a key. This key equips each row or database record with a unique identification which can be easily tracked.

SSS. Structured Query Language

SQL, Structured Query Languages a special-purpose programming language designed for managing data held in a relational database management system (RDBMS) and it is considered to be a declarative language. This language is composed of a set of commands that enables creating database and table structures, data manipulation and administration. There are subsets of SQL which are as follows:

- Data Definition Language (DML)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
- Data Retrieval Language (DRL)

TTT. Data Definition Language (DML)

Data Definition Language (DDL) is a standard for commands that define the different structures in a database and is responsible for creating database objects. DDL statements create, modify, and remove database objects such as tables, indexes, and users and all the objects are the part of data dictionary. Common DDL statements are CREATE, ALTER, and DROP.

- CREATE: to make a new database, table, index, user or view.
- ALTER: To modify an existing database object.
- DROP: To destroy an existing database, table, index, or view.

UUU. Data Manipulation Language (DML)

Data Definition Language (DDL) is syntax similar to a computer programming language that is responsible for manipulating Data in Database Table. More specifically, this language is used for selecting, inserting, deleting and updating data in a database. DML statements are used to work with the data in tables and are applied at the table level. DML comprises of the statement that can modify the stored data in tables but not the database schema or objects in data dictionary. Some common DML commands are:

- INSERT: to add new rows to a table.
- UPDATE: to change values already in a table.
- DELETE: to delete the rows in a table

XV. Data Control Language (DCL)

Data Control Language (DCL) is syntax similar to a computer programming language that is responsible for controlling the access to the data stored in a database. This is used to create privileges to allow users access to, and manipulation of, the database. Two important commands of DCL are GRANT (to give access rights) and REVOKE (to withdraw the access rights). It is also used to control transactional processing in a database and for this purpose the common commands are:

- COMMIT: to apply the transaction by saving the database changes.

- ROLLBACK: to undo all changes of a transaction.
- SAVEPOINT: to divide the transaction into smaller sections. It defines breakpoints for a transaction to allow partial rollbacks.

WWW. Data Retrieval Language (DRL)

Data Retrieval Language (DRL) is a command to retrieve data from a database object in the desired format. This is the most popular / flexible and the only way to retrieve data from a database. Command used for this purpose is called SELECT which allow us to specify the type of information which we want to retrieve.

Module 08: Conceptual Data Modeling and Entity Relationship Diagram Overview

A. Concept of Entity Relationship Diagram (ERD)

Entity–relationship modeling was developed by Peter Chen and published in 1976; it serves as building block of relational database design. Entity relationship diagram is a graphical representation of the relationships between data in a database. It is the result of using systematic process and it just only visualize the business data instead of defining the business process. In very simple terms, ERD is a visual representation of data that describes how the data is related to each other.

XXX. Rationale for ERD

It is the first diagram in database design which gives a higher level description of the system and actually visualizes the system requirements. It is probably the easiest way to describe the interaction or the relation between the different components by using different notations and to get a picture of functionality needed for the system. The data is represented as components (entities) that are linked with each other by relationships that express the dependencies and requirements between them thus making them easy to understand.

YYY. Components of ERD

There are three main components of ERD and these are:

- Entity
- Attributes
- Relationships

ZZZ. Entity & Attributes

The word entity is rooted from the Latin word "en" which means being. Entity is name of place, person or thing about which something can be stored in a system. An entity can be a real-world object that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity. An entity set is a collection of similar types of entities.

Entities are represented by means of their properties, called Attribute or Column. All attributes have values which are the qualities or data about Entities that is to be stored. An Attribute describes a property or characteristics of an entity. Continuing with the above example, a student entity may have name, class, and age as attributes. Attribute is the smallest storage unit of any database.

AAAA. Relationships

Relationship represents how data is connected among entities in a given System. The association among the entities can also be termed as relationships. In our school example, the two entities e.g. student and course have an association or relation with each other as student enroll in a course. Interaction among entities is captured using relationships.

BBBB. Process to Create ERD

The first step in creating the ERD is to define or determine the entities from the data. The second step is to chalk out the relationships and interaction between the entities identified in the very first step. The third step is to identify the cardinalities which is also known as degree of relationship which is the number of entities in an entity-set which are associated (or linked) to the number of entities in another entity-set. There are three degrees of relationship, known as: One-to-One, One-to-Many and Many-to-Many.

Module 09: Entities, Attributes and Relationship

A. Extracting Entities from Requirements

In the database design process, designers first produce requirements specifications from users, then transform the requirements into a formal representation schema. The Entity Relationship Diagram (ERD) model is one of the most popular of these formal representation schemata. It is a graphical way of displaying entity, relationship, and attributes types, which incorporates some of the important information about the situation, and it is considered to be a very natural and easy-to-understand way of conceptualizing the structure of a database.

Consider the following scenario and observe how the entities are being determined from the requirements.

Scenario:

In a Building apartment renting scenario, there are apartments, buildings and customers. There are multiple floors in the building and on each floor there are multiple apartments, floor can have zero or no apartment. Each apartment can be rented at most one customer but customer can rent out multiple apartments from same building and apartment can be available on multiple floors or same. At the end of month a receipt is generated against which a rent is deposited.

The following entities can be derived from the above stated scenario:

- BUILDING
- APARTMENT
- CUSTOMER
- FLOOR
- RECEIPT

CCCC. Extracting Attributes from Scenario

Considering the same scenario, attributes against each derived entities can also be determined. Recall the concept of attributes; an attribute describes the facts, details or the characteristics of an entity. From the above scenario, the following and many other attributes can be defined.

Entities & Attributes:

Entity	Attributes
Building	Building Name (bname), Address, Phone Number
Apartment	Apartment ID (aid), Covered Area, Status (available/rented), Rent…
Customer	Customer ID (cid), Name, Contact, CNIC
Floor	Floor Number, Number of Apartments
Receipt	Receipt ID (rid), Apartment Number, Date of Receipt, Amount

The ability to find meaningful names comes with fundamental understanding of what the model represents. Since the requirement specification is a description of a business, it is best to choose

meaningful business names wherever that is possible. If there is no business name for an entity, you must give the entity a name that fits its purpose in the model.

DDDD. Primary Key

The primary key of a relational table uniquely identifies each record or row in the table. As its name suggests, it is the primary key of reference for the table and is used throughout the database to help establish relationships with other tables. It usually comprises of a single table column, but may consist of a multiple columns as well. It is possible for a table to have more than one column with unique values in the table, however only one primary key can be defined. Each row in a table must have a distinct value (or a set of values) in the column marked as the primary key. As a rule, primary key cannot contain a NULL value.

EEEE. Identifying Primary Key from Sample Requirements

As per the above-explained concept of the primary key, and the requirement scenario mentioned in the beginning of this lesson, the primary key can be identified in the following manner.

Entities & Attributes:

Entity	Attributes	Primary Key
Building	Building Name***(bname)***, Address, Phone Number	Building Name – bname
Apartment	Apartment ID***(aid)***, Covered Area, Status (available/rented), Rent	Apartment ID – aid
Customer	Customer ID***(cid)***, Name, Contact, CNIC	Customer ID – cid
Floor	***Floor Number***, Number of Apartment	Floor Number
Receipt	Receipt ID***(rid)***, Apartment Number, Date of Receipt, and Amount	Receipt ID – rid

As a rule, a primary key should be minimal and it should not contain unnecessary information. So in the above case, building name fits into the criteria or becoming a primary as rest of the attributes can be complex containing unnecessary information. Apartment ID is unique and the apartments in the building can have same covered area, rent or status making these attributes common. The primary key for rest of the entities is determined in the very same way.

FFFF.Super Key

A superkey is a set of attributes within a table whose values can be used to uniquely identify a tuple or row in a database. A super key can be single or combination of multiple columns/attributes which can uniquely identify value(s) between multiple rows. There can be multiple superkeys in a table but no two rows will share same combination or pair or super key. The definition of a superkey is a set of columns in a table for which there are no two rows that will share the same combination of values. So, the super key is unique for each and every row in the table. Primary key is name after concept of Super and sub-set, so a super key is essentially all the superset combinations of keys, which will of course uniquely identify a row in a table.

A minimal superkey is the minimum number of columns that can be used to uniquely identify a single row. In other words, the minimum number of columns, which when combined, will give a unique value for every row in the table.

GGGG. Identifying Super Key from Sample Requirement

As per the requirement scenario of buildings and apartments, consider the following as identification of Super Key.

- Entities:
  - Building (bname, address, phone#)
  - Super Key:{(bname, address), (bname, phone)}
  - Minimal Super Key: bname

HHHH. Unique Key

A table in a database has unique key to identify a record or row within a table. In an entity-relationship diagram of a data model, one or more unique keys may be declared for each data entity. Each unique key is composed from one or more data attributes of that data entity. From the set of unique keys, a single unique key is selected and declared the primary key for that data entity. But unlike the primary keys, unique keys can have null values. Super keys, primary key can be declared as a type of unique keys.

IV. Candidate Key

A candidate key is the minimal number of attributes, whose value(s) uniquely identify each entity occurrence. The set of unique keys determined for a data entity is often referred to as the candidate keys for that data entity. A database table may have many candidate keys but at most one candidate key may be distinguished as the primary key. The candidate key is also termed as a potential primary key.

JJJJ. Identifying Candidate Key from Sample Requirements

Continuing with the same example of apartment rental for ease, the unique key and candidate keys can be identified from the scenario.

Entities & Attributes: (CK = Candidate Key & UK = Unique Key)

Entity	**Attributes**
Building	NTN# = **CK, UK**
Apartment	Apartment ID (aid) = **CK, UK)**
Customer	CNIC (UK) , Email (**UK)**

KKKK. Foreign Key

The foreign key is the field or attribute (or collection of fields) that is logically relate two table via common fields, in other words, foreign key is used to create relationship between two tables. The primary key of the first table or parent table becomes the foreign key of the child table. When an attribute appears in more than one relation, its appearance usually represents a relationship between tuples of the two relations. Foreign key is not the primary key by default but it can serve the purpose of the primary key, but this is not a rule. Just like unique keys, foreign keys can have NULL values.

LLLL. Example of Foreign Key

As an example to illustrate foreign keys, consider the requirement scenario mentioned above.

An receipt database has a table and each receipt is associated with a particular customer. Customer details (such as name and address) are kept in a separate table; each customer is given a ‘customerID' to identify it. Each receipt record has an attribute containing the customerID (cid) for that receipt. Then, the 'customerID' is the primary key in the customer table and that primary key will be the foreign key in the receipt table.

MMMM. Composite Primary Key (CPK)

Composite primary keys are the primary key which consists of more than one attributes. CPK combines more than one attribute of an entity to make a unique value. Sometimes it becomes necessary to combine more than one attribute in order to uniquely identify the records but as a rule, there will be just one primary key for each table. In relation between two entities, foreign key can become a part of CPK. CPK is unique as group and all the CPK values cannot be repeated. Besides, there are no composite foreign keys.

NNNN. Example of Composite Primary Key (CPK)

In our example of building & apartments renting scenario, a CPK can be created by joining two attributes of RECEIPT entity. For example, receipt ID and apartment number can be joined together to create a composite key which can uniquely identify each record. CPK are used where one attribute is not enough to uniquely identify records, CPK comes as a solution to this issue.

OOOO. Relationships

Relationship define how data is connected among the entities in a given system or in other words how one entity is logically connected with another entity of the system. Relationships in a database are said to be a combination of cardinality and optionality where optional relationship is one in which there may or may not be a matching record in parent / child table and cardinality represents the concept of “how many” and normally it is 0 or more. Equation for creating relationship is as follow:

$$Relationship\ (R) = Cardanilty\ (C) + Optionality\ (O)$$

The concepts of cardinality and optionality are explained below in details.

Relationships are bi-directional in nature; Relationship between two entities A and B is as follow:

i. Relationship from A to B
ii. Relationship from B to A

PPPP.Optionality with Example (…. Or___)

Participation in entity relationship is either optional or mandatory (…. Or_____). This means that one entity occurrence may or may not require a corresponding entity occurrence in a particular relationship. If you examine the *Parent & Child* relationship, it is quite possible for parent not to have any child record. Therefore, child is optional to parents.

On the other side, a child must have a parent and therefore, parent is mandatory to child. Hence minimum number of child record per parent in child table will show either 0 (…) or one

(_____) . A dotted or solid line shows this kind of relationship. To translate …… or _____ following rules are to be followed:

Optional: zero or one

Mandatory:____: Exactly one

QQQQ. Cardinality with Example (1 or >, <)

Cardinality expresses the minimum and maximum number of entity occurrences associated with one occurrence of the related entity. The cardinality tells what kind of relationship an entity has with another but it does not state whether it's mandatory or optional relationship. In the relational model, tables can be related as any of "one-to-many" or "many-to-many." This is said to be the cardinality of a given table in relation to another. Following rule are to be followed when translating cardinality

<or>: More than one

Cardinality is read with opposite entity.

RRRR. Basic Relationships

A cumbersome part of designing a relational database is dividing the data elements into related tables. Once you're ready to start working with the data, you rely on relationships between the tables to pull the data together in meaningful ways. The database system relies on matching values found in tables to form relationships. When a match is found, the system pulls the data from the tables to create a virtual record.

There are three kinds of relationships namely:

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

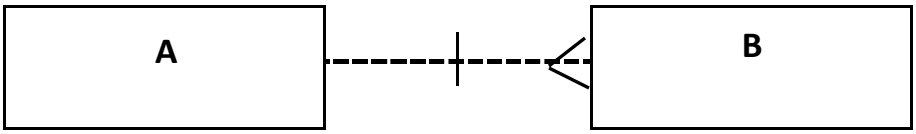
The first two types of the relationship have equivalent implementation in the database but the third type need conversion as it has no equal implementation in database. The details of these relationships are as follows.

a. One-To-Many Relationship

A one-to-many relationship is a type of cardinality that refers to the relationship between two entities A and B in which element of A may be linked to many elements of B, but a member of B is linked to only one element of A. One-to-many relationship is the most common type of relationship. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A. One-to-many relationship occurs when a parent record in one table can potentially reference several child records in another table.

As a rule, the Primary key of Parent table acts as Foreign Key in Child Table. Foreign Key is usually created as Composite Primary Key with PK of child table but this is no specific rule.

