Database Management System 1

Introduction to Database System

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Data & Information

Data & Information

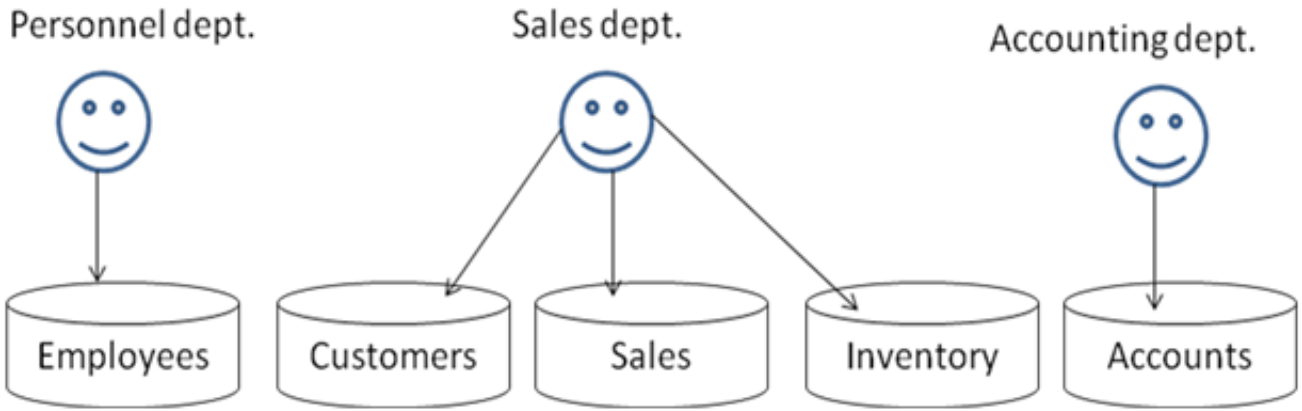
- ***Data***
  - Raw facts, unprocessed facts
  - Refers to what is actually stored
- ***Information***
  - Result of processing raw data
  - Refers to meaning of the data, understood by the user

Data management focuses on the generation, storage & retrieval of data

Limitations of File-Processing Systems

Limitations of File-Processing Systems

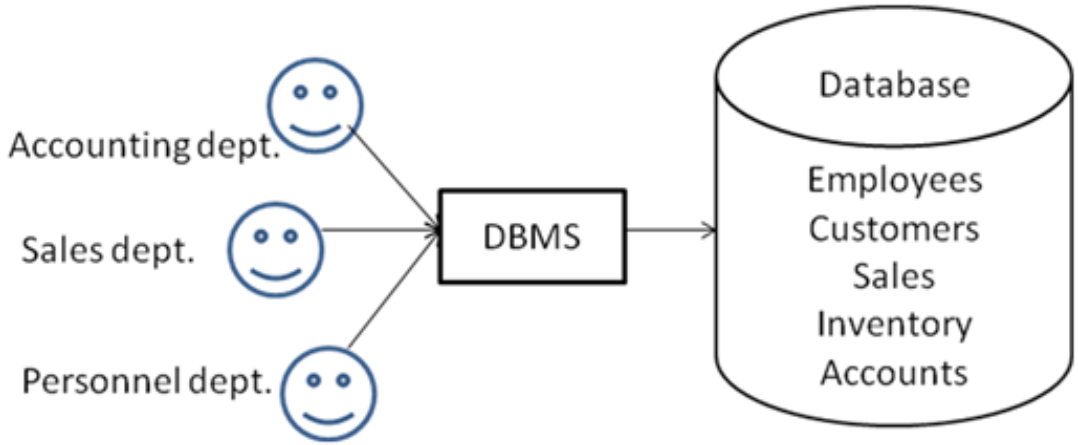
- Redundancy problem
  - Repetitive data
- Data-inconsistency problem
  - Incorrectness of data
- Lack of data integration
  - Complex and time consuming



Database

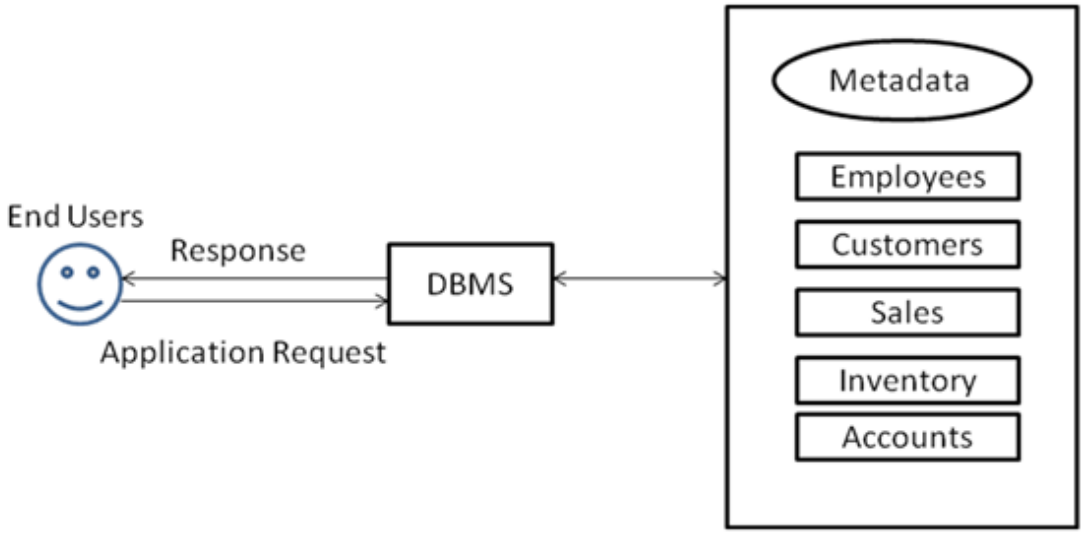
Database

- Database is a collection of interrelated data
- Database is a shared, integrated computer structure that stores:
  - **End- user data**: raw facts of interest to the end-user
  - **Meta data**: through which the end-user data are integrated & managed. The metadata provides a description of the data characteristics and the set of relationships that link the data found within the database
- Database is an organized collection of data of an organization or enterprise



DBMS

- DBMS (Database Management System) is a collection of programs that manages structure & controls access to the data stored in the database
- It includes tools to add, modify or delete data from the database, ask questions (or queries) about the data stored in the database and produce reports
- DBMS serves as the intermediary between the user & the database



Database Types

Database Types

- Depending on the number of users accessing the database, a database system may be classified as:
  - **Single-user database system**: It supports only one user at a time. When a single-user database runs on a personal computer, it is also called a desktop database system
  - **Multi-user database system**: It supports multiple users at the same time. When a multi-user database supports relatively small number of users, it is called as a workgroup database system. If the database is used by many users across globe, it is known as enterprise database system
- Depending on the location of the database, a database system may be classified as:
  - **Centralized database system**: It supports data located at a single site or single place
  - **Distributed database system**: It supports data distributed across several different sites. Here, the same database can be replicated and stored in another computer so that when ever the original server goes down; the data can be available to the user from the replicated data from other servers

Advantages of Database System

Advantages of Database System

A database system is comprised of a database, DBMS software and appropriate hardware

- *Controlling Redundancy & Inconsistency*: ensures the consistency of data and saves storage space
- *Allows Data Sharing*: allows data to be shared among different users located at different locations
- *Restricting Unauthorized Access*: controls the type of data access
- *Providing Storage Structures for efficient query processing*: provides capabilities for efficiently executing queries and updates. It must provide specialized data structures to speed up disk search for the desired record

Advantages of Database System...

Advantages of Database System...

- *Providing Backup & Recovery*: provides facilities for recovery from hardware or software failures
- *Providing multiple user interfaces*: provides a variety of user interfaces
- *Enforcing Integrity Constraints*: provides capabilities for defining and enforcing database constraints
- *Solving data isolation*: each user transaction is isolated from other's transaction on the same critical data
- *Providing economies of scaling*: easily scalable to any number of records/users

Database Management System 2

Data Model

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Data Model

Data Model

Data model is a collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints. That means a data model provides a way to describe the design of a database

- It is relatively simple representation, usually graphical, of complex real-world data structures
- Data modeling is considered as the most important part of the database design process

Data Model Basic Building Blocks

Entity

An entity is anything about which data are to be collected and stored. An entity represents a particular type of object in the real world

Entity Set

Set of entities of the same type that share the same properties are called as entity sets

Attribute

An attribute is a characteristic of an entity

Constraints

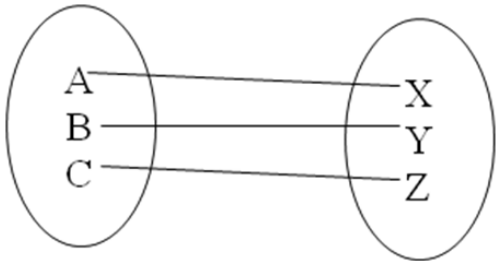
A constraint is a restriction placed on the data. Constraints are important because they help to ensure data integrity

Data Model Basic Building Blocks...