SSSS. Example-I of One to Many Relationship



Relationships are bi-directional:

A to B:

Optionality is Optional (zero or one) because line is dotted from A till half line and cardinality is more than one due to < symbol attach with B

Relationship:

Every instance of A is having one or more instances in B

B to A:

Optionality is optional due to dotted line from B till middle and cardinality is 1 because there is no > symbol with A

Relationship:

Every instance of B belong to exactly one A

Overall Relationship:

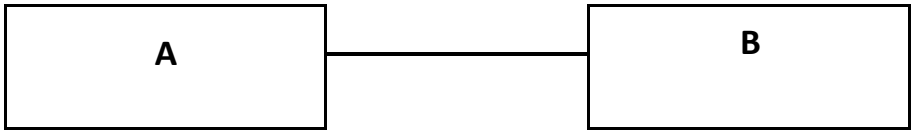
One to Many from A to B

b. One-To-One Relationship

A one-to-one relationship is a type of cardinality that refers to the relationship between two entities A and B in which one element of A may only be linked to one element of B, and vice versa. As the one-to-one label implies, in this relationship, one entity can be related to only one other entity, and vice versa. It is important to note that a one-to-one relationship is not a property of the data, but rather of the relationship itself as there is no parent-child relationship in on-to-one relation scenario. The following are rules to implement One-to-One:

i. There will be foreign key in any one of the participating table.
ii. Foreign key will be made as Unique key.

c. Example of One-to-One Relationship



A to B:

Optionality is mandatory (exactly one) because line is solid from A till half line and cardinality is also one since there is no < or > symbol attach with B

Relationship:

Every instance of A is having exactly one instance in B

B to A:

Optionality is mandatory (exactly one) because line is solid from B till half line and cardinality is also one since there is no < or > symbol attach with A

Relationship:

Every instance of B is having exactly one instance in A

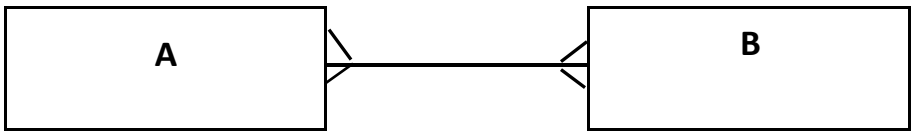
Overall Relationship:

One-to-One from A to B

d. Many-To-Many Relationship

A many-to-many relationship is a type of cardinality that refers to the relationship between two entities A and B in which A may contain a parent record for which there are many children in B and vice versa. This means that a parent row in one table contains several child rows in the second table, and vice versa. Many-to-Many relations are tricky to represent and are not supported directly in the relational environment. To represent this kind of relationships, a third entity or intersection table is created where PK of two table act as FK and CPK in third table.

e. Example of Many – to – Many Relationship



A to B:

Optionality is mandatory (exactly one) because line is solid from A till half line and cardinality is more than one due to < symbol attach with B

Relationship:

Every instance of A is having one or more instances in B

B to A:

Optionality is mandatory (exactly one) because line is solid from B till half line and cardinality is more than one due to < symbol attach with B

Relationship:

Every instance of A is having one or more instances in B

Overall Relationship:

Many- to- Many from A to B

TTTT. Referential Integrity Constraint

Referential integrity is a relational database concept in which multiple tables share a relationship based on the data stored in the tables, and that relationship must remain be consistent. The referential integrity rule is defined on a key (a column or set of columns) in one table that guarantees that the values in that key match the values in a key in a related table (the referenced value).In simple words, when a foreign key value is used it must reference a valid & existing primary key in the parent table otherwise the referential integrity will break. The child table can only be created if parent record is present as the child table is dependent on parent table. To create a child table, there must be a solid reference (parent table) as theorized by the referential integrity concept.

UUUU. Cascading

The purpose of the Foreign Key is to identify a particular row of the referenced table, it is required that the foreign key has a valid reference to from the referenced table or has NULL value. This rule is called the referential integrity constraint and violation of these constraints can create many issues. Cascading is the mechanisms to ensure the foreign keys have valid reference in the parent table. Cascading defines the behavior of foreign key when the record from the parent table is deleted or updated. To maintain referential and data integrity, cascading define two rules; CASCADE DELETE and CASCADE UPDATE. These are discussed in detail in the following lines.

VVVV. Cascade Delete

This rule states that if a row in the referenced table is deleted, then all rows in the referencing table with a foreign key value equal to the primary key value of the row should also be deleted. In simple words, if a parent row is deleted, then all the child records that are referencing to parent record will be deleted automatically. Consider the following example in order to understand the concept in a better way. The record can be deleted from the child and this will have no impact on the parent table.

Example:

Suppose company X has 2 tables, an Employee table, and a Department table. In the Employee table we have 2 columns – the employee ID and the Department ID. In the Employee Department table, we have 2 columns – the Department ID and the Department name for the given ID. Department ID is the foreign key in the Employee table. Now suppose we wanted to remove a department due to any reason, when a department is deleted, any rows in the employee table, that references the DepartmentID, are also deleted.

WWWW. Cascade Update

If the primary key value of a row in a referenced table is updated, all rows in the referencing table with a foreign key value equal to the primary key value of this row, should also be updated to the new value. As primary keys when migrated to another table to create logical relation becomes foreign key, so the value of foreign key must be dependent on primary key and hence it would be updated when the value of primary key is updated.

Example:

Continuing with the previous example of employee and department table, now suppose we wanted to update a department ID due to any reason, when a department ID is updated, any rows in the employee table, that references the DepartmentID, are also updated.

XXXX. Restrict Delete and Update

The concepts articulate that the parent record can neither be deleted nor updated if the child record exists. There is no restriction in deleting or updating the child records. The restrict Delete and Restrict Update prevents the deletion of data from the master table. In other words, restrict Delete or Update modifies the behavior of master table, not the child table.

YYYY. Surrogate Keys

Surrogate key is an artificial key which is not a part of the system. The purpose of the surrogate keys is to avoid the complexity of the composite primary keys. In order to enable surrogate, a single column is to be added in relevant table. Whenever the primary key is considered to be unsuitable, designers use surrogate keys which are artificially produced (most often system generated) and generally in numeric values.

ZZZZ. Example of Surrogate Keys

In our building-apartment rental scenario, the receipt ID can serve as an example of Surrogate Key. The receipt ID can be added automatically in a sequence. Instead of using a combination of one or more attributes of RECEIPT entity as a primary key, it would be more appropriate to use a sequential and auto generated ID as a primary key which can uniquely identify each record of RECEIPT.

Module 10: Extended Entity Relationship Diagram

A. Extended Entity Relationship Diagram (EERD)

The EERD incorporate the extension to the original Entity Relationship Diagram (ERD) Model with reusability concept. It was developed to reflect more precisely the properties and constraints that are found in more complex systems. Common attributes which are repeated among multiple entities are placed in Super Type / Super entity and are shared with Sub-types to avoid redundancy. Attributes and relationships of super types are shared with sub-types but this relationship is not bottom-up meaning Super type can't access any attributes or relationship of subtypes.

The EER model includes all of the concepts introduced by the ER model. Additionally it includes the concepts of a Sub-Type and Super-Type. Extended ERDs are high-level conceptual models that accurately represent the requirements of complex databases.

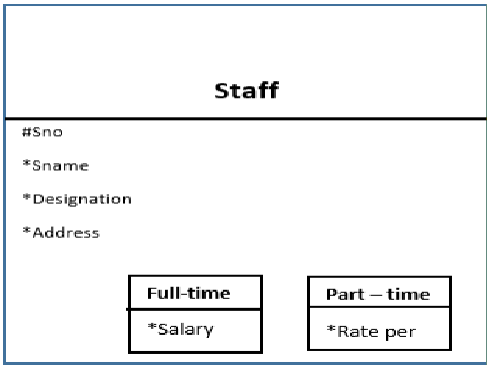
AAAAA. Super Types

Common attributes are placed in super type. It is also known as super class and many sub classes or subtypes can be derived from a super type. The details of the super types are available in the subtypes but on the other side, super types do not know anything about subtypes. In simple words, it can be defined as an entity type that has a relationship with one or more subtypes.

BBBBB. Sub Types

Sub Type or sub class is derived from the super class and it can have its own attributes and relationships. Besides having own relationships and attributes, the sub types inherit all the relationships and attributes of a super class. A class that is derived from another class is called a subclass (also a derived class). The class from which the subclass is derived is called a super class (also a base class or a parent class).

CCCCC. Example of Super Type & Sub Type

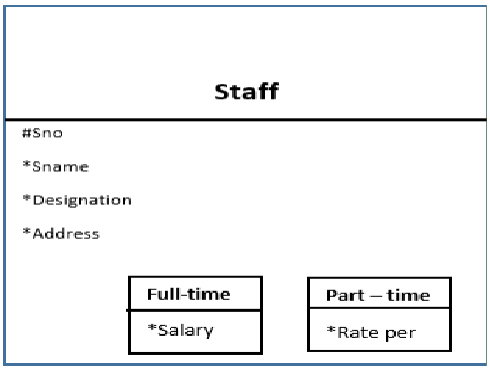


The diagram above explain the concept of super-type and sub-type. Common attributes (sno, sname, designation, address) are placed in Staff entity (Super type) as there will be multiple employee type (Full-time, part-time) and every employee type will share common attributes so instead of writing (sno, sname, designation, address) multiple times for Full-time and Part-time sub-types these 4 attributes are written only once and are shared by super type.

DDDDD. Exhaustive

It's the first property of Super & Sub Types which states that there should be at least two sub types and each subtype should have at least one attribute. Subtype without no specific attributes is not a valid Sub-type In other words, every instance of the super type is also an instance of one of the sub types.

EEEEE. Example of Exhaustive

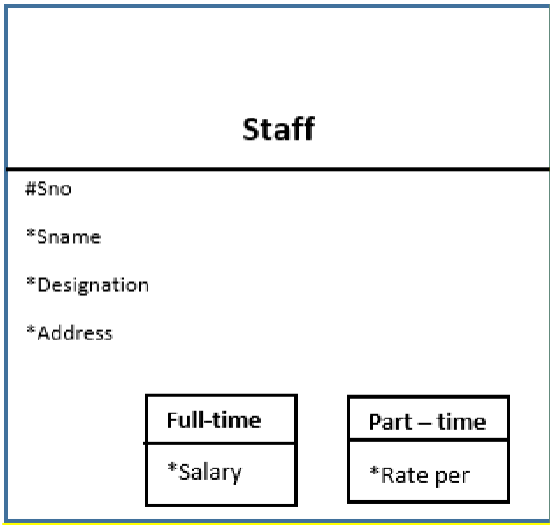


If we remove Salary attribute from sub-type Full-time then its not a valid sub-type because as per Exhaustive rule there has to be at least one specific attribute for sub-type to qualify to be a valid sub-type

FFFFF. Mutually Exclusive

The sub types are mutually exclusive as they don't know about each other. Attributes and relationship which a specific sub-type is having with other entities are not shareable or now known to other sub-types in the system. Every sub-type maintains isolation of relationships and attributes from other sub-types. This rule or property articulates that every instance of the super type is of one and only one subtype.

GGGGG. Example of Mutually Exclusive



In the diagram above, for example if there is sno=1 at any particular time, then this sno(1) can be associated with only one of its sub-type i-e either Full-time or Part-time; not the both sub-types. If sno=1 is associated with Full-time then sub-type part-time is not aware about this association due to mutually exclusive rule.

HHHHH. Example of EERD

Consider the following scenario:

In HR System there are different types of employees working in an organization mainly full-time, part-time. Only Full-time employee can avail the facility of loan however all type of employee can apply for study-leaves. Account department prepare the salary slip in the same way for all the employees.

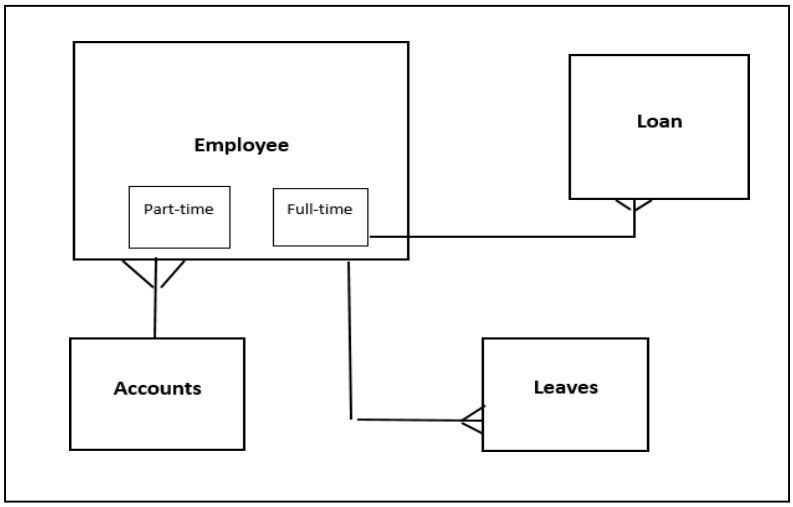


Figure 6: Example of EERD

Module 11: Example of Entity Relationship Diagram (ERD)

A. Example 01: Scenario and Generating ERD

Recall the Building – Apartment Renting Scenario:

Scenario:

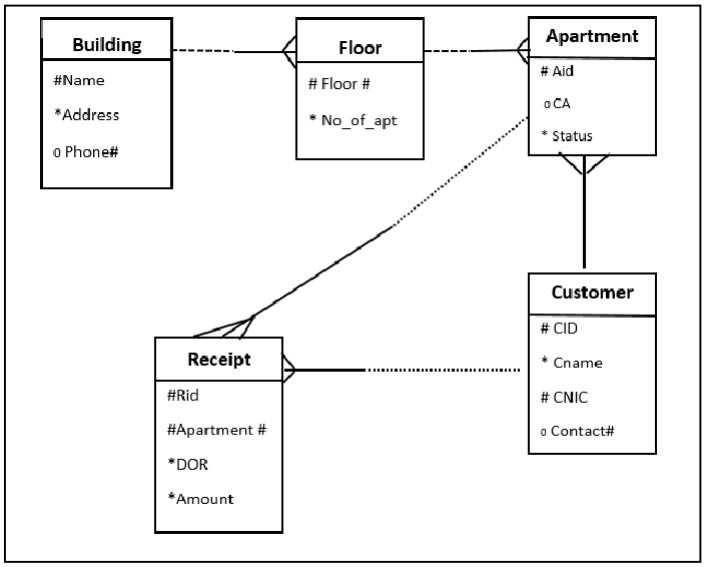
In a Building – Apartment renting scenario, there are Apartments, building and customer. There are multiple floors in the building and on each floor there are multiple apartments, floors can have zero or no apartment. Each apartment can be rented at most by one customer but customer can rent out multiple apartments from same building, apartment can be available on multiple floors or same. At the end of month a recipe is generated against which a rent is deposited

The entities and attributes can be derived in the following way:

Entities & Attributes:

Entity	Attributes
Building	Building Name (bname), Address, Phone Number
Apartment	Apartment ID (aid), Covered Area, Status (available/rented), Rent
Customer	Customer ID (cid), Name, Contact, CNIC
Floor	Floor Number, Number of Apartments
Receipt	Receipt ID (rid), Apartment Number, Date of Receipt, Amount

IIIII. Generating ERD



JJJJJ. Generating Physical Data Model

In ERD it's not possible to show foreign keys as a rule but it is possible in Physical Data Model to show attributes and foreign keys.

As rule of thumb primary key of parent table is referred as foreign key in child table in One-to-Many relationship.

In Many-to-Many a new third table is introduced where primary key of both tables are written as FK and made part of CPK.

In One-to-One any one table will contain FK but FK will be made as UK as rule.

Physical Data model is last step towards before database Implementation.

KKKKK. Physical Data Model of ERD

- Building(bname, address)
- Floor (floor #,no_apt, bname)
  Assumption:
  Floor# can be repeated among multiple building due to this bname is made part of Composite primary key along with Floor # in floor table

- Customer (cid, name, address, contact#)
- Apartment(aid, status, floor#, bname, CA, cid)
- Receipt (rid, aid, cid,rent,dor)

B. Example 02: Scenario and Generating ERD

Consider the following scenario:

Scenario # 2:

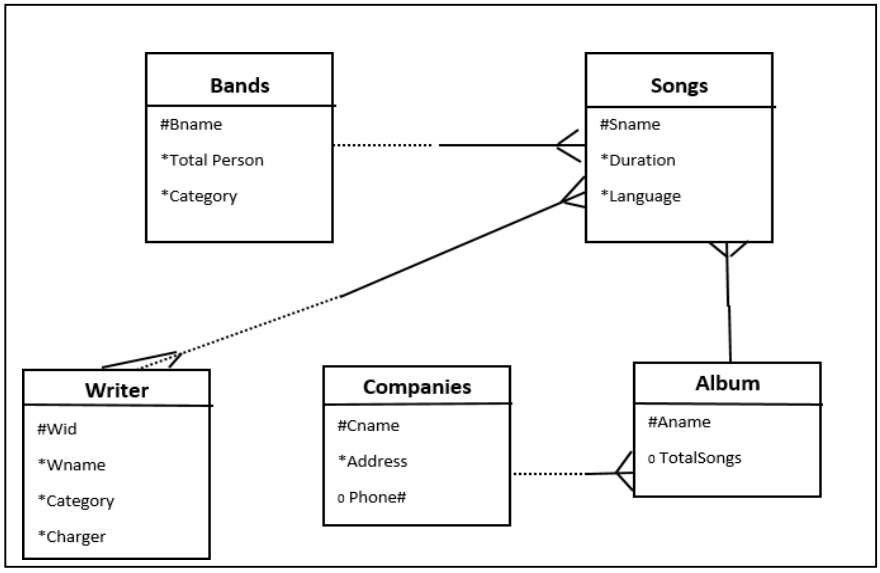
There are musical bands which record songs and request musical companies to launch their songs in the form of Album. Songs are written by song writers. Album can contain at least one song to be album and max of 12 songs. Writer can write multiple songs but songs are usually written irrespective of the number of person in the bands.

The entities and attributes can be derived in the following way:

Entities & Attributes:

Entity	Attributes
Bands	bname,total person,cat
Songs	sname, duration,language
Writer	wid, wname, charges, category
Company	cname, address, phone#
Album	aname, total_songs

LLLLL. Generating ERD of the Scenario



MMMMM. Physical Data Model of ERD:

- Band (bname,totalperson,cat)
- Company(cname, address, phone#)
- Songs(sname,duration,language,bname,aname)
- Writer(wid, wname, charges, category)
- Album(aname, total_songs,cname)
- Writer_song(wid,sname)

C. Example 03: Scenario and Generating ERD

Consider the following scenario:

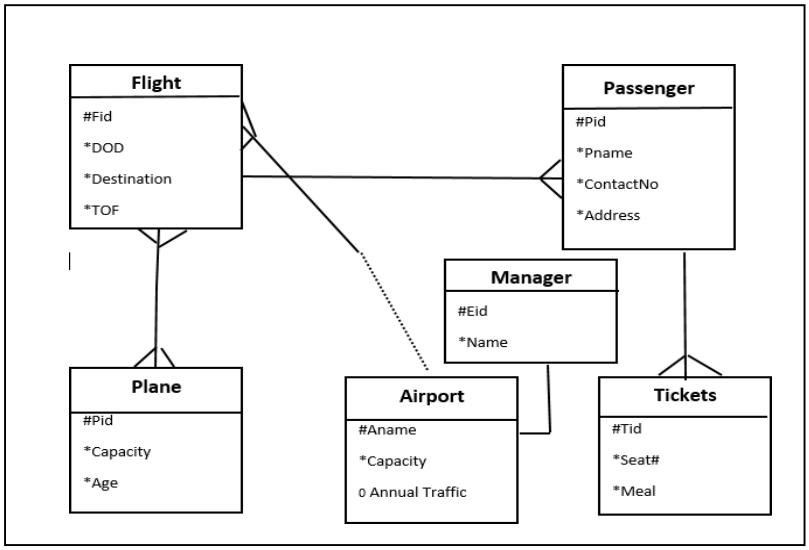
Scenario:

In airport management system, there are several flights which are either landing or taking off from Airport. There is more than one passenger on a particular flight. Passengers are allocated seat # when they purchase ticket. Plane can be associated to different flights during its lifecycle. Passenger can order different meals during flight and details are mentioned on the ticket in the form of either Asian or Non-Asian. Airport is managed by a manager.

Entities & Attributes:

Entity	Attributes
Airport	Aname, capacity, annual traffic
Manager	Employee ID (eid) ,Airport Name (aname),name
Flight	Flight ID (fid), Date of Departure (dod), Time of Dep. (tod) ,destination,aname
Passenger	Passenger ID (pid),pname,fid,contact,address,date
Ticket	Flight ID (fid), Ticket ID (tid), Passenger ID (pid),seat number,meal
Plane	Plane ID (pid),capacity,age

NNNNN. Generating ERD of the Scenario



OOOOO. Generating Physical Model from ERD

- Airport (aname, capacity, annual traffic)
- Manager (eid,aname,name)- **aname is FK and UK due to One-to-One relationship with Manager.**

- Flight (fid,dod,tod,destination,aname)
- Passenger(pid,pname,fid,contact,address,date)
- Ticket(fid,tid,pid,seat#,meal)
- Plane(pid,capacity,age)
- Plane_flight(pid,fid) – **Resolving Many-to-Many between flight and Plane**

Module 12: Anomalies

A. What is Anomaly?

Anomalies can be defined as the problems that can occur due to poor planning and designing of the databases. This usually occurs when a single table is created instead of creating multiple tables. An anomaly is an irregularity, or something which deviates from the expected or normal state.

PPPPP. Why Anomalies are Critical

As mentioned above, anomalies are critical as it can affect the end user experience. Besides, duplication or redundancy in the data is increased due to anomalies which in results create other serious issues. Anomalies may not have a huge impact on the small databases but a database with a large data size may have serious performance issues caused by anomalies. There are three types of anomalies which are explained below.

QQQQQ. Insert Anomaly

Insert anomaly occurs when certain data cannot be entered without entering unwanted data. In other words, in other words, some attributes cannot be entered without the presence of some other attributes. This results from poor design decision of storing data in just one table instead of multiple tables. As in the example below, if we want to enter a new product (pid, pname, price) then we have to add a customer (cid) against the product which might not be possible at the time of entering the product meaning to add a wanted row (product) we have to enter an unwanted (customer) row because CID is primary key and it can't be left null.

		Product			
Pid	**Pname**	**Price**	**CID**	**Cname**	**Address**
1	HD	1500	1	Ahmed	12-K, DHA
2	RAM	2500	1	Ahmed	12-K, DHA
3	MotherBoard	12000	3	Kashif	M/2-Cavalry
4	Mouse	400	2	Ali	183//A-1, Gulberg
5	LCD	20000	3	Kashif	M/2-Cavalry

RRRRR. Delete Anomaly

This anomaly occurs when the data is lost due to the deletion of some other data. The reasons of this anomaly are same as mentioned above i.e. poor design decision. Due to this anomaly certain attributes are lost due to the deletion of other attributes. If we want to delete pid : 4 then we will be deleting cid=2 also because when we delete complete row is deleted and there is only record for cid=2 meaning we might end up losing customer data when we delete product data which is a loss of unwanted data.

Figure 7: Delete Anomaly

Product					
Pid	**Pname**	**Price**	**CID**	**Cname**	**Address**
1	HD	1500	1	Ahmed	12-K, DHA
2	RAM	2500	1	Ahmed	12-K, DHA
3	MotherBoard	12000	3	Kashif	M/2-Cavalry
4	Mouse	400	2	Ali	183//A-1, Gulberg
5	LCD	20000	3	Kashif	M/2-Cavalry

SSSSS. Update Anomaly

An Update Anomaly exists when one or more instances of duplicated data are updated, but not all. An update anomaly occurs when the same data item has to be updated more than once. This can lead to errors and inconsistency of data. The same information can be expressed on multiple rows; therefore all the instances must be updates. If this is not done properly, update anomaly arises. If there are multiple records and we fail to update all the instances then it will result in data-inconsistency across database which will result in wrong reporting of data which is not acceptable.

Product					
Pid	**Pname**	**Price**	**CID**	**Cname**	**Address**
1	HD	1500	1	Ahmed	12-K, DHA
2	RAM	2500	1	Ahmed	12-K, DHA
3	MotherBoard	12000	3	Mohsin	M/2-Cavalry
4	Mouse	400	2	Ali	183//A-1, Gulberg
5	LCD	20000	3	Kashif	M/2-Cavalry

In the table given above if we decide to update Cname of cid=3 and we fail to update all the rows (cname) then it will end up in wrong information. Records with yellow color are showing that cid=3 is having different Cname and system will report you both Cname against CID=3 and its wrong. It is result that all the records are not updated.

Module 13: Normalization

A. Normalization Basics

Database normalization is the process of storing the data efficiently in order to reduce data redundancy and undesirable characteristics from the system. The two main objectives of the database normalization are to reduce redundancy and to make sure the relationships / dependencies are logical. That's why the related data is stored together. This process serves as a solution to database anomalies as normalization is a method to remove all these anomalies and bring the database to a consistent state.

TTTTT. Normalization through Reports

Normalization can be done through physical data model or through reports. In the course we will discuss Normalization through reports primarily due the reason that it will give a clear structure of the whole process.

The process of normalization consists of three steps called Normal Forms and these are as under:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)

UUUUU. First Normal Form (1NF)

First normal form (1NF) sets the very basic rules for an organized database: As per First Normal Form, no two Rows of data must contain repeating group of information i.e. each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique. These rules can be summarized as following:

- Data should be organized in tables
- There should be no repeating groups
- There should be Primary Keys in each tables.
- Primary key non-repeating group should be written with every group following the first non-repeating

Student

Sid	Sname	Address	Total_Courses-Reg	Status
1	Ali	12-k	4	Full-time

Course

Cid	Cname	Credithrs
1	DB	3
2	OOP	4
3	Web Eng	4

Semester

Semid	Start Date	End Date
Fall-15	10-Oct-15	26-Feb-16
Spring-16	15-Mar-16	30-Jun-16

group as FK.

- First group has to be non-repeating group

VVVVV. Definition of Non- Repeating group:

Group in which there is only one row in the given scenario – In the example given above student is a non-repeating group as there is only one row given in the report.

WWWWW. Definition of Repeating group:

Group in which there are more than one row in the given scenario – In the example given above course and semester are non-repeating group as there is only one row given in the report.

L. Example of First Normal Form (1NF)

Based on the figure 24 above, the following 1NF is as below entities and attributes can be defined:

Primary key of first group which has to be Non-repeating group – Student group; has to be written as FK with every other following group – Course and Semester. Just like One-to-Many primary key of Parent table is written as FK in child table, student is a parent table and Course & Semester table are child table.

First Normal Form (1NF)	
Entities	**Attributes**
Student	Sid, Sname, Total_Registered_Courses, Status
Course	Cid, Cname, Credithrs, Sid
Semester	Semid, Sid, Start Date, End Date

The data presented in the above format fulfill all the criteria of being in first normal form.

YYYYY. Second Normal Form (2NF)

2NF is derived from 1NF which means that the prerequisite to perform 2NF is 1NF. This rules states that here must not be any partial dependency of any non- key column on primary key.The objective of the second normal form is to take data that is only partly dependent on the primary key and enters that data into another table.

ZZZZZ. Example of Second Normal Form (2NF)

Considering again the data presented in the figure 24 above, the following conditions needs to be fulfilled:

- First Normal Form (1NF) is achieved
- There should be no partial dependency
- Each non-key attribute should be fully-functional dependent on Key

First Normal Form (1NF)	
TableName	**Attributes**
Student	Sid, Sname, Total_Registered_Courses, Status
Course	Cid, Cname, Credithrs, Sid
Semester	Semid, Sid, Start Date, End Date

Second Normal Form (2NF)		
TableName	**Attributes**	**Comments**
Student	Sid, Sname, Total_Registered_Courses, Status	No Change. Table with single PK is already in 2NF.
Course	(Cid, Cname, Credithrs, Sid)	Cid ⇒ Cname & Cid ⇒ Credit Hour because to find out correct value of cname &credithr only value of CID is required mean if we don't know the value of SID even then correct value of cname, credithr can found – Non-key attributes (cname, credithrs) are dependent on part of Composite key – only CID ; this is partial dependency, **course table is not in 2NF**
Course	cid, sname, credits	New Table (update of course table) **– In 2NF now**
Student-course	cid, id	Original table – Course is renamed to student_course and cid, sid are mention. **– In 2NF now**
Semester	Semid, Sid, Start Date, End Date	Semid⇒ start date, end date… Both have functional dependency on part of CPK i-e semid only. To find out correct start and end date **only** value of Semid is required.**Semester table is not in 2NF**
Semester_info	(semid, startdate, enddate)	New Table - **– In 2NF now**
Semester	(semid, sid)	Original table - **– In 2NF now**

AAAAAA. Third Normal Form (3NF)

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. This is known as transitive dependency. So this transitive dependency should be removed from the table and also the table must be in Second Normal form.

BBBBBB. Example of Third Normal Form (3NF)

Continuing with the same example, the third normal form would be like:

Note: First Normal Form (1NF) and Second Normal Form (2NF) are shown above.