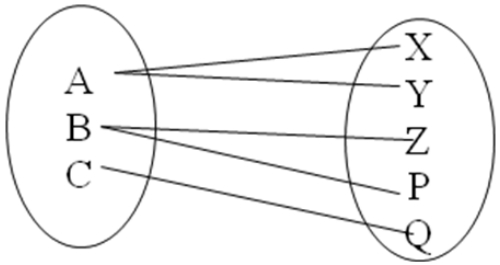
Relationship

A relationship describes an association among entities. Different types of relationship are:

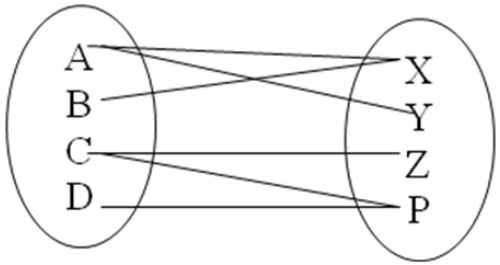
- **One-to-One (1:1) Relationship**:



- **One-to-Many (1:M) Relationship**:



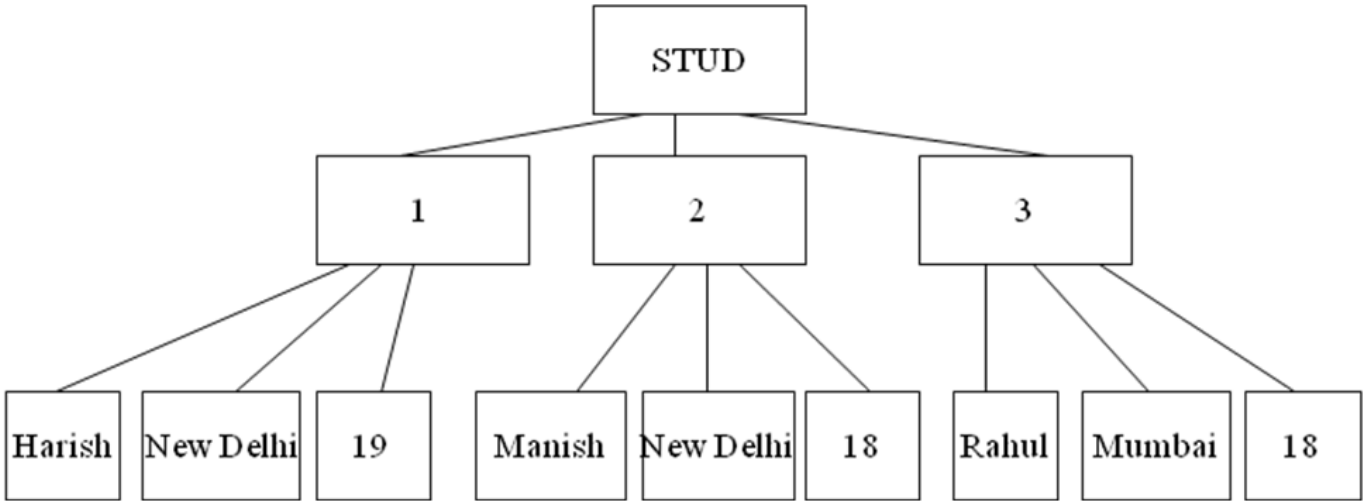
- **Many-to-Many (M:N) Relationship**:



Hierarchical Model

Hierarchical Model

The hierarchical model was developed in the 1960s to manage large amount of data for complex manufacturing projects. Its basic logical structure is represented by an *upside-down tree*. The hierarchical structure contains levels of segments. It depicts a set of 1:M relationships between a parent and its children segments



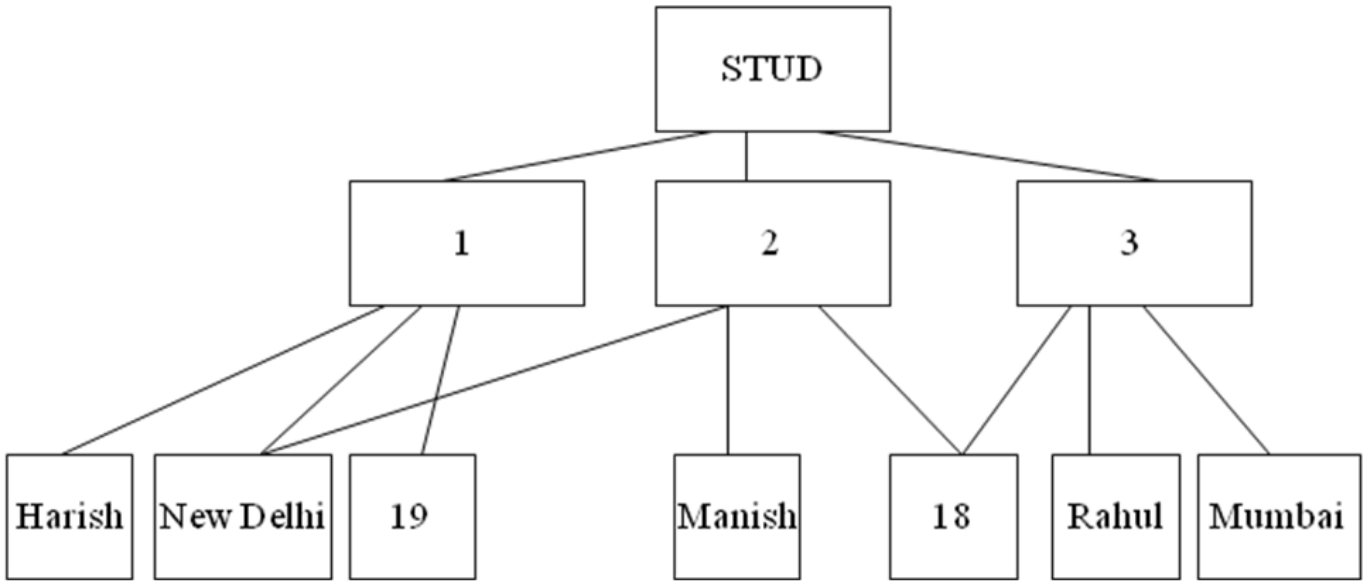
Hierarchical Model...

Advantages & Disadvantages

- **Advantages**:
  - Efficient storage for data that have a clear hierarchy
  - Parent/child relationship promotes conceptual simplicity & data integrity
  - It is efficient with 1:M relationships
  - It promotes data sharing
- **Disadvantages**:
  - It is complex to implement
  - It is difficult to manage
  - There are implementation limitations, that means it can't represent M:N relationships
  - There is no DDL and DML
  - There is lack of standards

Network Model

The network model was created to represent complex data relationships more effectively than the hierarchical model, to improve database performance, and to impose a database standard. A user perceives the network model as a collection of records in 1:M relationships



Network Model...

Advantages & Disadvantages

- **Advantages**:
  - It represents complex data relationships better than hierarchical models
  - It handles more relationship types, such as M: N and multi-parent
  - Data access is more flexible than hierarchical model
  - Improved database performance
  - It includes DDL and DML
- **Disadvantages**:
  - System complexity limits efficiency
  - Navigational system yields complex implementation and management
  - Structural changes require changes in all application programs
  - Database contains a complex array of pointers that thread through a set of records
  - Put heavy pressure on programmers due the complex structure
  - Networks can become chaotic unless planned carefully

Relational Model

Relational Model

The relational model was introduced by E. F. Codd in 1970. This data model is implemented through RDBMS; which is easier to understand and implement

The most important advantage of the RDBMS is its ability to hide the complexities of the relational model from the user. Another reason for the relational data model's rise to dominance is its powerful and flexible query language. Generally, SQL is used for this purpose

STUD

roll	name	city	age
1	Harish	New Delhi	19
2	Manish	New Delhi	18
3	Rahul	Mumbai	18

GRADE

regdno	roll	cgpa
S001	1	7.9
S002	2	8.5
s003	3	9.4

Relational Model...

Advantages & Disadvantages

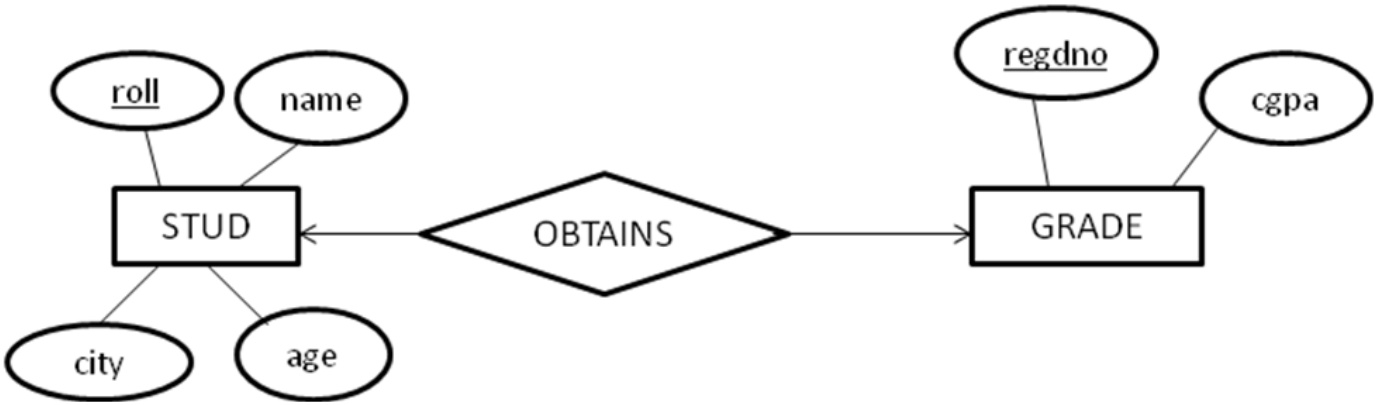
- **Advantages**:
  - Changes in a table's structure do not affect data access or application programs
  - Tabular view substantially improves conceptual simplicity, thereby promoting easier database design, implementation, management and use
  - Have referential integrity controls ensure data consistency
  - RDBMS isolates the end-users from physical level details and improves implementation and management simplicity
- **Disadvantages**:
  - Conceptual simplicity gives relatively untrained people the tools to use a good system poorly
  - It may promote *islands of information* problems as individuals and departments can easily develop their own applications

Entity-Relationship(ER) Model

Entity-Relationship(ER) Model

Peter Chen first introduced the ER data model in 1976; it was the graphical representation of entities and their relationships in a database structure that quickly became popular. Thus, the ER-model has become a widely accepted standard for data modeling

ER models are normally represented in an ER diagram



Entity-Relationship(ER) Model...