Third Normal Form (3NF)		
TableName	**Attributes**	**Comments**
Course	(cid, sname, credits)	No Change
Student-course	(cid, id)	No Change
Semester	(semid, sid)	No Change
Semester_info	(semid, startdate, enddate)	No Change
Student	Sid, Sname, Total_Registered_Courses, Status	Total Course Registered ⇒ Status. Both are non key attributes and have transitive dependency because if value of total_course_registered is known then we can tell the value of status – Part-time or full-time student but if value of status of is known then correct value of total_course_registered is not known. For example if total_course_registered is 4 then status is full-time but if status is full-time it is not possible to tell whether total_course_registered is 4 or 5 or 6 , in this cause status is dependent on total_course_registered and both are non-key . This relation is not in 3NF
Student-Status	Total_courses_registered, status	New Table
Student	sid,sname, reg_courses	New Table (update of Student table)

After applying the normalization up to third normal form (3NF) the data would be in the following shape:

TableName	**Attributes**
Course	(cid, sname, credits)
Student-course	(cid, id)
Semester	(semid, sid)
Semester_info	(semid, startdate, enddate)
Student-Status	Total_courses_registered, status
Student	sid,sname, reg_courses

CCCCCC. Second Example of Normalization

Consider the data presented in the figure 25 on right having the following entities:

- Flight
- Passenger
- Airport

The attributes of each entity are also presented in the figure. The

Flight

Flight#	Destination	Origin	Distance	DOD	TOD
PK-124	Lahore	London	500	18-Sep-15	1700hrs

Passenger

Pid	Pname	Ticket#	Seat#	SeatType	Cost_ticket
1	Aamir	982	9B	Window	7000
2	Kashif	932	8A	Isle	6000

Airport

AID	Aname	Gate#	Runway#
1	Heathrow	12-B	7-A
2	Dubai	2-C	3-B

normalization of the data will occur as follows:

DDDDDD. Non-repeating group:

i. Flight

EEEEEE. Repeating Groups:

i. Passenger
ii. Airport

First Normal Form (1NF)	
Table Name	**Attributes**
Flight	Flight#, destination, origin, distance, dod, tod
Passenger	Flight#,pid, pname, seat#, ticket#, cost
Airport	aid, Flight#,, gate#, aname, runway#

Second Normal Form (2NF)		
Entities	**Attributes**	**Comments**
Flight	Flight#, destination, origin, distance, dod, tod	No Change because there is single attribute Primary key.
Passenger	Flight#,pid, pname, seat#, ticket#, cost	No Change because each non-key is dependent full-functionally on Flight# and PID
Airport	aid, Flight#,, gate#, aname, runway#	Aid⇒gate#, aname, runway#, they have partial dependency on CPK (aid, flight#). As per rule following tables will be created in the place of airport table
Airport-Info	aid,gate#,aname,runway#	New Table
Airport	aid, fid	New Table

Third Normal Form (3NF)		
Entities	**Attributes**	**Comments**
Flight	Flight#, destination, origin, distance, dod, tod	Origin, Destination->distance, this is transitive dependency because to find out the correct value of distance values of origin, destination should be known and all three attributes are non-key.
Org-Dest	origin, destination,distance	New Table
Flight	Flight#,, origin, destination, dod,tod	New Table
Passenger	Flight#,pid, pname, seat#, ticket#, cost	No Change
Airport-Info	aid,gate#,aname,runway#	No Change
Airport	aid, fid	No Change

After applying the normalization up to third normal form (3NF) the data would be in the following shape:

TableName	Attributes
Flight	Flight#, destination, origin, distance, dod, tod
Org-Dest	origin, destination,distance
Flight	Flight#,, origin, destination, dod,tod
Passenger	Flight#,pid, pname, seat#, ticket#, cost
Airport-Info	aid,gate#,aname,runway#

Module 14: Denormalization

A. What is Denormalization?

Denormalization is the process of systematically adding redundancy in a database with the purpose of improving the database performance. The tables are merged into each other in order to group data as the more tables you have, the more joins you have to perform in your queries, and joins have a negative impact on performance and more joins mean more time is required to retrieve data because data is required from multiple table. The purpose of denormalization is to reduce the running time of, the time it takes the database management system (DBMS) to calculate the results.

FFFFFF. Need for Denormalization

Denormalization comes in as a solution if the running time of a database is of more importance than the database structure. In a large database, there are cases which require data from multiple tables in a single query and the application may need to perform this type of query hundreds of time in a minute. In such situation, a fully normalized database may be unacceptably slow. In such denormalization of database increase the performance of the database. So denormalization is a tradeoff between database structure and running time.

GGGGGG. Normalization vs. Denormalization

The difference between normalization and denormalization is in the focus of these two activities. Normalization focus is to decrease the data redundancy in a database while on the other side the purpose or the focus of the denormalization is to decrease the running time of a database. As normalization reduce inconsistency, the database designer needs to ensure that the denormalized database does not become inconsistent.

HHHHHH. When to Denormalize?

Denormalization does not have any fix rule like there are rules in normalization. It is totally scenario or need based activity. As mentioned above, when the requirement is to reduce the running time of a database system, denormalization is done. It should be performed only if performance issues indicate that it is needed.

VI. Scenarios for Denormalization

a. Storing End Date

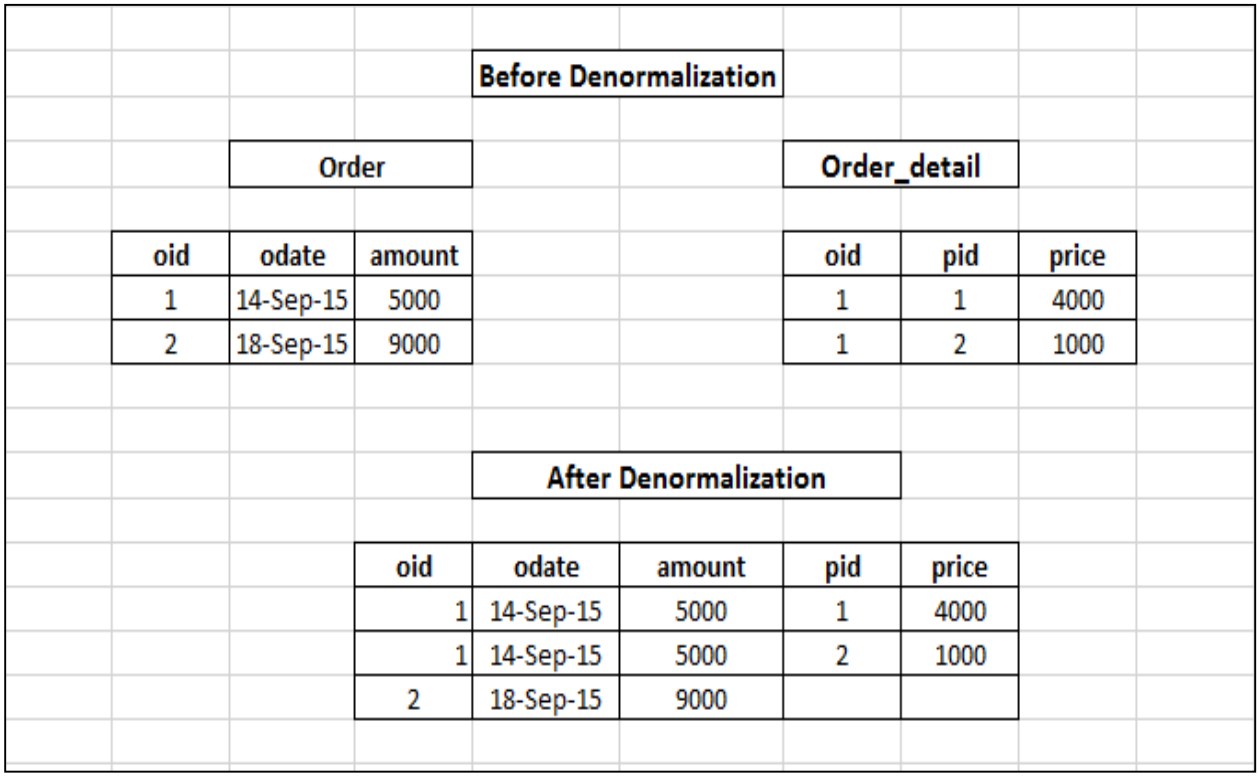
The most common denormalization decision is to store the end date for periods that are consecutive; then the end date for a period can be derived from the start date of the previous period. It is appropriate when queries are needed from tables with long lists or records that are historical and user is interested in the most current record. Add an end date column to speed up queries so that they can use a between operator which is one of the efficient operator. Between is one of the most efficient operator that can be used to retrieve the data and it require start and end range.

b. Storing Details in Master Table

In a situation where the number of detail record in child table per master record are fixed (or has a fixed maximum) and where usually all detail records are queried with the master, you may consider adding the detail columns to the master table. This denormalization works best when the numbers of records in the detail table are small. This way you will reduce the number of joins during queries. The child table will be removed and it will be merged with the parent table. As this denormalization will occur, an anomaly will be created.

c. Example of Storing Details in Master Table

The phenomenon of storing details in the master table can be illustrated with the help of the following figure:



Before Denormalization

Order

oid	odate	amount
1	14-Sep-15	5000
2	18-Sep-15	9000

Order_detail

oid	pid	price
1	1	4000
1	2	1000

After Denormalization

oid	odate	amount	pid	price
1	14-Sep-15	5000	1	4000
1	14-Sep-15	5000	2	1000
2	18-Sep-15	9000		

Considering the above image, the normalization of the database resulted in two separate tables for Order and Order Details as shown in the figure. In order to speed up the query processing time, these two tables will be merged together i.e. denormalization. In this way no joins will be required, saving the space and speeding up the query processing time and ultimately improving the database performance.

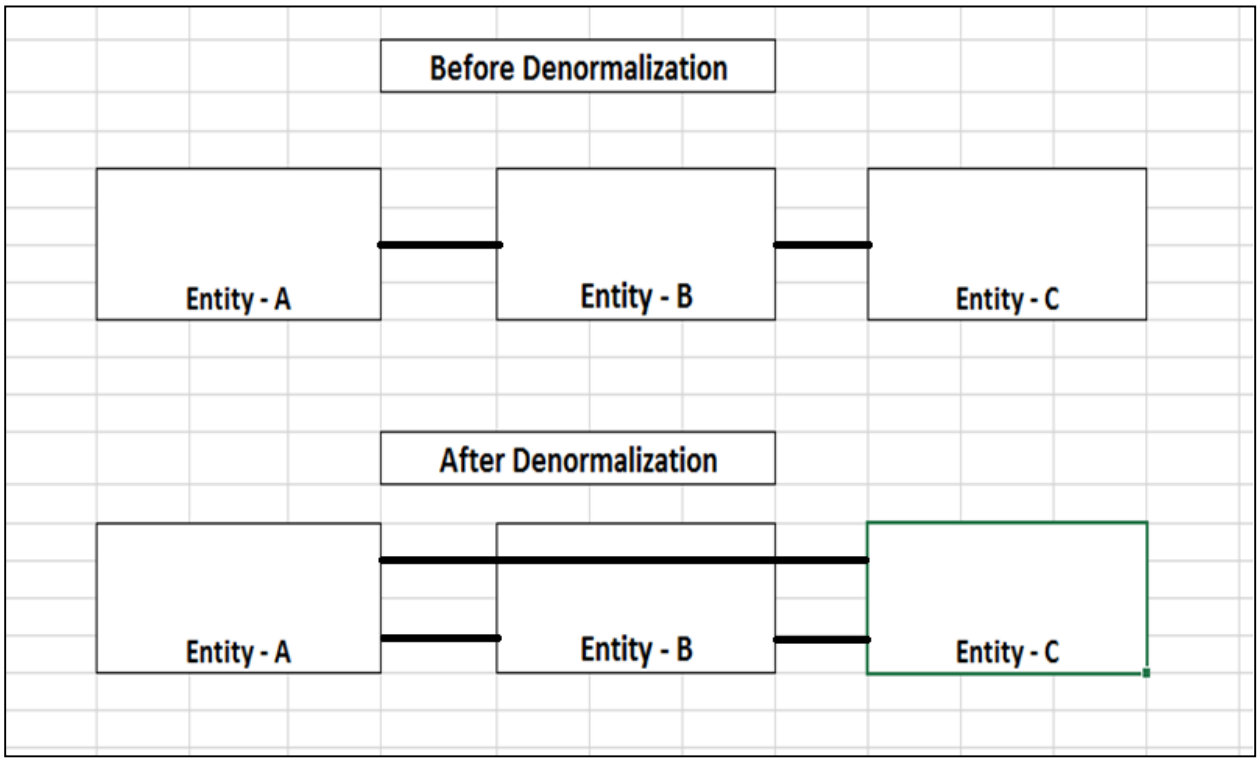
d. Short Circuit Key

The short circuit key ensures the relationships between the required entities. In a database that contains multiple levels of the master table and there is a need to query the lowest and highest level records only, consider creating short-circuit keys. These new foreign key definitions directly link the lowest level detail records to higher level grandparent records. The result can

produce fewer table joins when queries execute. It is appropriate when queries frequently require values from a grandparent and grandchild, but not from the parent.

e. Example of Short Circuit Key

As per the following example, before denormalization of the database, all the entities have parent-child relationship but no grandparent-grandchild relationship exist. In certain scenario where grandparent-grandchild is required to process the query, short circuit key comes in as a solution. As shown in the figure below, after denormalization, grandparent-grandchild also exist in the database. To achieve this, extra foreign keys will be required.



f. Current Indicator Column

This type of denormalization can be used in similar situations to the end date column technique. It can even be used in addition to an end date. It is a very common type of denormalization. Suppose most of the queries are to find the most current detail record. With this type of requirement, you could consider adding a new column to the details table to represent the currently active record. In simple words frequently updated column should have last-updated timestamp and a status column in added to show updated or old record.

g. Example of Current Indicator Column

Consider a table where the stock price of the company is being recorded. The stock price changes very frequently and the database also records the last time of update. But as the records changes frequently, so getting the latest stock price is difficult without having a status column in the table. This is where current indicator column is added (as a result of denormalization) with the motive of having status of all records. This column will tell whether a specific record / stock price are latest one or not. This whole scenario is illustrated in the figure below:

Before Denormalization

Cid	Cname	Stockprice	Last-updated
1	ABC	98	9:15
1	ABC	97.7	9:19
1	ABC	98.2	9:13

After DeNormalization

Cid	Cname	Stockprice	Last-updated	Status
1	ABC	98	9:15	Old
2	ABC	97.7	9:19	Latest
1	ABC	98.2	9:13	Old

h. Storing Calculated Value

When a calculation is frequently executed during queries, it can be worthwhile storing the results of the calculation. Queries that required calculation require more time to execute and to reduce this running time calculated fields should be pre-stored in the table. Through this type of denormalization the calculation does not need to be performed during a query and the source values do not need to be looked up every time the calculated value is required. As a result, query processing time will be shortened.

i. Example of Storing Calculated Value

Consider a database keeping record of cars with engine number, fuel tank size and average fuel consumption per liter. A query requiring total kilometers that can be traveled will require calculation which will ultimately increase the processing time. After denormalization, a column is added calculating the number of kilometers that a car can travel as per its fuel capacity. In this way, the calculation is pre stored and the calculation will not be required during query processing, reducing running time. This example is illustrated in the figure below:

Before DeNomralization

CarName	Engine#	TankSize in Liter	Average per Liter
City	93848293	40	17
Civic	9483823	45	15

After DeNormalization

				Calculated Field
CarName	Engine#	TankSize in Liter	Average per Liter	Total KM Travel
City	93848293	40	17	680
Civic	9483823	45	15	675

Module 15: Introduction to Oracle 11g on Cloud

B. Introduction

It's a cloud service hosted by Oracle with full access to the features and operations that are available with Oracle Database, but Oracle hosts the VM and cloud storage. You can perform all database management and development operations—without purchasing and maintaining hardware, without knowing backup and recovery commands, and without having to perform such complex tasks as database software upgrades and patching.