Advantages & Disadvantages

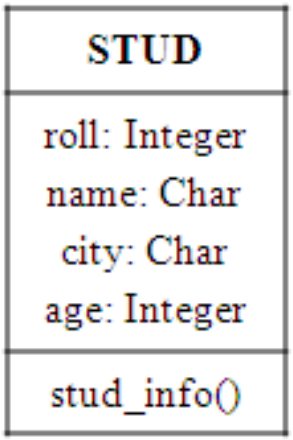
- **Advantages**:
  - Visual modeling yields exceptional conceptual simplicity
  - Visual representation makes it an effective communication tool
  - It is integrated with dominant relational model
- **Disadvantages**:
  - There is limited constraint representation
  - There is limited relationship representation
  - There is no DML
  - Loss of information content when attributes are removed from entities to avoid crowded displays

Object-Oriented(OO) Model

Object-Oriented(OO) Model

In object-oriented data model, both data and their relationships are contained in a single structure called an *object*. Like the relational model's entity, an object is described by its factual content. But quite unlike an entity, an object includes information about relationships between the facts within the object, as well as information about its relationships with other

Attributes describe the properties of an object. Objects that share similar characteristics are grouped in classes. Thus, a *class* is a collection of similar objects with shared structure (attributes) and methods



Object-Oriented(OO) Model...

Advantages & Disadvantages

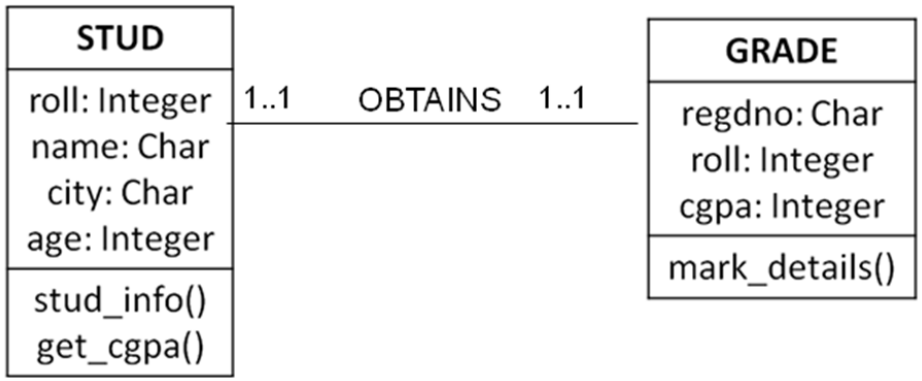
- **Advantages**:
  - Semantic content is added
  - Support for complex objects
  - Visual representation includes semantic content
  - Inheritance promotes data integrity
- **Disadvantages**:
  - It is a complex navigational system
  - High system overheads slow transactions
  - Slow development of standards caused vendors to supply their own enhancements, thus eliminating a widely accepted standard

Object-Relational(OR) Model

Object-Relational(OR) Model

The object-oriented data model is somewhat spherical in nature, allowing access to unique elements anywhere within a database structure, with extremely high performance. But, it performs extremely poorly when retrieving more than a single data item
The relational data model is best suited for retrieval of groups of data, but can also be used to access unique data items fairly efficiently

Thus, by combining the features of relational data model and object-oriented data model, object-relational data model was created



Semi-structured Model

Semi-structured Model

The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. The XML (Extensible Markup Language) is widely used to represent semi-structured data. It supports unstructured data

Database Management System 3

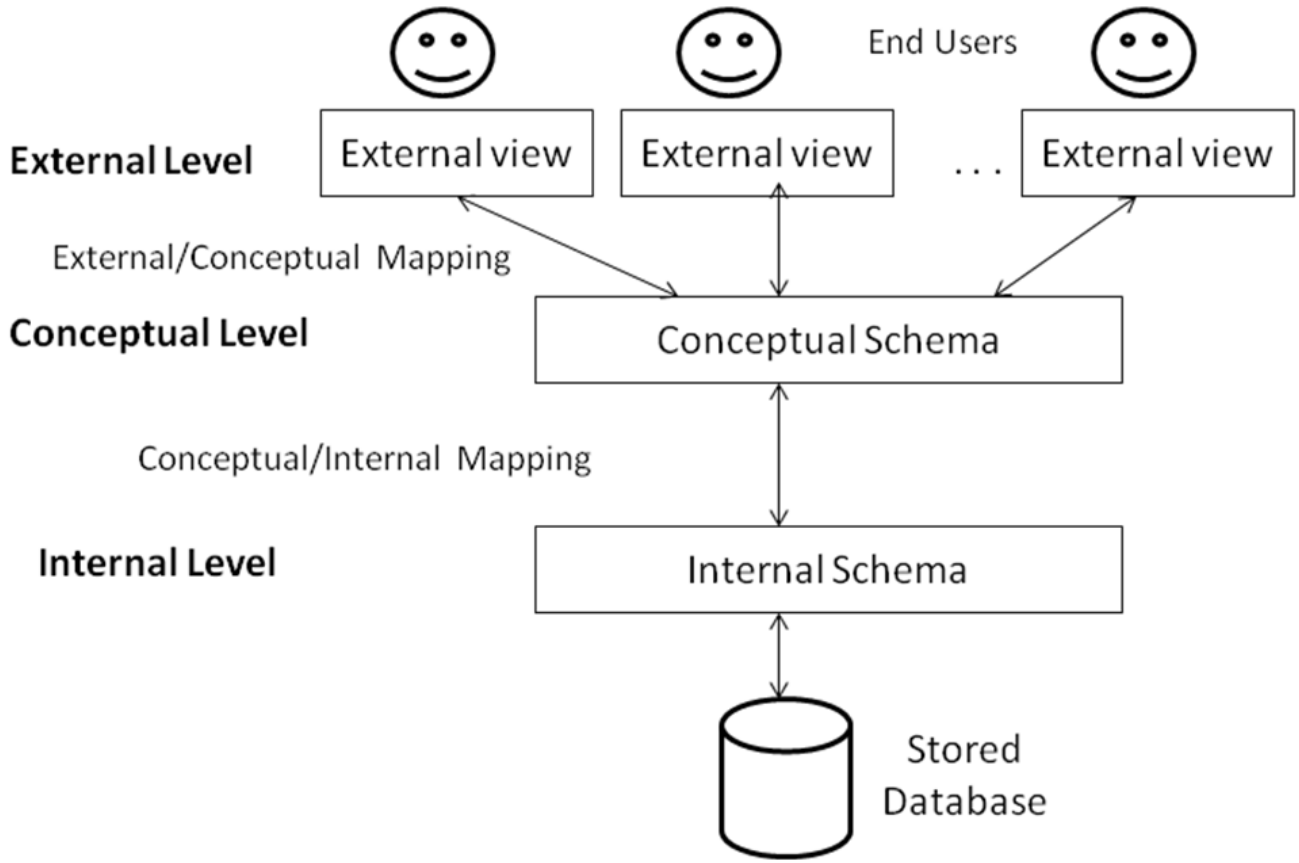
3-Level Abstraction of Database

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

3-Level Abstraction of Database

3-Level Abstraction of Database

The goal of the ANSI/SPARC 3-level abstraction is to separate the user applications and the physical database. It deals with the data, the relationship between them and the different access methods implemented on the database. The logical design of a database is called a ***schema***



3-Level Abstraction of Database...

External/View Level

The external level includes a number of external schemas or user views. Each external schema or user view describes the part of the database that a particular user group is interested in and hides the details of the database from that user group

Conceptual Level

The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships and constraints

It represents global view of the entire database. Thus; for a database, there is only one conceptual schema available

3-Level Abstraction of Database...

Internal Level

The internal level has an internal schema, which describes the physical storage structure of the database system. Like conceptual schema, there is only one internal schema available for a database. It is the one which is closest to physical storage

The internal schema not only defines the various stored record types, but also specifies what indices exist, how stored fields are represented

Mapping and Data Independence

Mapping and Data Independence

In a database system based on the 3-level architecture, each user group refers only to its own external schema. The process of transforming requests and results between different levels are called mapping

Conceptual/Internal Mapping

It defines the correspondence between the conceptual view and the stored database. ***Physical Data Independence*** indicates that the internal schema can be changed without any change to the conceptual schema

External/Conceptual Mapping

It defines the correspondence between a particular external view and the conceptual view. ***Logical Data Independence*** indicates that the conceptual schema can be changed without affecting the existing external schemas

Database Users

Different database users are:

Naive Users

They are the normal or unsophisticated users who interact with the system by invoking application programs that have been written previously. The typical user interface for naive users is a form interface, where the user can fill in appropriate fields of the form

Application Programmers

They are computer professionals who write application programs to access data from the database. Application programmers can use different tools to develop user interfaces

Database Users...

Sophisticated Users

They interact with the system without creating any application program. Rather, they form their requests in a database query language and submit each such query to a query processor. *Analysts* who submit queries to explore data in the database fall in this category

Specialized Users

They are sophisticated users who write specialized database applications that don't fit into the traditional data processing framework

Database Administrator(DBA)

The person who has central control of the whole database system is called DBA. The DBA coordinates all the activities of the database system

Database Users...

The roles of DBA are:

- DBA creates the original database schema by executing a set of DDL statements
- DBA defines and controls the access methods for the different users
- DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance
- By granting different types of authorization, DBA can regulate which parts of the database various users can access
- DBA specifies the different types of constraints to different tables or objects
- DBA is responsible for the periodically backing up the database
- DBA ensures that enough free disk space is available for normal operations and upgrading disk space as required
- DBA monitors the jobs running on the database and ensures that the performance is not degraded by very expensive tasks submitted by some users

Database Management System 4

Database Architecture

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

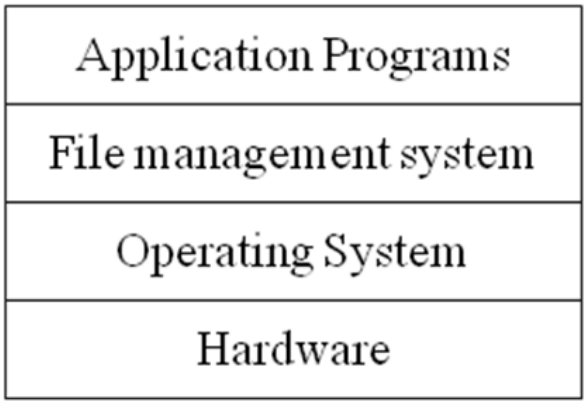
Data Storage and Querying

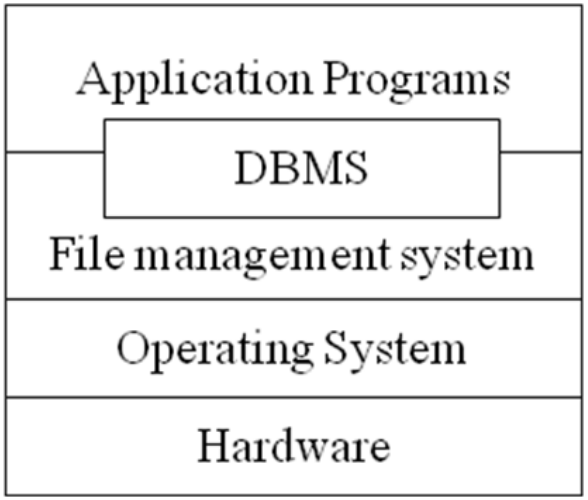
Data Storage and Querying

The functional components of a database system can be broadly divided into:

- *Storage Manager*: It is important because databases typically require a large amount of storage space
- *Query Processor*: It is important because it helps the database system simplify and facilitate access to data

The overall computer system consists of four modules as: Hardware, Operating system, File management system and Application program





Storage Manager

A storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager. Storage manager is responsible for storing, retrieving and updating data in the database
The storage manager components include:

- ***Authorization and Integrity Manager***:This module tests for the satisfaction of integrity constraints and checks the authority of users to access data
- ***Transaction Manager***:Transaction manager ensures that the database remains in a consistent (correct) state despite system failures and concurrent transaction executions proceed without conflicting
- ***File Manager***:This module manages the allocation of space on disk storage and data structures used to represent information stored on the disk

Storage Manager...

Storage Manager...

- ***Buffer Manager***:Buffer manager is responsible for fetching data from the disk storage into main memory. The buffer manager is a critical part of the database system

The storage manager implements several data structures as part of the physical system implementations:

- **Data files**:These are files in the physical memory used to store the database itself
- **Data Dictionary**:Data dictionary stores the metadata (data about data) that provides the information about the definitions of the data items and their relationships, authorizations, and usage statistics. In addition, any changes made to the physical structure of the database are automatically recorded in the data dictionary
- **Indices**:Indices are used to provide faster access to data items stored in the physical storage

Query Processor

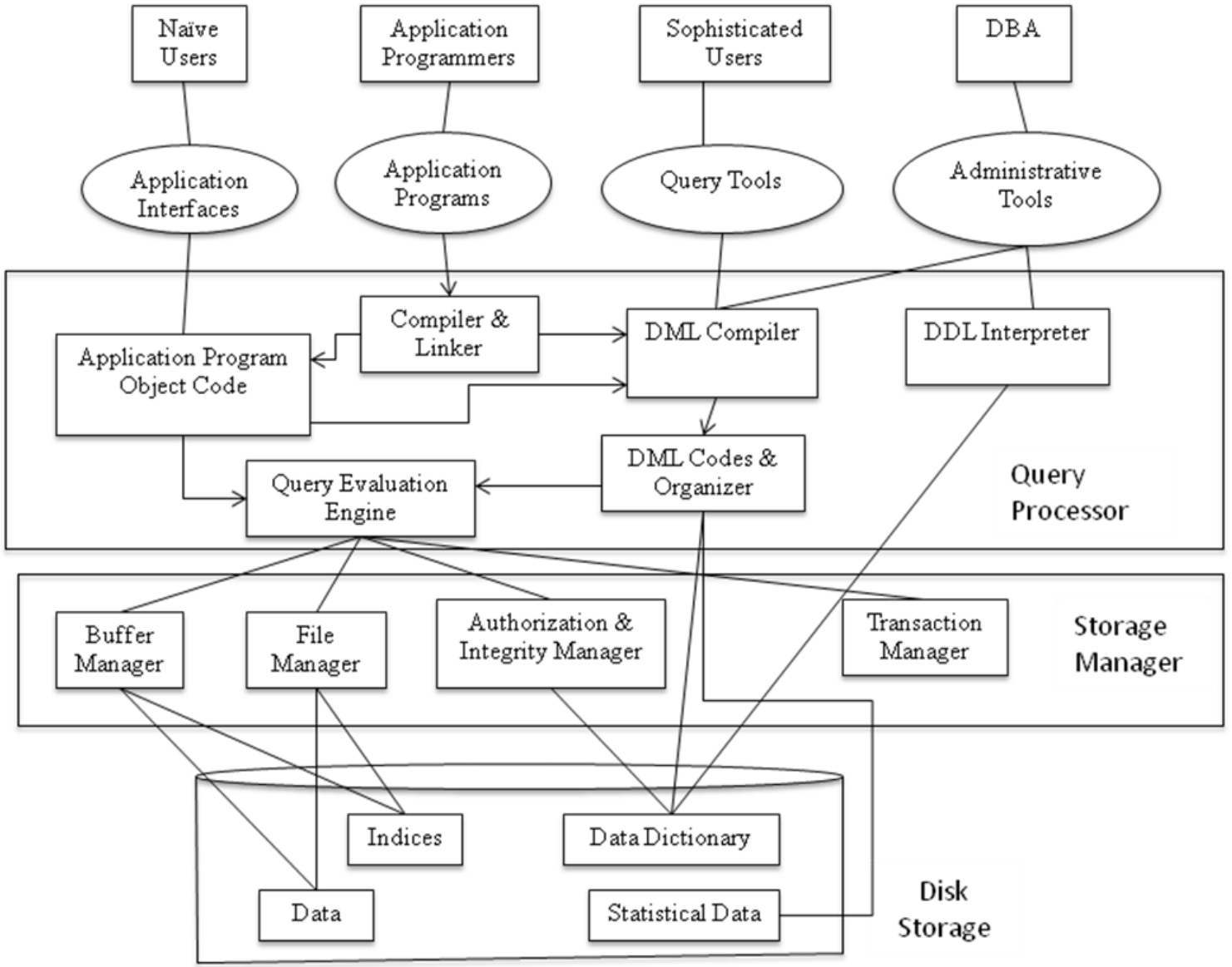
The work of query processor is to execute the query successfully

The major components of query processor include:

- ***DDL Interpreter***:This is the interpreter used to interpret DDL statements and records the definitions in the data dictionary
- ***DML Compiler***:DML compiler translates the DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. When a user wants to perform a DML operation, the data dictionary has to be checked for the validation purpose
- ***Query Evaluation Engine***:This module executes the low-level instructions generated by the DML compiler

Database Architecture

The overall database architecture is:



Application Architecture

Application Architecture

Client machines are those on which the remote database users work. Server machines are those on which the database system runs

2-Tier Architecture

- Here, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language. The two tiers are: *Data server* and *Client application*
- It consists of two layers: Client Tier and Database (Data Tier)
- The application logic is either placed inside the user interface on the client or within the database on the server
- It is a client-server architecture
- It is easy to build and maintain

Application Architecture....

3-Tier Architecture

- Here, the client machine acts as a front end and doesn't contain any direct database calls. The client end communicates with an application server, usually via a form interfaces. The application server in turn communicates with a database system to access data
- It consists of three layers: Client layer, Business layer and Data layer
- The 3-tier applications are more appropriate for large applications, and the applications that run on the web
- The application logic lives in the middle tier. It is separated from the data and user interfaces
- 3-tier systems are more scalable, robust and flexible
- They can integrate data from multiple sources
- It is complex to build and maintain
- It runs faster

Disadvantages of Database Processing

Disadvantages of Database Processing

- **Larger file size**: In order to support all the complex functions that it provides to users, DBMS occupies a great amount of disk space as well as a substantial amount of internal memory
- **Increased Complexities**: The complexity and breadth of the functions provided by a DBMS make it a complex product
- **Greater Impact of Failure**: If several users are sharing the same database, a failure on the part of any one user that damages the database in some way might affect all the other users connected
- **More difficult recovery**: The database must first be restored to the condition it was in when it was last known to be correct, any updates made by users since that time must be redone. The greater the number of users involved in updating the database, the more complicated this task becomes

Database Management System 5

ER Modeling

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Overview of the Database Design Process

Overview of the Database Design Process

- The initial phase of database design is to characterize fully the data needs of the prospective database users. It usually involves in textual description
- Next, the designer chooses a data model and, by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database. The ER model is typically used to represent the conceptual design
- The designer reviews the schema to confirm that all data requirements are satisfied and are not in conflict with one another
- At this stage of conceptual design, the designer can review the schema to ensure it meets all the functional requirements

Overview of the Database Design Process...

Overview of the Database Design Process...

- The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases:
  - In the logical design phase, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The implementation data model is typically the Relational data model
  - Finally, the designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified

Entity-Relationship(ER) Model

Entity-Relationship(ER) Model

The ER model was developed to facilitate the database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The ER model is very much useful in mapping the meaning and interactions of real-world enterprises onto a conceptual schema

Entities

An entity is a thing or object in the real world that is distinguishable from all other objects, i.e. an entity is an object of interest to the end user. The set of similar types of entities is called entity set; which is represented by a rectangle containing the entity set's name. The entity set name, a noun, is usually written in all capital letters

Entity-Relationship(ER) Model...