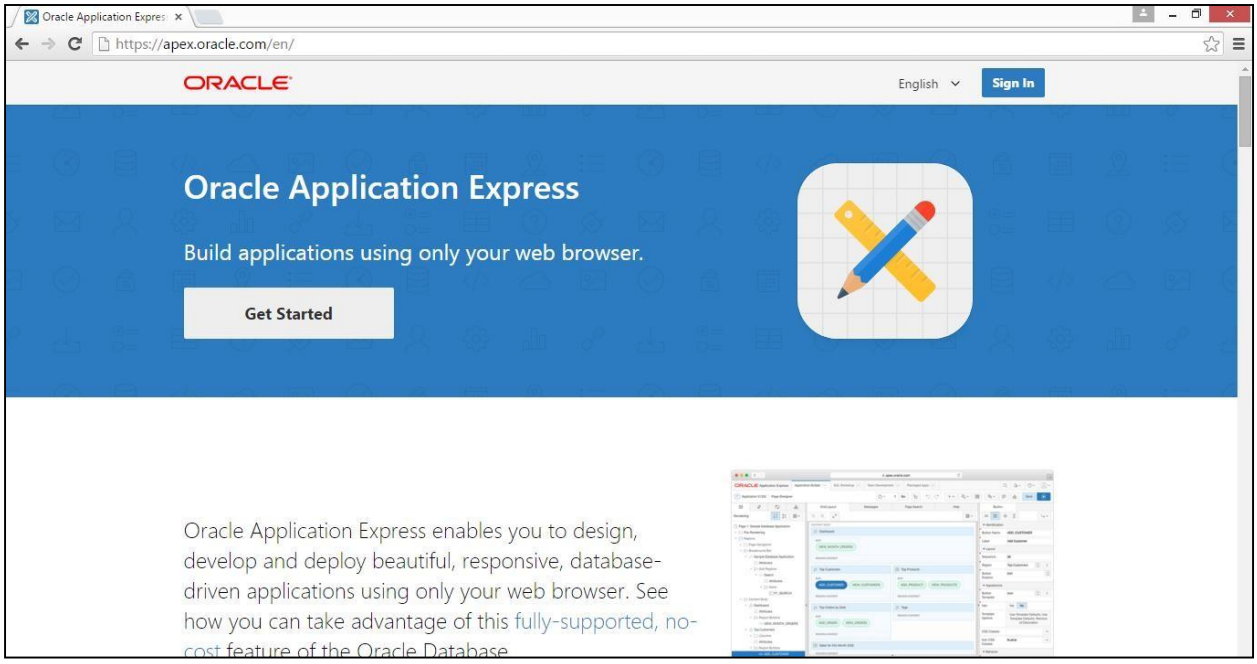
C. Login to Cloud

Login to the cloud will require the following steps:

Step 01:

Follow the following address to access Oracle 11g:

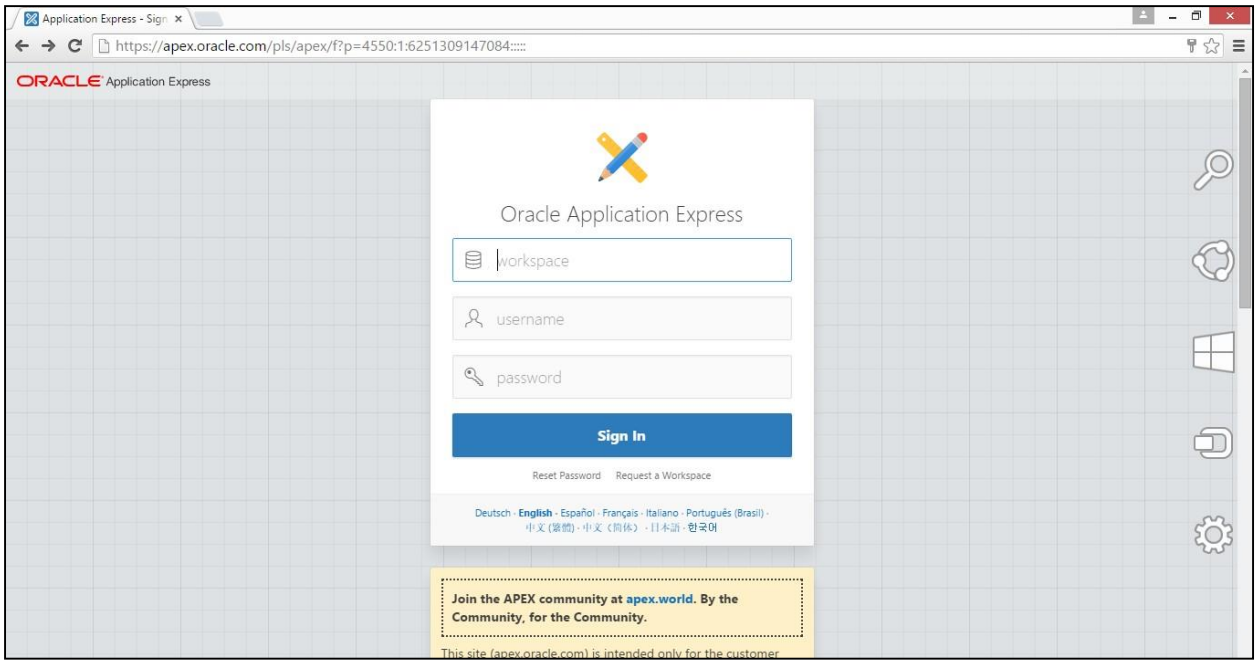
https://apex.oracle.com/en/



Step 02:

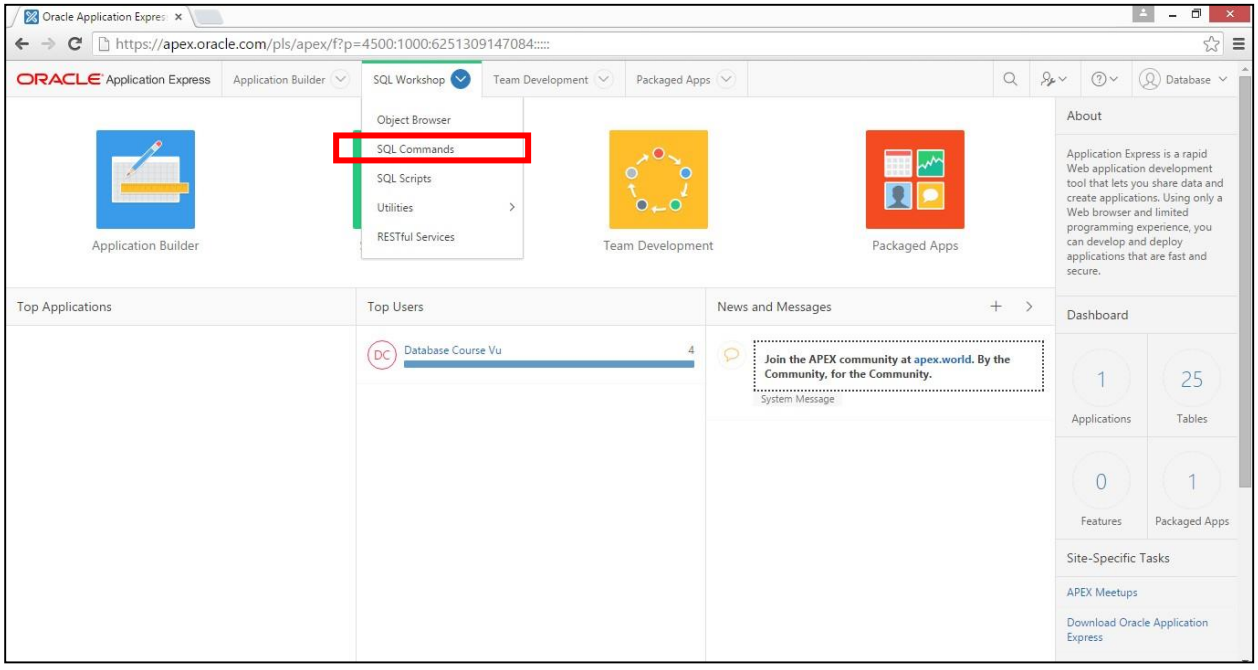
Click on the Sign In button on the upper right corner of the page and enter the following credentials:

Workspace Name	
Username	
Password	



Step 03:

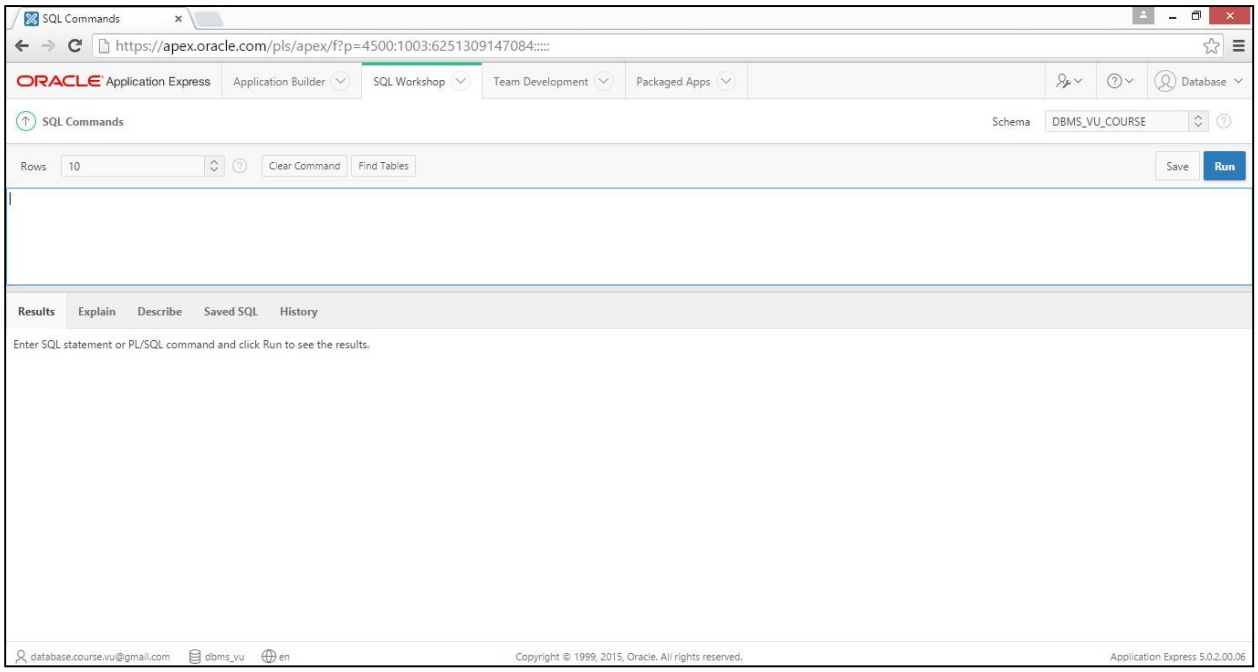
- Click on the SQL Workshop tab at the top-mid of the page



Step 04:

- Click on SQL Commands tab from the drop down list to access the code or enter SQL Statements / Commands and click Run to see the results

- Click on History Tab to access the saved code



Module 16: Using Data Definition Language (DDL) in Oracle 11g

Data Definition Language (DDL) is used to create database object in the Database. Database Object has Data dictionary created and managed by System in the Database. Table is the most important database important which is required to storage and retrieve data. DDL include Create, Alter and Drop statements.

A. Syntax of DDL

General syntax of Create Table statement is as follow:

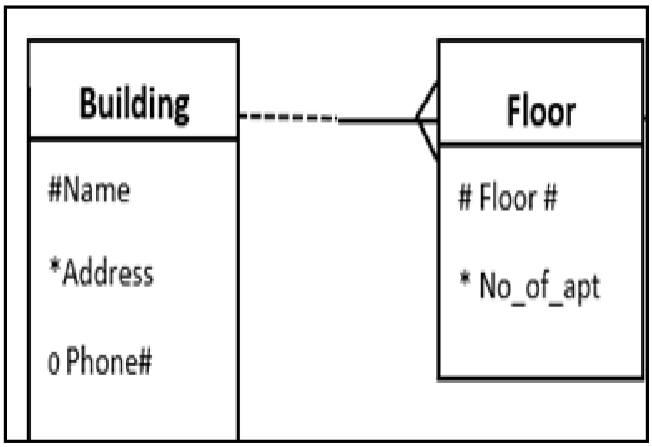
Create Table TableName (Column datatype [null | not null], ….)

DataType:

i. Number
ii. VarChar2 - Variable Length Character
iii. Date – DD/Mon/YYYY

B. Creating Single Table

To implement the single table (building table) from the ERD given below:



DDL statement to implement Building Table:

- Create table Building (bname varchar2 (30) primary key, address varchar2 (40), phone number number (10));

C. Creating Table with Foreign Key

In floor table, bname is foreign key, Floor# and bname is Composite Primary key. Code to implement the table is as below:

- Create table Floor (floorno number (10), no_of_apt number (10), bname varchar2 (30) references building (bname), primary key (floorno, bname));

D. Alter Table Command

Alter command is used to either add new column or modify data type or size in existing table

Adding New Column in Building Table

- Alter table Building add (rent number (10));

To modify data type of phone# from number to varchar2, following code can be used:

Alter table Building modify (phoneno varchar2 (10));

E. Drop Table Command

To drop table from the database, drop command can be used. While dropping the table it is important to maintain the sequence to drop the table. All the child table should be dropped first and then parent table can be drop.

- Drop table floor;
- Drop table building;

Module 17: Using Data Manipulation Language (DML) and Data Control Language (DCL)

A. Insert Statement Syntax

The insert statement is used to add data in one or more tables in a database. The basic syntax of the insert statement is as follows:

```
INSERT into TableName
VALUES (value 1, value 2…)
```

Every column in list-of-columns must have a valid value in the corresponding position in list-of-values. Therefore, before you insert a row into a table, you must know what columns the table has, and what their valid values are. Referential Integrity constraints are implemented by Default.