Attributes

Attributes are characteristics of entities. Attributes are represented by ovals and are connected to the respective entity set with lines. In the conceptual modeling, the value of an attribute comes from a domain of possible values

Relationships

In modeling, the association between entities are referred to as relationship. The relationship name is a verb. A relationship set is a set of relationships of the same type. Relationship sets are represented by diamonds and are connected to the participant entity sets

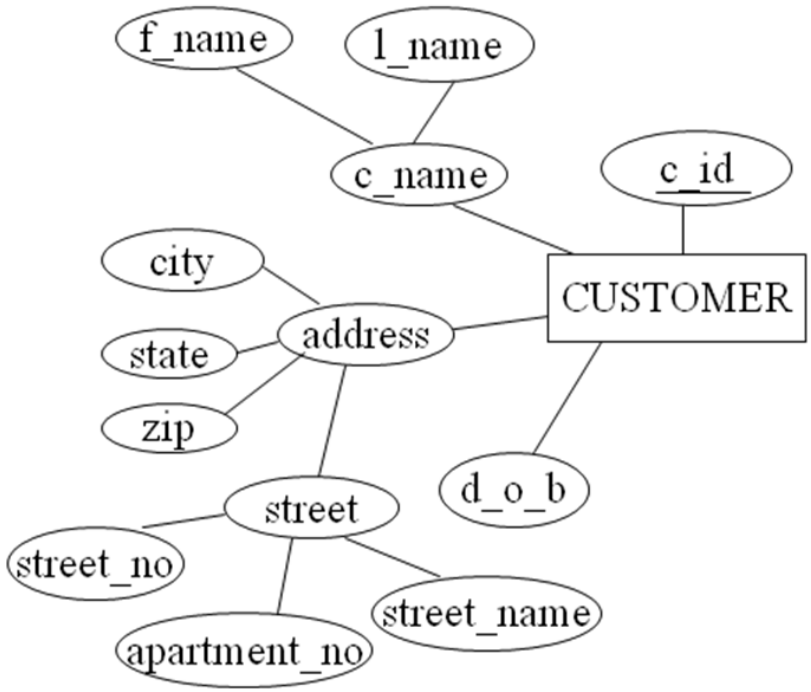
NULL Values

An attribute takes a NULL value when an entity doesn't have a value for it. The NULL values may indicate not applicable, i.e. the value doesn't exist for the entity. NULL can also designate that an attribute value is unknown. An unknown value may be either missing or not known. NULL value is an entry in all the domains

Attribute Types

Simple and Composite attributes

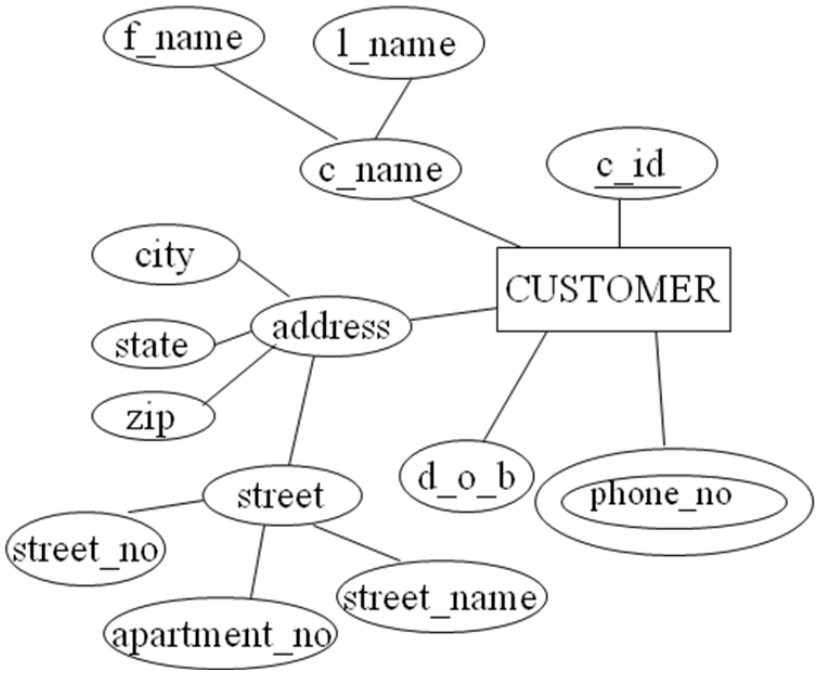
An attribute that has a discrete factual value and cannot be meaningfully subdivided is called an atomic or simple attribute.On the other hand, a composite attribute can be meaningfully subdivided into smaller subparts (i.e. simple attributes) with independent meaning



Attribute Types...

Single-valued and Multi-valued attributes

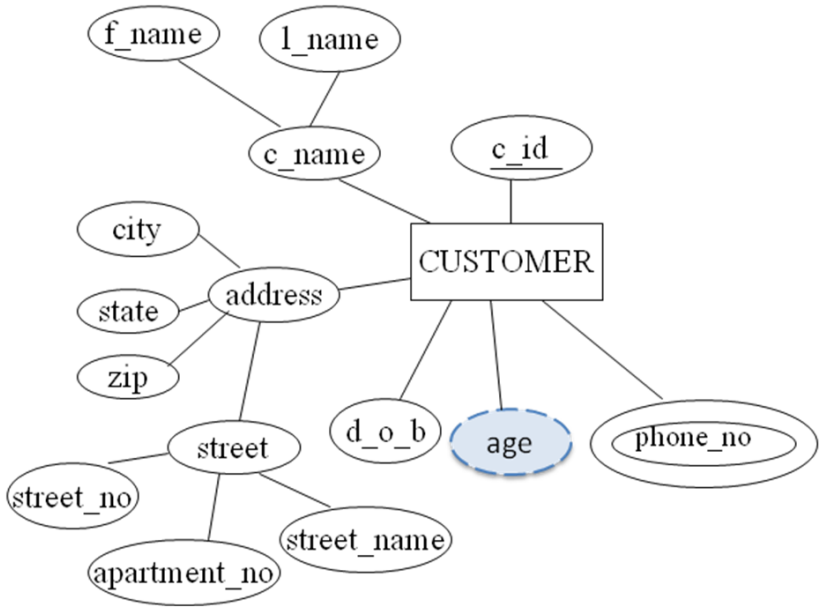
Most attributes have a single value for a particular entity and are referred to as single-valued attribute. However, attributes that can have more than one value are known as multi-valued attributes. It is represented by double oval



Attribute Types...

Stored and Derived attributes

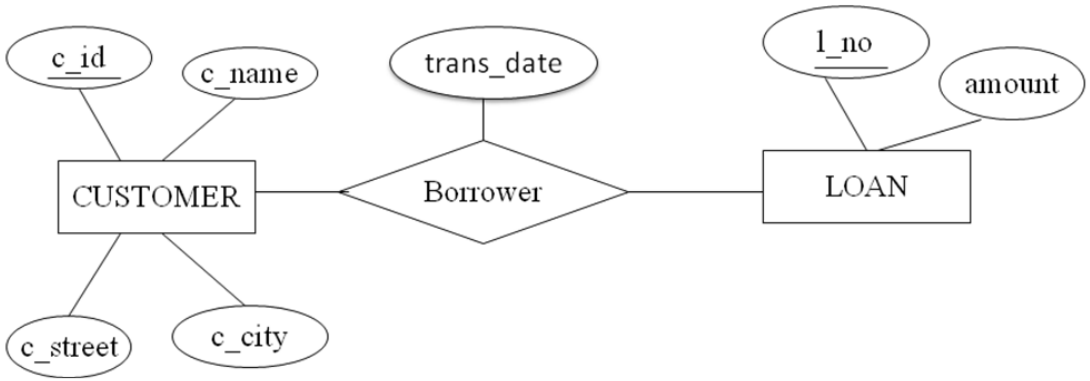
The attribute with independent existence is called as stored attribute where as the attribute, whose value is depending on other stored attribute, is called as derived attribute. The derived attribute is represented by the dotted oval



Attribute Types...

Descriptive attributes

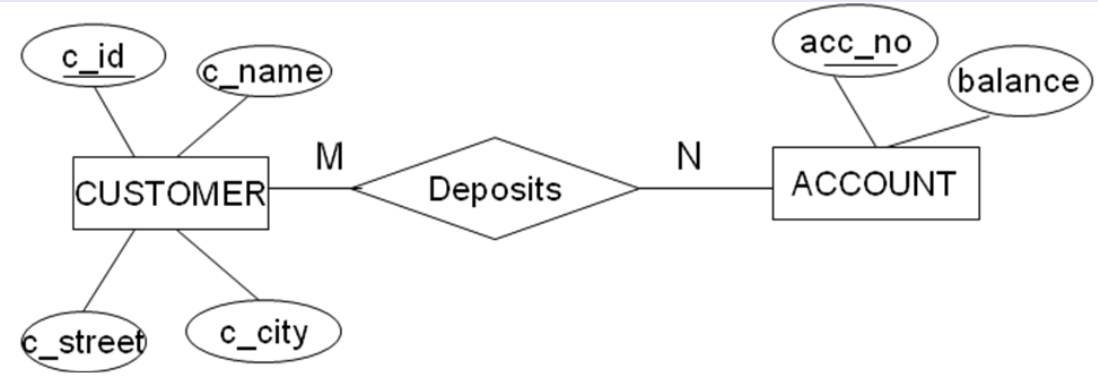
A relationship may also have attributes called descriptive attributes for representing the description about the association