JJJJJJ. Implementing Insert Statement

Consider the following relation model to illustrate the example:

Entity	Attirbutes
Building	Building Name***(bname)***, Address, Phone Number
Floor	Floor Number, Number of Apartments, Bname

Continuing with the above example, the example of insert statement would be:

```
INSERT into building
VALUES ('Test', 'VU', 3948283);
```

KKKKKK. Viewing Data from Table

To verify the insertion of data in the table following query can be used

Select * from building;

Note: Detail on how to use Select will be discussed in detail later in the course

LLLLLL. Limited Column Data Insertion

If value of limited columns of the table are known then in this case limited values can be added in the table, but value of Primary Key column can’t be left NULL.

```
INSERT into building (bname, address)
VALUES ('Test-1', 'VU -LRO');
```

MMMMMM. Update Table Syntax

Update command is used to update one or more column of a table. Following is its general syntax:

```
UPDATE TableName
SET colum1 = value, column2 = value
WHERE = condition
```

Each value must be valid for its column-name. If you include the WHERE clause, the statement updates column values only in rows that satisfy condition.

NNNNNN. Implementing Single Column Update Statement

```
UPDATEemp
SET ename='Anders'
WHERE empno=7369;
```

This will update the Employee Name to Anders of employee no: 7369

To verify the changes following command can be used:

```
SELECT * from emp;
```

OOOOOO. Implementing Multiple Column Update Statement

```
UPDATE emp
SET job ='MANAGER' ,sal = 2000
WHERE empno=7369;
```

This will update Job and Sal column of the Emp table

To verify the changes following command can be used:

```
SELECT * from emp;
```

PPPPPP. Delete Statement Syntax

The Delete statement is used to delete one or more row from a table. The deleted rows can be rollback if not committed. Being committed means that all the changes in the database has been made permanent using Commit Command. Rollback command is used to restore the changes

made in the database to the way it was before, (this only works if you haven't already used Commit command). Delete command can also be used with condition to delete a particular row.

```
DELETEFROM Table Name;
DELETEFROM Table Name WHERE condition 1;
```

QQQQQQ. Implementing Delete Statement

```
DELETEFROM empWHERE empno=7369;
```

The command below will delete all the rows from table emp:

```
DELETE FROM emp;
```

RRRRRR. Truncate Table

TRUNCATE TABLE removes all rows from a table, but the table structure and its columns, constraints, indexes, and so on remain. Removing rows with the TRUNCATE TABLE statement can be faster than removing all rows with the DELETE statement, especially if the table has numerous triggers and other dependencies.

SSSSSS. Transaction Basics

A transaction is a unit of work that is performed against a database. Transactions are units or sequences of work accomplished in a logical order, whether in a manual fashion by a user or automatically by some sort of a database program.

A transaction is the propagation of one or more changes to the database. For example, if you are creating a record or updating a record or deleting a record from the table, then you are performing transaction on the table. It is important to control transactions to ensure data integrity and to handle database errors.

TTTTTT. Commit Statement

Commit Statement is used to save the changes made in the database. The Commit Statement is the transactional command used to save changes invoked by a transaction to the database. The Commit Statement saves all transactions to the database since the last Commit or Rollback Statement. The DML statements are not auto commit and should be saved by giving the commit statement.

UUUUUU. Rollback Statement

The Rollback Statement is the transactional command used to undo transactions that have not already been saved to the database. The Rollback Statement can only be used to undo transactions since the last Commit Statement or Rollback Statement was issued. In very simple words, Used to undo the work performed by the current transaction.

Module 18: Structured Query Language (SQL) Basics

A. Structured Query Language (SQL)

SQL is a database computer language designed for the retrieval and management of data in relational database. SQL stands for Structured Query Language.SQL consists of a data definition language (DDL), data manipulation language (DML), data retrieval language (DRL) and a data control language (DCL). The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control.

XXX. SQL Syntax

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

```
SELECT DISTINCT * ColumnName FROM TableName;
```

- * mean all the columns
- ColumnName is one or more column from table
- Distinct mean unique values from column

WWWWWW. Implementing SQL

Few sample statements of SQL are as follows where the words in capital letters are SQL commands or statements and words in capital letters are table names.

The following command will display all the values from Deptno column of Emp table

```
SELECT depno FROM emp;
```

The following command will display all the unique values from Deptno column of Emp table

```
SELECT DISTINCT deptno FROM emp;
```

The following command will display values from all column with all the row mean it will display the whole table

```
SELECT * FROM emp;
```

The following command will display ename and job column from emp table

```
SELECT ename, job FROM emp;
```

XXXXXX. SQL and WHERE Clause

The WHERE clause includes a condition, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set where the condition does not evaluate to True. From a large table, required rows can be fetch by using WHERE clause which applied to each row of the table. If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

YYYYYY. Implementing SQL and WHERE Clause

Example 01: Consider a situation where you need to get list of all employees (from the employee table) who are titled as managers in the job column. The code will be as follows:

```
SELECT ename, job FROM emp
WHERE job ='MANAGER';
```

Example 02: In a situation where you need to get list of all employees (from the employee table) whose salary is not equal to 2000. The code will be as follows:

```
SELECT ename, sal FROM emp
WHERE sal<> 2000;
```

ZZZZZZ. SQL and Logical Operators

Logical operators are used to specify multiple conditions in a single WHERE clause to, say, retrieve rows based on the values in multiple columns. For this purpose AND, NOT and OR operators are used to combine two or more conditions into a compound condition to get the specific required results. These operators combine two conditions at a time to select a value for output. The details of the logical are as follows:

Logical Operators	Description	Syntax
AND	Returns TRUE if both conditions are TRUE.	SELECT * FROM EMP WHERE job='CLERK' AND deptno=10
NOT	Reverses the value of the condition	SELECT * FROM EMP WHERE NOT (job IS NULL)
OR	Returns TRUE if either condition is TRUE.	SELECT * FROM emp WHERE job='CLERK' OR deptno=10

The following table shows the result of comparing two conditions with Logical Operator:

Truth Table				
Condition 01	**Condition 02**	**OR**	**AND**	**Description**
TRUE	TRUE	TRUE	TRUE	If both conditions are TRUE, AND/OR will results in TRUE
TRUE	FALSE	TRUE	FALSE	If one condition is TRUE and other is FALSE, OR Operator will result TRUE, AND Operator will results FALSE
FALSE	TRUE	TRUE	FALSE	
FALSE	FALSE	FALSE	FALSE	If both conditions are FALSE, AND/OR will results in FALSE

AAAAAAA. Implementing Logical Operator 01

Consider the following scenario:

Scenario: Write a query to find out list of all employee names who are earning more than 2500 but less than 5000.

```
SELECT ename, sal FROM emp
WHERE sal> 2500 AND sal< 5000;
```

BBBBBBB. Implementing Logical Operator 02

Scenario: Write a query to find out all those employees who are working in Dept # 20 with designation of Analyst

```
SELECT * FROM emp
WHERE deptno=10 AND job='ANALYST';
```

CCCCCCC. Implementing Logical Operator 02

Consider the following scenarios:

Scenario 01: Write a query to display list of all those empno who are earning more than 400 as commission with designation of Manager

```
SELECT empno FROM emp
WHERE comm> 400 and JOB='MANAGER';
```

Scenario 02: Write a query to find out all sales man working in dept # 30 but their salary is less than 1500

```
SELECT * from emp;
SELECT * from emp WHERE JOB='SALESMAN' AND deptno=30 and sal< 1500;
```

DDDDDDD. Wildcard Characteristics in SQL

To broaden the selections of a SQL statement, two wildcard characters, the percent sign (%) and the underscore (_), can be used. The percent sign is analogous to the asterisk (*) wildcard

character used with MS-DOS. The percent sign allows for the substitution of one or more characters in a field. The underscore is similar to the MS-DOS wildcard question mark character. The underscore allows for the substitution of a single character in an expression. Wild cards are used where part of value to be searched is known not the exact value mean if pattern is known not the complete value. Wild cards are used to make regular expressions to identify patterns in the queries. There are two wild cards as below:

%: Zero or more characters

_: Exactly one character

EEEEEEE. Like Operator in SQL

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. Wildcard Operators (% and _) are used to identify the patterns. The SQL LIKE condition allows you to use wildcards to perform pattern matching. The LIKE condition is used in the WHERE clause of a SELECT, INSERT, UPDATE, or DELETE statement.

FFFFFFF. LIKE Syntax

The basic syntax of LIKE clause is as follows:

```
SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern and column_name LIKE pattern;
```

The basic Syntax of LIKE with Percentage (%):

```
SELECT FROM table_name
WHERE column LIKE 'XXXX%'
```

The basic Syntax of LIKE with Underscore (_):

```
SELECT FROM table_name
WHERE column LIKE 'XXXX_'
```

GGGGGGG. Implementing Like – 01

Consider the following Scenario:

Scenario: Write a query to display list of name of all those employees who are having either E in the name or the name should end with G with at least two characters

```
SELECT * FROM emp
WHERE ename LIKE‘%E%’ OR ename LIKE‘%-G’;
```

HHHHHHH. Implementing Like – 02

Consider the following Scenario:

Scenario: Write a query to display all information about all those employees who are having RE in the job with at least three characters in job and should be earning at least 2500 but at most 5000.

```
Select * FROMemp
WHERE job LIKE‘%RE-%’ AND sal> 2500 AND sal< 5000;
```

Practice Scenario:

- Write a query to display all information about all those employees who are at least one vowel in the ename?
- Write a query to display list of those employees who are earning more than 2000 but at most salary is 5998 and are having at least two occurrence of E in the ename?

IIIIIII. IN Operator

The IN operator is SQL, works like OR Operator, allows to Search for a value from given list of value. The basic syntax of IN Operator is shown below:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1,value2,...);
```

JJJJJJJ. Implementing IN Operator

Consider the following scenario:

Scenario: Write a query to display all information about all those employees working either as MANAGER or ANALYST or CLERK and should belong to department number 20 and should be earning at most 2500

```
SELECT * FROM emp
WHERE job IN (‘MANAGER’, ‘ANALYST’, ‘CLERK’) AND deptno = 20 AND
sal<=2500;
```

KKKKKKK. BETWEEN Operator

The BETWEEN operator selects values within a range, the values can be numbers, text, or dates. The border values of the range are included in the search. The between operator takes two values: minimum and maximum values and search between the two including the min and max values. The basic syntax of BETWEEN operator is as follows:

The general syntax of the between logical operator is:

```
Select * from table_name
Where col_name BETWEEN val1 AND val2
```

In above syntax val1 and val2 can be two numbers, characters or dates.

LLLLLLL. Implementing BETWEEN Operator

Here is the example of BETWEEN Operator:

Scenario: Write a query to display all information about all those employees who are earning at least 2975 and at most 5000 and are working in department number 20 and are reporting to employee with empno 7839

```
SELECT * FROM emp
WHERE sal BETWEEN 2975 AND 5000 AND deptno= 20 AND mgr = 7839
```

MMMMMMM. Single Row Function

The Single Row Function operates on single rows only and returns one result per row. Single-row functions are used to manipulate data items. They accept one or more arguments and return one value for each row returned by the query. Single row functions can be character functions, numeric functions, date functions, and conversion functions. These functions require one or more input arguments and operate on each row, thereby returning one output value for each row. Single row functions can be used in SELECT and WHERE statement.

NNNNNNN. Implementing Single Row Function

ROUND and TRUNC:

```
SELECTROUND(194.683,1), TRUNC(194.683,1) FROM dual;
```

LENGTH, INSTR & CONCAT:

```
SELECTLENGTH (ename), INSTR (ename, 'E'), CONCAT(ename,job) FROM emp;
```

OOOOOOO. Group or Multiple Row Function

These functions manipulate groups of rows to give one result per group of rows. Group functions compute an aggregate value based on a group of rows. The example of the group or multiple row function is given below:

```
SELECT COUNT *, sum (sal), min(sal), max (sal),Avg (sal) FROM emp;
```

PPPPPPP. GROUP BY Clause

GROUP BY clause group together similar row together to form group and the multiple row function is used with GROUP BY Clause. The SQL GROUP BY clause is used in collaboration with the SELECT statement to arrange identical data into groups. The basic syntax of GROUP BY clause is given below.

```
SELECT column1, column2
FROM  table_name
WHERE [ conditions ]
GROUP BY column1, column2
```

QQQQQQQ. Implementing GROUP BY Clause 01

Situation: What is the salary paid of each department?

Steps to Solution:

- Need to group together all rows of each department separately i-e creating groups
- Need sum of salary each department – group

```
SELECTSUM (sal), deptno FROM emp
GROUP BY deptno;
```

The complete steps are shown below:

Step-1

Deptno	Sal	Group #
10	5000	Group -1
20	1200	Group -2
30	5002	Group-3
10	2400	Group -1
10	1900	Group-1
20	2100	Group-2

Step-2

Group #	No of Rows	Group Function: Sum of Sal - Output is required by Query
1	3	5000+2400+1900 = 9300
2	2	1200+2100 = 3300
3	1	5002

The final output of the query is shown in the figure below:

Output of Query

Deptno	Sum of Salary
10	9300
20	3300
30	5002

RRRRRRR. Implementing GROUP BY Clause 02

Consider the following:

Scenario: What is average and maximum salary paid to each Job who are reporting to MGR 7839?

Solution – 1

```
SELECT avg (sal), maximum (sal) FROM emp
WHERE mgr = 7839
GROUP BY job;
```

Solution – 2

```
SELECT avg (sal), max(sal), job, mgr, sal FROM emp
WHERE mgr=7839
GROUP BY job
```

Result – Error job, mrgare not written after group by clause

SSSSSSS. HAVING Clause

The having clause, is just like the where clause, that filters the results in aggregated / grouped data. The Where clause cannot be used in the aggregated data, so SQL having clause is introduced to filter the results. The HAVING clause enables you to specify conditions that filter which group results appear in the final results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

The basic syntax is as follows:

```
SELECT column1, column2
FROM table1, table2
GROUP BY column1, column2
HAVING [ conditions ]
```

TTTTTTT. Implementing HAVING Clause – 1

Scenario: Write a query to display average salary of each department if there are at least 2 employees working in the department

Steps to Solution:

- Need to group together all rows of each department separately i-e creating groups
- Need sum of average each department – group
- Count number of employees in each group
- Display those groups in which there are more than 2 rows

Solution:

```
SELECT avg (sal) FROM emp
GROUP BY deptno
HAVING count (*) > 3;
```

UUUUUUU. Implementing HAVING Clause – 2

Scenario: Write a query to display maximum and minimum salary by each department if average salary is more than 1500 of the department and less than 3000. The employee should not be included if there is any occurrence of 'A' in the ename.

Solution:

```
SELECT max (sal), min (sal) FROM emp
WHERE ename NOT LIKE'%A%'
GROUP BY deptno
HAVING avg (sal) > 15000 AND avg (sal) < 3000
```

VVVVVVV. ORDER BY Clause

The Order by clause is used with the SQL SELECT statement to sort the results in ascending or descending order. You have to specify one or more columns for what you want to sort the table result set. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword. The basic syntax is as follows:

```
SELECT column_name,
FROM table_name
ORDER BY column_name ASC|DESC,
```

WWWWWWW. Date Handling

Date is format specific in SQL and use a specific format i.e. MM-DD-YY. Two dates can't be added, subtracted, multiply and divide.Days can be added to Date only.

```
SELECT hiredate + 30, hiredate-50, sysdate, comm, sal, deptno FROM emp WHERE
deptno=30 AND comm is NOT null OR sal< 3000;
```

XXXXXXX. Date Functions

Following are the date functions in SQL with their descriptions and syntax:

Function	Description & Syntax
MONTHS_BETWEEN	The MONTHS_BETWEEN function returns the number of months between *date1* and *date2* Syntax: MONTHS_BETWEEN(date1, date2)
ADD_MONTHS	The ADD_MONTHS function returns a date with a specified number of months added. Syntax: ADD_MONTHS(date1, number_months)
NEXT_DAY	The NEXT_DAY function returns the first weekday that is greater than a *date*. Syntax: NEXT_DAY(date, weekday)
LAST_DAY	The LAST_DAY function returns the last day of the month based on a *date* value. Syntax: LAST_DAY(date)

An example of date function can be assumed to be as follows:

```
SELECT MONTHS_BETWEEN (sysdate, hiredate)/12, ADD_MONTHS (sysdate,
hiredate), NEXT_DAY (sysdate, 'Wednesday), LAST_DAY (sysdate) FROM emp;
```

Module 19: Advance SQL

A. Cartesian Product

Cartesian product is mathematically a binary operation in which two objects or sets (or tables) are combined in an "everything in combination with everything" fashion. In SQL statement, a Cartesian product is where every row of the first table is joined with every row of the second table. Simply defining the Cartesian product, pairing of one element in set with every element of second set I called Cartesian product. The whole concept is further explained by the following example:

Example:

Suppose the following two sets:

- Set A = {2, 5, 6}
- Set B = {8, 1}

Cartesian product of the above two sets would be:

- A * B = {(2,8), (2,1), (5,8), (5,1), (6,8), (6,1)
- Total Number of Pairs = No. of Element in A * No. of Element in B
  = 3 * 2 = 6

YYYYYYY. Cartesian Product & Joins

As mentioned above, Cartesian product is the pairing of one element in one set with every element of other set. But, as the relational databases are usually normalized, combining two tables creates another table which combines the information from the both tables. This will result in duplication in the data, losing the data integrity. Joining tables will show wrong data. Go through the following example to understand the whole concept.

Example: Employee Database

Consider the following table in an employee database:

Empno	Ename
1	Imran
2	Kashif
3	Asif

Deptno	Dname
10	Sales
20	Marketing
30	Engineering

The output of a Cartesian product would be:

Output of Cartesian Product:
Total Rows = 3 * 3 = 9

In an SQL Statement:

```
SELECT empno,ename, d.deptno, dname
FROMemp e, dept;
```

The output of the product will be the total number of rows in employee table multiplied with the total number of rows in department table means every employee is working in every department which is **WRONG** information. In an equation format:

No. of Rows in EMP * No. of Rows in DEPT = 3*3 = 9

ZZZZZZZ. Join or Inner Join

Joins are required when data from multiple tables is required. The standard join operation is knows as inner join. It horizontally combines two or more tables into a single working table. An inner join is performed by logically performing the Cartesian product (generating all combinations of rows) of the tables.

A primary key field in one table can be foreign key field in another table and a join operation is used to combine tables using a common key in both tables.

Basic Join Statement (Continuing with the example of Employee Database):

```
SELECT empno,ename, d.deptno, dname
FROMemp e, dept d
WHERED.deptno=e.deptno;
```

Joins with Other Conditions:

```
SELECT empno,ename, d.deptno, dname,
ROUND (months_between(sysdate, hiredate),0), hiredate
FROM emp e, dept d
WHERED.deptno=e.deptnoANDmonths_between(sysdate, hiredate) > 410 ;
```

AAAAAAAA. Self-Join

A self-join is a query in which a table is joined (compared) to itself. Self-joins are used to compare values in a column with other values in the same column in the same table. In self-join a table is joined with itself, especially when the table has a FOREIGN KEY which references its own PRIMARY KEY. To join a table itself, means that each row of the table is combined with itself and with every other row of the table. Self joins are used in a recursive relationship. To explain that further, think of a COURSE table with columns including PREREQUISTE, COURS_NO and others. There is a recursive relationship between PREREQUSITE and COURSE_NO as PREREQUSITE is valid only if it is also a valid COURSE_NO. the basic syntax of self-join is:

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_field = b.common_field;
```

BBBBBBBB. Implementing Self-Joins

Example 01:

```
SELECT e.ename,e.empno, b.ename, b.empno
FROM emp e, emp b
WHERE e.empno=b.mgr;
```

Example 02:

Consider a Customer Table with the following attributes:

- Table: Customer (id, name, age, address, salary)

The code for the Self-Join would be:

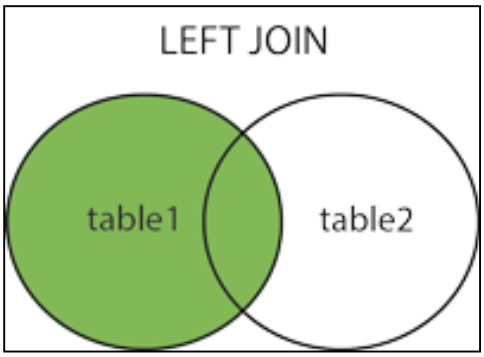
```
SELECT a.ID, b.NAME, a.SALARY
FROM CUSTOMERS a, CUSTOMERS b
WHERE a.SALARY<b.SALARY;
```

CCCCCCCC. Outer Join

An outer join extends the result of a simple join. An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists, non-matching as NULL. The SQL OUTER JOIN returns all rows from both the participating tables which satisfy the join condition along with rows which do not satisfy the join condition. The SQL OUTER JOIN operator (+) is used only on one side of the join condition. Outer joins subdivide further into left outer joins and right outer joins, depending on which table's rows are retained.

DDDDDDDD. Left Outer Join

A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table, including rows with NULL (empty) values in the link field. All the rows from the left table will be included and all the matching rows from the right table will be included. A NULL value will be displayed for a non-matching value. Simply defining, the result of a left outer join (or simply left join) for tables A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B).



EEEEEEEE. Implementing Left Outer Join

Suppose the following example to understand the concept:Suppose a table called suppliers with two fields: supplier_id and supplier_name) and another table called the order table with three fields (order_id, supplier_id and order_date). These two tables contain the following data:

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

Suppliers Table

order_id	supplier_id	order_date
500125	10000	2003/05/12
500126	10001	2003/05/13

Orders Table

If we run the following SQL statement, containing left outer join:

```
SELECT supplier.sid, supplier.sname, orders.doe
FROM supplier, orders
WHERE supplier.sid = orders.sid(+);
```

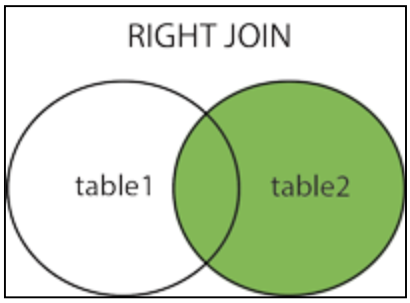
The result set will look like this:

supplier_id	supplier_name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13
10002	Microsoft	<null>
10003	NVIDIA	<null>

The rows for Microsoft and NVIDIA will be included as full outer join was used but the outer_date filed contain the NULL value for these two.

FFFFFFFF. Right Outer Join

The RIGHT JOIN keyword returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is NULL in the left side when there is no match. A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B.



GGGGGGGG. Implementing Right Outer Join

Consider the following example: again we have two tables; Supplier and Orders. The tables contain the following data.

supplier_id	supplier_name
10000	Apple
10001	Google

Suppliers Table

order_id	supplier_id	order_date
500125	10000	2013/08/12
500126	10001	2013/08/13
500127	10002	2013/08/14

Orders Table

If we run the following SQL statement, containing full outer join:

```
SELECT supplier.sid, supplier.sname, orders.doe
FROM supplier, orders
WHERE supplier.sid(+) = orders.sid;
```

The result will be:

order_id	order_date	supplier_name
500125	2013/08/12	Apple
500126	2013/08/13	Google
500127	2013/08/14	<null>

The row for 500127 (order_id) would be included because a RIGHT OUTER JOIN was used. However, you will notice that the supplier_name field for that record contains a <null> value.

HHHHHHHH. Subquery Basics

A subquery is a query within a query. Subqueries enable you to write queries that select data rows for criteria that are actually developed while the query is executing at run time. SQL subquery is usually added in the WHERE Clause of the SQL statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value in the database. Subqueries are an alternate way of returning data from multiple tables, or simply, alternative of joins. The general syntax is as follows:

```
SELECT *
FROM t1
WHERE column1 = (SELECT column1 FROM t2);
```

IIIIIIII. Implementing Subqueries

Consider the following scenarios:

Scenario 01: Write a query to display information of all those employees who are earning minimum salary.

Solution:

```
SELECT ename, sal, deptno
FROM emp
WHERE sal = (SELECT MIN (sal) FROM emp);
```

Scenario 02: Write a query to display all those deptno where minimum salary is less than average salary of all the salary among all the employee.

Solution:

```
SELECT deptno, MIN (sal)
FROM employees
GROUP BY deptno
HAVING MIN (sal) < (SELECT AVG (sal)
FROM employees)
```

Module 20: Database Views and Data Dictionaries in Oracle 11g

A. Motivation for Views

A view can be thought of as either a window of table or a virtual table. A view is a logical representation of another table or combination of tables. A view derives its data from the tables on which it is based; View does not have its own data. These tables are called base tables. Base tables might in turn be actual tables or might be views themselves. The SQL VIEW is, in essence, a virtual table that does not physically exist. Rather, it is created by a pre-defined SQL statement that joins one or more tables.

Access to data can be improved using SQL views as they Summarize data from various tables which can further be used to generate reports. Views are very powerful because they allow you to tailor the presentation of data to different types of users. While views can help to obscure large queries from users and to standardize data access, a view can also draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view and the queries can be reused to reduce time. All the DML statement can be issued against a view.

JJJJJJJJ. Syntax of a View

The following is a general syntax of the view:

```
CREATE VIEW view_name AS
SELECT columns
FROM tables
WHERE conditions;
```

KKKKKKKKK. Writing Views

Suppose we have EMP and DEPT table. To see the empno, ename, sal, deptno, department name and location we have to give a join query like this.

```
CREATE VIEW emp_det AS
SELECT e.empno, e.ename, e.sal, e.deptno, d.dname, d.loc
FROM emp e, dept d
WHERE e.deptno=d.deptno;
```

This SQL CREATE VIEW example would create a virtual table based on the result set of the select statement. You can now query the SQL VIEW as follows:

```
SELECT * FROM emp_det;
```

This will show same result as you have type the long join query. Now you can treat this EMP_DET view same as any other table.

LLLLLLLL. Writing Complex Views

WITH CHECK OPTION creates the view with the constraint that INSERT and UPDATE statements issued against the view are not allowed to create or result in rows that the view cannot select. Consider the following example:

Suppose all the employee working in Department No. 10 belongs to accounts department and most of the time you deal with these people. So every time you have to give a DML or Select statement you have to give a WHERE condition like WHERE DEPTNO=10. To avoid this, you can create a view as given below

```
CREATE VIEW accounts_staff AS
SELECT Empno, Ename, Deptno
FROM Emp
WHERE Deptno = 10
WITH CHECK OPTION CONSTRAINT ica_Accounts_cnst;
```

Now to see the account people you don't have to give a query with where condition you can just type the following query.

```
SELECT * from accounts_staff;
```

The query that defines the ACCOUNTS_STAFF view references only rows in department 10.

Considering the example above, the following INSERT statement successfully inserts a row into the EMP table through the ACCOUNTS_STAFF view:

```
INSERT INTO Accounts_staff VALUES (110, 'ASHI', 10);
```

However, the following INSERT statement is rolled back and **returns an error** because it attempts to insert a row for department number 30, which could not be selected using the ACCOUNTS_STAFF view:

```
INSERT INTO Accounts_staff VALUES (111, 'SAMI', 30);
```

MMMMMMMM. Data Dictionary Concepts

Data dictionary is a read-only set of tables that provides information about its associated database. A data dictionary contains the definitions of all schema objects in the database (tables, views, indexes, clusters, synonyms, sequences, procedures, functions, packages, trigger, and so on). A dictionary, as said above, is built in Tables which are created and maintained by System.

The data dictionary has three primary uses:

- Oracle accesses the data dictionary to find information about users, schema objects, and storage structures.
- Oracle modifies the data dictionary every time that a data definition language (DDL) statement is issued.
- Any Oracle user can use the data dictionary as a read-only reference for information about the database.

Using Data Dictionary Concepts

Few data dictionary views are mentioned in the table below:

Data Dictionary View	Description
SELECT * FROM all_tables;	ALL_TABLES describes all relational tables accessible to the user.
SELECT * FROMuser_tables;	ALL_CONSTRAINTS describes constraint definitions on tables accessible to the current user.
SELECT * FROMuser_constraints;	USER_CONSTRAINTS describes all constraint definitions on tables owned by the current user. Its columns are the same as those in "ALL_CONSTRAINTS".

A complete list of data dictionary views is available at the below address:

http://www.oracle.com/pls/tahiti/tahiti.catalog_views

Data Dictionary & Views

The views that summarize and display the information stored in the base tables of the data dictionary. These views decode the base table data into useful information. Most users are given access to the views rather than the base tables. All data dictionary information is stored in tables, but much of the data is presented to users through views. In other words, user don't get direct access to the tables of the data dictionary, they instead get access to the views, which provide somewhat limited access in order to protect the integrity of the data dictionary. Code of the View can be retrieved by using Data Dictionary using following syntax:

```
SELECT * FROM user_views;
```

Module 21: Introduction to Sequence and Synonyms with implementation in Oracle 11g

A. Motivation for sequence

Sequence is a feature supported by some database systems to produce unique auto number values on demand. Through sequences unique numbers can be generated automatically to be used in tables. A sequence is a database object from which multiple users can generate unique integers. Using a sequence generator to provide the value for a primary key in a table is an easy way to guarantee that the key value is unique. Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables.