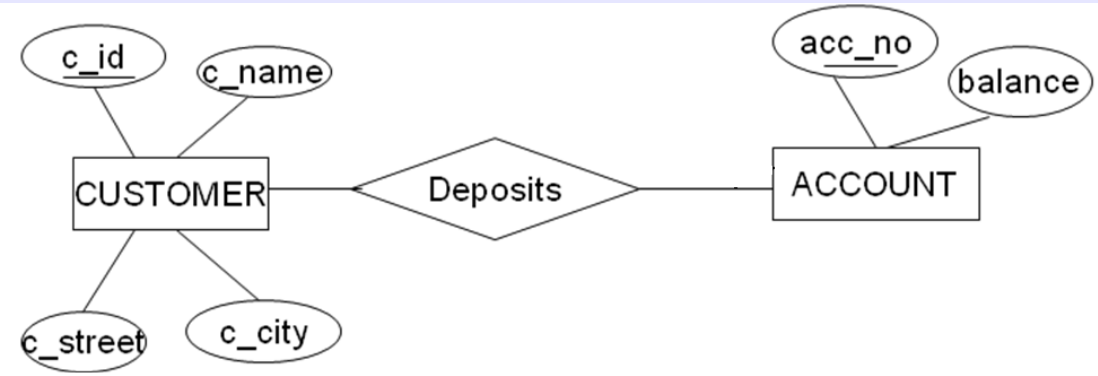
Mapping Cardinality Representation

M:N relationship (Chen Notation)

An entity in A is associated with any number (zero or more) of entities in B and vice versa



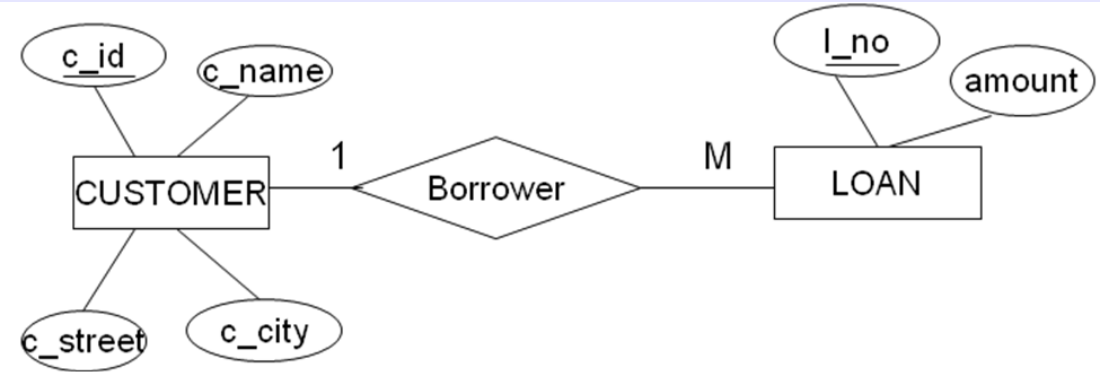
M:N relationship (Bechman Notation)



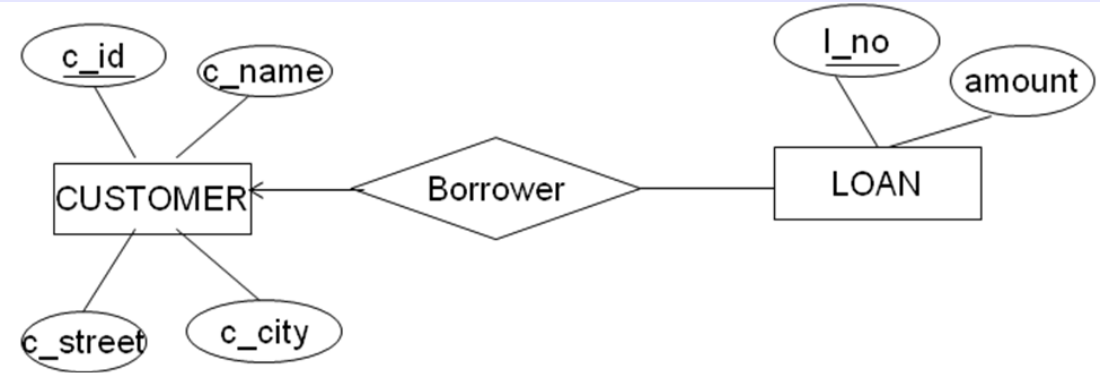
Mapping Cardinality Representation...

1:M relationship (Chen Notation)

An entity in A is associated with any number (zero or more) of entities in B; an entity in B, however, is associated with no more than 1 entity set of A



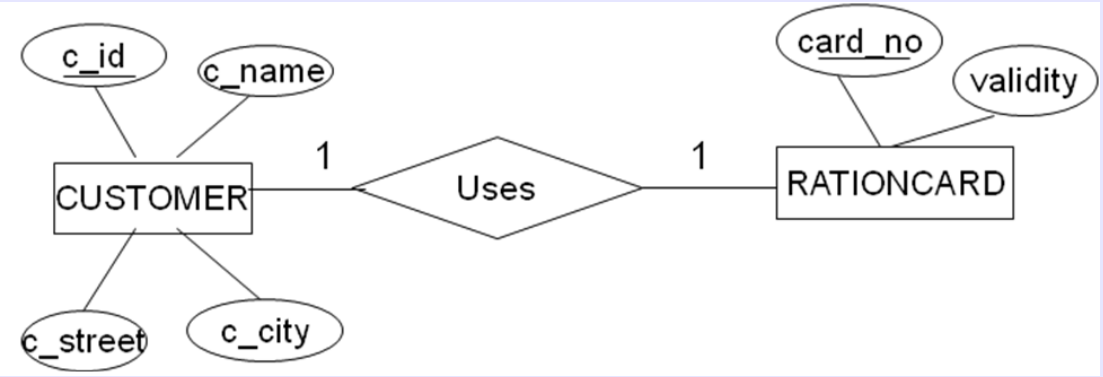
1:M relationship (Bechman Notation)



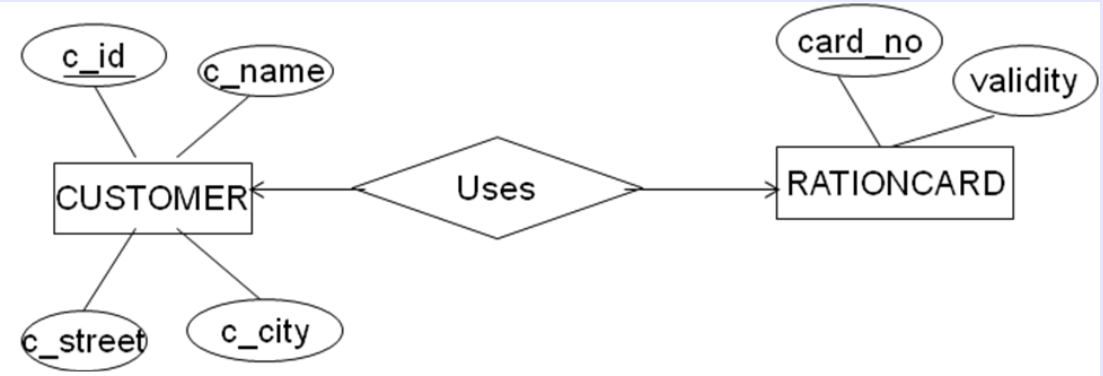
Mapping Cardinality Representation...

1:1 relationship (Chen Notation)

An entity in A is associated with no more than 1 entity of B; and an entity in B is associated with no more than 1 entity of A



1:1 relationship (Bechman Notation)



Database Management System 6

ER Modeling...

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Keys

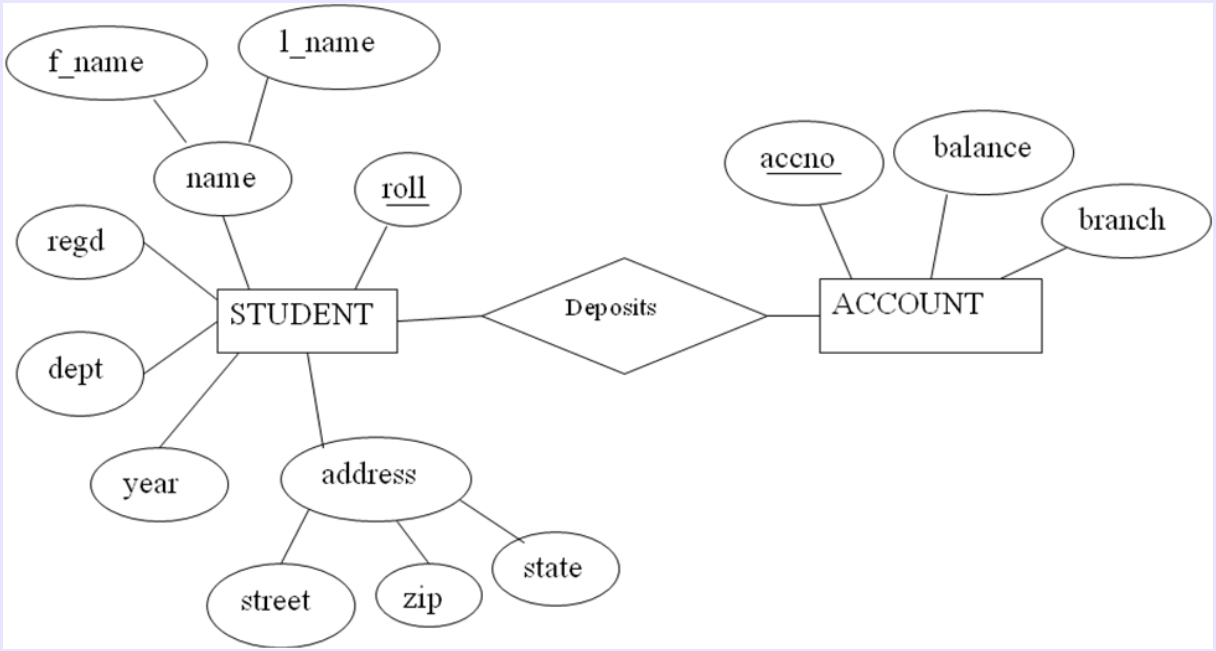
Keys

A key allows us to identify a set of attributes that suffice to distinguish entities from each other

- A key is a property of the entity set, rather than of the individual entities
- A super key is a set of one or more attributes that allow us to identify uniquely an entity in an entity set
- The minimal super keys are called candidate keys
- Primary key is a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. The primary key can be represented by underlying the attribute name. *The primary key should be chosen such that its attributes are never or very rarely changed*
- The remaining candidate keys except the primary key are called as alternate keys

Keys...