NNNNNNNN. Syntax of Sequence

The syntax of sequence is as follows:

```
CREATE SEQUENCE sequence_name
MINVALUE value
MAXVALUE value
START WITH value
INCREMENT BY value
CACHE value;
CYCLE / No CYCLE;
```

Where:

- MINVALUE: Specify the minimum value of the sequence.
- MAXVALUE: Specify the maximum value the sequence can generate.
- START WITH: Specify the first sequence number to be generated.
- INCREMENT BY: Specify the interval between sequence numbers
- CACHE: Specify how many values of the sequence the database pre-allocates and keeps in memory for faster access
- CYCLE: Specify CYCLE to indicate that the sequence continues to generate values after reaching either its maximum or minimum value. After an ascending sequence reaches its maximum value, it generates its minimum value. After a descending sequence reaches its minimum, it generates its maximum value
- NO CYCLE: Specify NOCYCLE to indicate that the sequence cannot generate more values after reaching its maximum or minimum value. This is the default.

OOOOOOOO. Implementing Sequence:

The following statement creates the sequence customers_seq in the sample schema. This sequence could be used to provide customer ID numbers when rows are added to the customers table.

```
CREATE SEQUENCE customers_seq
START WITH     1000
INCREMENT BY 1
NOCACHE
NOCYCLE;
```

The first reference to customers_seq.nextval returns 1000. The second returns 1001. Each subsequent reference will return a value 1 greater than the previous reference.

PPPPPPPP. Sequence & Data Dictionary

```
SELECT * FROM user_sequences
WHERE sequence_name = 'customers_seq';
```

Retrieving value of sequence without using Data Dictionary

```
SELECT customers_seq.nextval, customers_seq.currval from dual;
```

QQQQQQQQ. Sequence & DML

Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables.Sequence numbers are generated independently of tables, so the same sequence can be used for one or for multiple tables. Data Manipulation Language (DML) commands can be used with sequence for inserting, deleting or updating data in the sequence. Here is the example:

Consider an employee database in which sequence is created with the name of mySeq to auto generate employee numbers. Statements are given below for DML commands:

To insert a row into the sequence:

```
INSERT into emp (empno) VALUES (mySeq.nextVal);
```

To drop the sequence:

```
DROP sequence mySeq;
```

Module 22: Indexes in Databases

A. Index Basics

An index is a data structure that the database uses to find records within a table more quickly. Indexes are built on one or more columns of a table; each index maintains a list of values within that field that are sorted in ascending or descending order. Rather than sorting records on the field or fields during query execution, the system can simply access the rows in order of the index. Index, in simple terms, is a database object which helps in efficient searching of the data. Every time database table is accessed, all the rows in the table are searched to find the required data which limits the database performance. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed.

An example to clarify the concept, consider the data shown in the table. In order to search the salary of Aqeel, for example, all the rows in the table will be searched to locate the salary of Aqeel. In large tables, this can tremendously increase the run time of a query. Imagine the same scenario if the data is to be located from different tables.

Searching Without Indexes

Name	Salary
Ahmed	10000
Kamran	23000
Allen	42000
Samar	43000
Asim	42993
Zahid	51000
Gates	89000
Farhan	32999
Aqeel	39200

RRRRRRRR. Searching Using Index

The index is created on one or more columns. An index is created to make it easier for locating records in a table. The index will create another table or index structure, which is not a part of the actual table. (See the tables below from the previous example to understand better)

Actual Table

Searching Without Indexes

Name	Salary
Ahmed	10000
Kamran	23000
Allen	42000
Samar	43000
Asim	42993
Zahid	51000
Gates	89000
Farhan	32999
Aqeel	39200

Index Created Table

Searching With Indexes

Name	RowID
Ahmed	3948ABCF
Allen	3818AACF
Aqeel	3948ABCD
Asim	3948ABCA
Farhan	3248ABC0
Gates	3548ABCF
Kamran	3748ABCF
Samar	3248ABCF
Zahid	3148ABCF

Note that the data is sorted alphabetically in the index table

It is like a supplement to the table and it will contain just 2 columns- the key values and a row ID/address to the actual row in the table. The index table / structure sort the data of the column in alphabetical order and the address or row ID in another column. As shown in the above tables, after the index table is created, the data has been sorted alphabetically and the row ID against all the records is also stored in that table. Now to search any record, index table can tell the row ID for each record.

SSSSSSSS. Creating Index in Oracle

Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index. The basic syntax is as follows:

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

Examples are as follows:

```
CREATE INDEX test_emp
ON emp (ename);
```

```
CREATE INDEX emp_ind
ON emp (job, sal);
```

TTTTTTTT. Indexes and Data Dictionary

Index is also a Database Object and Data Dictionary is maintained. Following are the different Data dictionaries maintained for Indexes.

It will display all the indexes created by currently logged in user schema

```
SELECT * FROM user_indexes;
```

It will display all the columns of indexes created by currently logged in user schema

```
SELECT * FROM user_IND_COLUMNS;
```

Module 23: Transaction

A. Transaction Basics

A transaction is a sequence of operations performed as a single logical unit of work, whether in a manual fashion by a user or automatically by some sort of a database program. A transaction is the propagation of one or more changes to the database. Practically, many SQL queries (DDL or DML)can be clubbed into a group and can be executed together as a part of a transaction. A logical unit of work i.e. Transaction must exhibit four properties, called the atomicity, consistency, isolation, and durability (ACID) properties, to qualify as a transaction. The details are given in the following.

UUUUUUUU. Atomicity Property of Transaction

This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. Atomicity ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure, and previous operations are rolled back to their former state. Consider the following example:

- Transaction 1 (T1): Update command to update 100 rows
- Transaction 2 (T2): 20 rows are updated
- Transaction 3 (T3): System failure
- *Result: Database should rollback updates of 20 Rows*

XL. Consistency Property of Transaction

This property states that before and after transaction database should be in consistent state and System constraints should not be violated in any case. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well. Example is as follows:

- Constraint: Only characters are allowed in ENAME column
- Transaction:
  - Start of Transaction
  - Commands
  - End of Transactions
- Results: There are values in ENAME column with _ and - character in it
- Summary: Transaction is not consistent because constraint is violated.

In the above example, during the transaction, a record is entered in the ENAME column with characters _ mentioned in it. This will result in inconsistency as the constraint is violated.

WWWWWWWW.Isolation

This property states that all the steps during the execution of transactions are done without interference. This simply means that now two transactions can interfere with each other and if another transaction wants to access same data then it should wait. n a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

XXXXXXXX. Durability

This property states that System should be capable enough to hold transactional data and if transaction fails then there should be backup and recovery process. The database should be durable enough to hold all its latest updates even if the system fails or restarts. After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

Module 24: Locks & Granularity

A. Concurrent Transaction

Concurrent execution of database transactions in a multi-user system means that any number of users can use the same database at the same time. Data is shared and accessed by multiple users simultaneously and this simultaneous access and processing of data may lead to dirty data. The uncontrolled execution of concurrent transactions in a multi-user environment can lead to various problems. The two main problems and examples of how they can occur are listed below.

YYYYYYYY. Lost Update Problem

This problem occurs when multiple transactions are accessing same data have their operations interrupted in such a way, that one transaction will access the data before the other has applied any updates. In such scenario, the second transaction will not have access to the updated data i.e. the data updated after the first transaction. The isolation property of the transaction is violated as the transaction have interfered each other. Besides the transactions were having the access to the partial results only which results in 'dirty data'. Let's take an example:

Consider two transactions to book seat(s) in a flight and the system is processing two transactions in parallel. You can see that the operations are interleaved in such a way that Transaction 2 had access to the data before Transaction 1 could reduce the seats by 1. Transaction 2's operation of reducing the seats by 2 has been overwritten, resulting in an incorrect value of available seats.

Transaction1	Transaction2	Operation	Data Value
Read Flight Information			seats = 15
	Read Flight Information		seats = 15
	Book 2 seats	seats = seats -2	
Book 1 seat		seats = seats -1	
	Write seats		seats = 13
Write seats			seats = 14

In the end multiple transitions are having different value of one share variable which is WRONG, for Transaction1 value of Seats variable is 14 and for Transaction2 is 13 but seat will have value of 14 so value which transation2 is holding is wrong

ZZZZZZZZ. Uncommitted Data

This problem occurs when one transaction updates a data item, but has not yet committed the data permanently to the database. Because of failure, the transaction is rolled back and the data item is returned to its previous value. A second transaction accesses the updated data item before it is returned to its original value. The second transaction will read the share data between failure and rollback i.e. before the changes become permanent. Again, isolation property of the transaction is violated here. This is further illustrated in the following example:

Again the example of two transactions to book seats in a flight. In this example, you can see that Transaction 2 has access to the updated data of Transaction 1 before the changes were committed permanently to the database. The transaction 1 rollbacks and this result in the dirty data as the incorrect value of the seats available will be shown.

Transaction1	Transaction2	Operation	Data Value
Read Flight Information			seats = 15
Book 1 seat		seats = seats -1	
Write seats			seats = 14 (Uncommitted)
	Read Flight Information		seats = 14
	Book 2 seats	seats = seats -2	seats = 12
	Write seats		seats = 12
Rollback			seats = 15

In the end multiple transitions are having different value of one share variable which is WRONG, for Transaction1 value of Seats variable is 12 and for Transaction2 is 15 but seat will have value of 15 so value which transation2 is holding is wrong

AAAAAAAAA. Rationale for Lock

The issues in the concurrent transaction are due the isolation property of the transaction. The issues in concurrency may lead to the dirty or unreliable data. So this pinpoints a need for a mechanism to ensure the isolation property of all the transaction. Here the locks come in as a solution.

BBBBBBBBB. Lock Basics

A lock, as a read lock or write lock, is used when multiple users need to access a database concurrently. Locks avoid multiple users to access the share data on the same time and if one transaction is in progress then other transactions must wait to access same share data. This prevents data from being corrupted or invalidated when multiple users try to read while others write to the database. A read lock can be used to prevent other users from reading a record (or page) which is being updated.

CCCCCCCCC. Granularity of Locks

The granularity of locks in a database refers to how much of the data is locked at one time. In theory, 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). The following table describes the multiple levels of locks.

Lock Level	Description
Database	• Only One Session can be created with Database • Not feasible to keep all the user in wait state • Feasible if there is major support update i-e to update Database to new version
File	File can be a single table, part of table or combination of multiple table
Table	• Entire Table is locked for particular user • Useful when change is impacting whole table.
Row	A row level lock applies to a row in a table. This is also the most commonly locking

	level, and practically all major database vendors support row level locks.
Column	Particular column or multiple columns are locked for a particular users.

DDDDDDDDD. Level of Locks

There are two levels of locks in database. These are:

1. ***Exclusive Lock:*** When a statement modifies data, its transaction holds an exclusive lock on data that prevents other transactions from accessing the data. This lock remains in place until the transaction holding the lock issues a commit or rollback. Table-level locking lowers concurrency in a multi-user system.
2. ***Shared Lock:*** When a statement reads data without making any modifications, its transaction obtains a shared lock on the data. Another transaction that tries to read the same data is permitted to read, but a transaction that tries to update the data will be prevented from doing so until the shared lock is released

Using exclusive locks the locked data can be read or processed by one user only. A request for another exclusive lock or for a shared lock is rejected, as shown in the 'Compatibility Graph'.

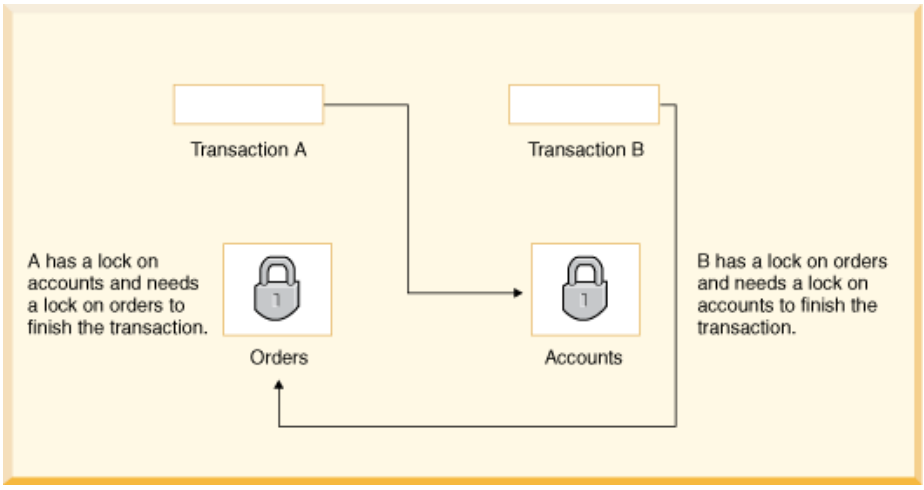
	S	X
S	true	false
X	false	false

Compatibility Graph

In shared locks, several users can read the same data at the same time, but as soon as a user edits the data, a second user can no longer access this data. Requests for further shared locks are accepted, even if they are issued by different users, but exclusive locks are rejected.

EEEEEEEEE. Deadlock

In a database, a deadlock is a situation in which two or more transactions are waiting for one another to give up locks. For example, Transaction A might hold a lock on some rows in the ACCOUNTS table and needs to update some rows in the ORDERS table to finish. Transaction B holds locks on those very rows in the ORDERS table but needs to update the rows in the ACCOUNTS table held by Transaction A. Transaction A cannot complete its transaction



because of the lock on Orders. Transaction B cannot complete its transaction because of the lock on Accounts. All activity comes to a halt and remains at a standstill forever unless the DBMS detects the deadlock and aborts one of the transactions.

FFFFFFFFF. Deadlock Prevention

Database use to avoid deadlocks increase transaction throughput and reduce system overhead. To avoid deadlocks, each lock request is inspected to identify the risk of potential deadlock. If this inspection reveals the deadlock potential, the lock is not granted. Well, very few modern DBMS's can actually prevent or avoid deadlocks, because there's a lot of overhead required in order to do so. This is because the DBMS's that do try to prevent deadlocks have to try to predict what a database user will do next. Thus, every lock request must be accepted after assessing the deadlock potential of the lock request.

GGGGGGGGG. Deadlock Detection

Instead of deadlock prevention, the more popular approach to dealing with database deadlocks is deadlock detection. There are two common approaches to deadlock detection:

1. Set the lock wait (time for lock to be released) time period to a certain preset limit (like 5 seconds). So, if a session waits more than 5 seconds for a lock to free up, then that session will be terminated.
2. Inspect all the locks currently in place to see if there are any two sessions that have locked each other out and are in a state of deadlock.

In either of the above mentioned approaches, one of the requests will have to be terminated to stop the deadlock and any transaction changes which came before the request will have to be rolled back so that the other request can make progress and finish.