- If none of the columns is a candidate for the primary key in a table, sometimes database designers use an extra column as a primary key instead of using a composite key. Such key is known as the surrogate key
- Foreign key is the set of attributes which is used for referring to another entity set having the primary key. *In ER diagram, foreign key can not be represented*



Keys for Relationship sets

Keys for Relationship sets

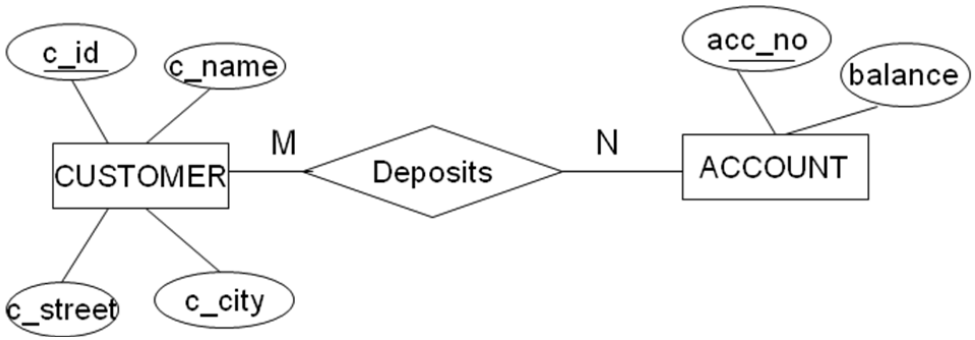
Let R be a relationship set involving entity sets E_1, E_2, ... E_n. Let PK(E_i) denotes the set of attributes that forms the primary key for entity set E_i

- If the relationship set R has no descriptive attributes associated with it, then the set of attributes $PK(E_1)\ U\ PK(E_2)\ U\ ...\ U\ PK(E_n)$ describes an individual relationship in set R
- If the relationship set R has attributes a_1, a_2 ... a_m associated with it, then the set of attributes $PK(E_1)\ U\ PK(E_2)\ U\ ...\ U\ PK(E_n)\ U\ \{a_1,a_2,...a_m\}$ describes an individual relationship in set R

Keys for Relationship sets...

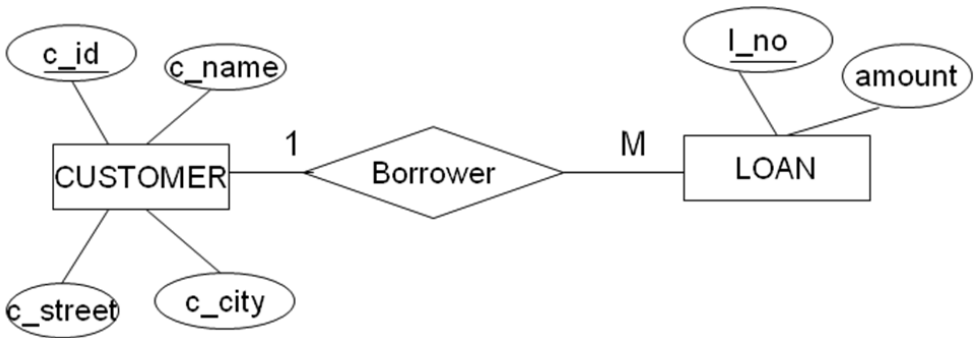
M:N relationship

The primary key of the relationship set consists of the union of the primary keys of the entity sets



1:M relationship

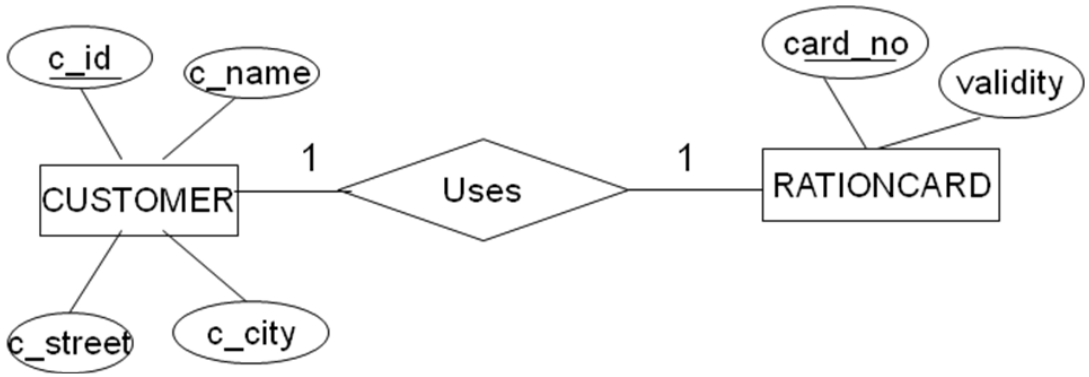
The primary key of the relationship set is the primary key of the many side entity set



Keys for Relationship sets...

1:1 relationship

The primary key of the relationship set is either the primary key of any entity set



Relationship Types

Relationship Types

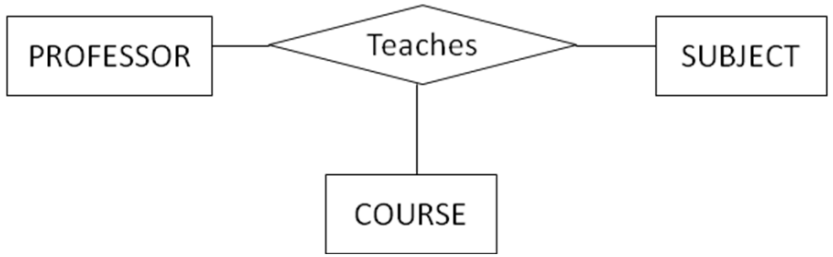
A relationship type is a meaningful association among entity types. The **degree** of a relationship type is defined as the number of entity sets participating in that relationship type

Binary relationship

A relationship type is said to be binary when two entity sets are involved

Ternary relationship

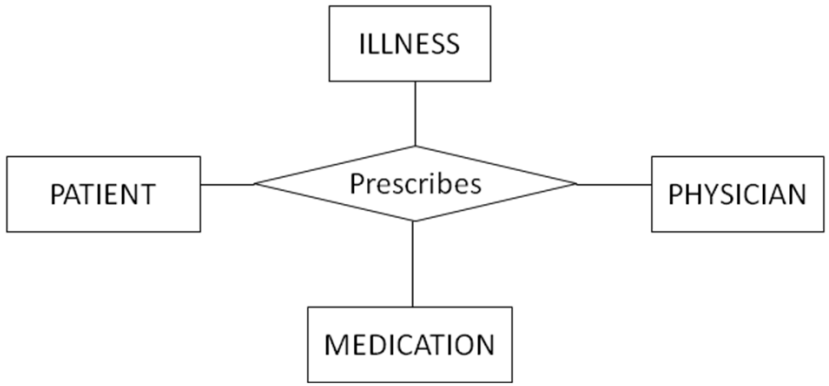
Relationship types that involve three entity sets are defined as ternary relationships



Relationship Types...

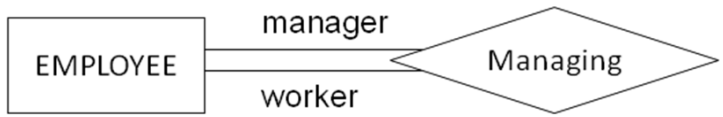
Quaternary relationship

A relationship of degree four can be referred to as a quaternary relationship



Recursive relationship

The participation of an entity set in a relationship type can be indicated by its role name. When used in recursive relationship types, role names describe the functionality of the participation

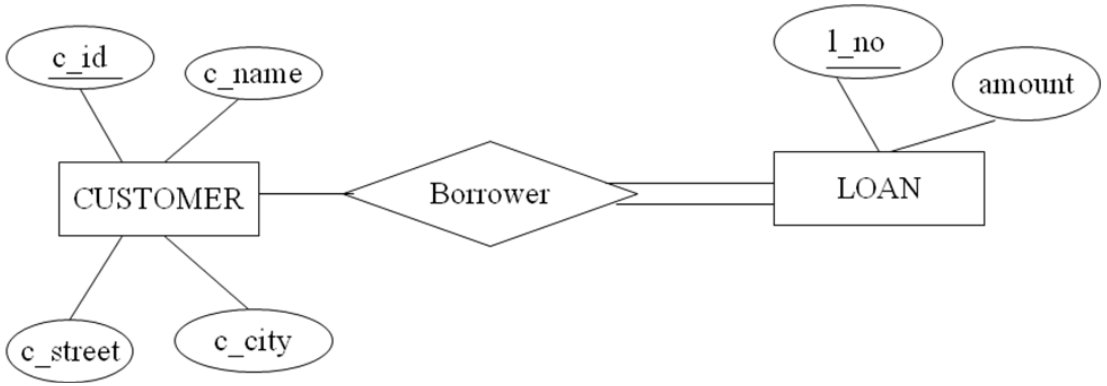


Participation Constraints

Participation Constraints

The participation constraint for an entity set in a binary relationship type is based on whether an entity of that entity set needs to be related to an entity of the other entity set through this relationship type

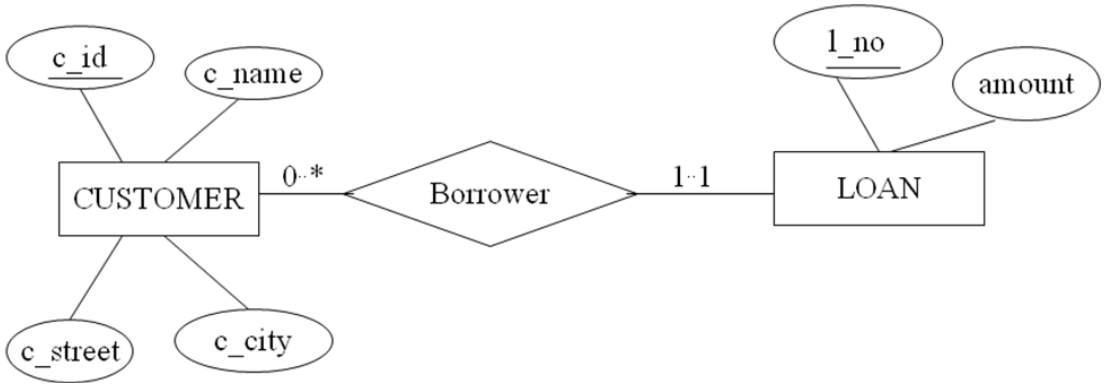
- **Total participation**: If, in order to exist, every entity must participate in the relationship, then participation of the entity set in that relationship type is total or mandatory. The total participation is represented by double lines
- **Partial participation**: If an entity can exist without participating in the relationship, then participation of the entity type in that relationship type is partial or optional



I..h Representation

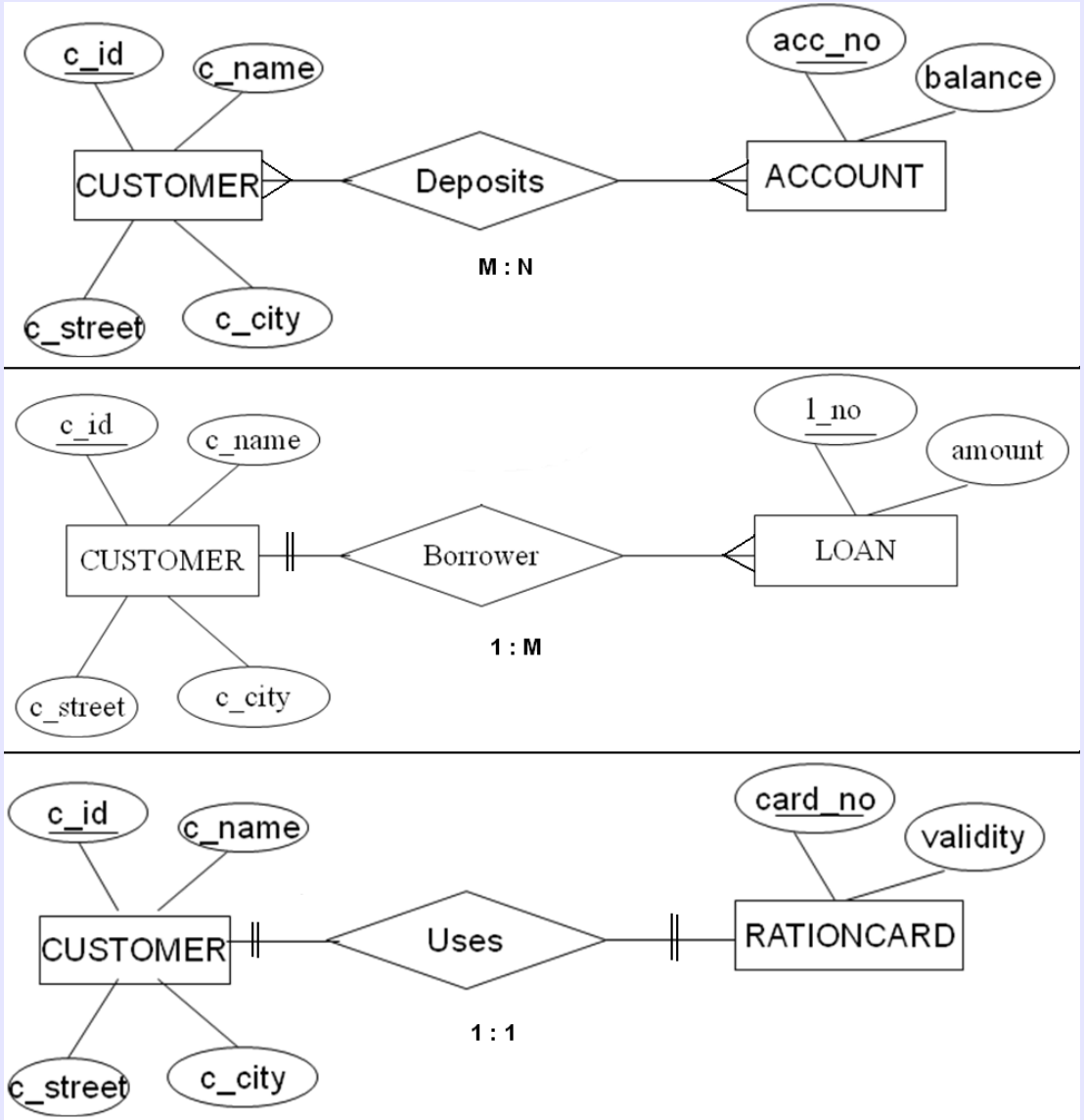
An edge between an entity set and a binary relationship set can have an associated minimum and maximum cardinality; shown as *l..h*, where *l* is the minimum and *h* is the maximum cardinality

- A minimum value of 1 indicates total participation of the entity set in the relationship set
- A maximum value of 1 indicates that the entity participates in at most one relationship, while a maximum value * indicates no limit
- 1..* indicates total participation or double line



Mapping Cardinality Representation (Crow's Foot Notation)

Mapping Cardinality Representation (Crow's Foot Notation)



Strong Entity sets and Weak Entity sets

Strong Entity sets and Weak Entity sets

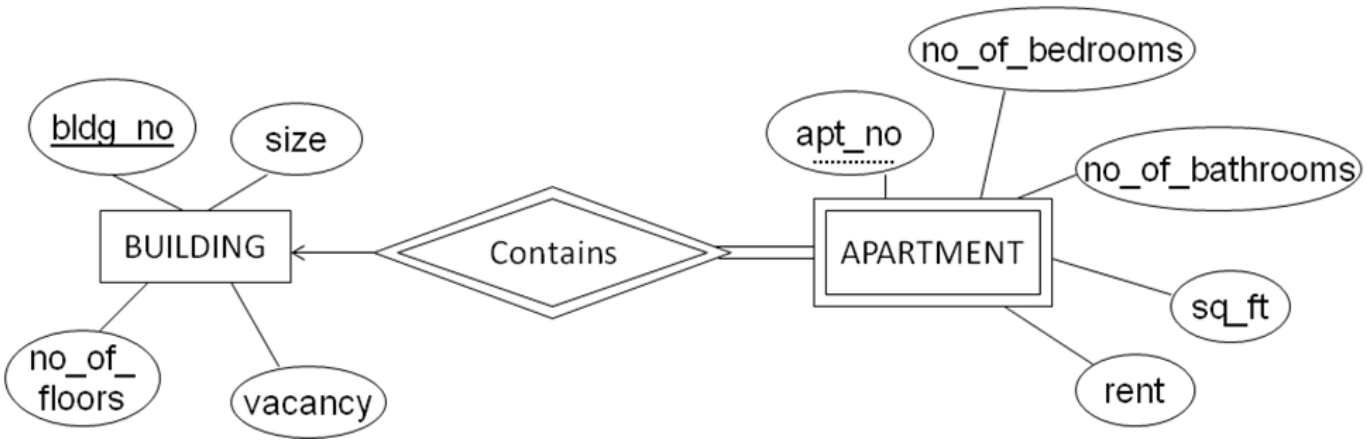
An entity set where the entities have independent existence (that is, each entity is unique) is referred to as a strong or base entity set. On the other hand, the entity set that does not have independent existence, that is, an entity set that does not have its own unique identifier is known as weak entity set

- For a weak entity set to be meaningful, it must be associated with another strong entity set called **identifying or owner entity set**
- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**
- The identifying relationship is usually many-to-one from the weak entity set to the identifying entity set and the participation of the weak entity set in the relationship is total participation

Strong Entity sets and Weak Entity sets...

Strong Entity sets and Weak Entity sets...

- An attribute in a weak entity set, which in conjunction with a unique identifier of the parent entity set in the identifying relationship type uniquely identifies weak entities, is called the partial key of the weak entity set and is denoted by a dotted underline. The partial key of a weak entity set is sometimes referred to as a **discriminator**
- The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator



Database Management System 7

ER Design Issues

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

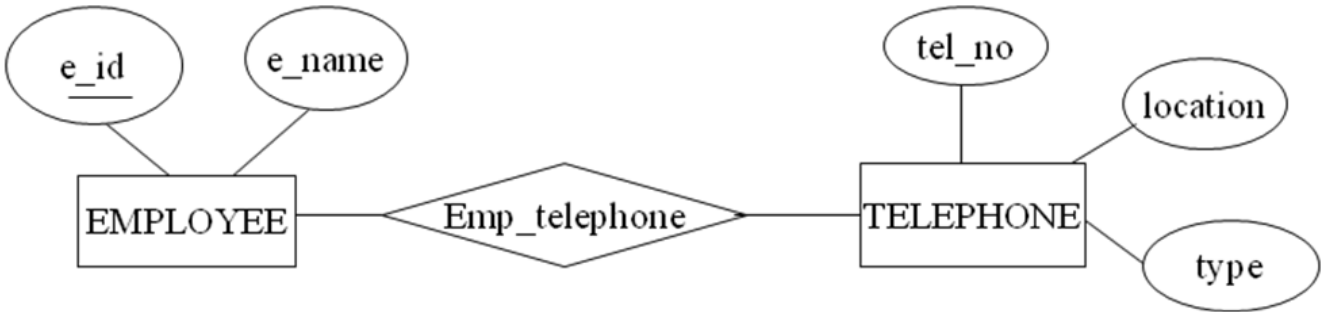
ER Design Issues

ER Design Issues

ER design issues need to be discussed for better ER- design

1. Use of Entity set vs. Attributes

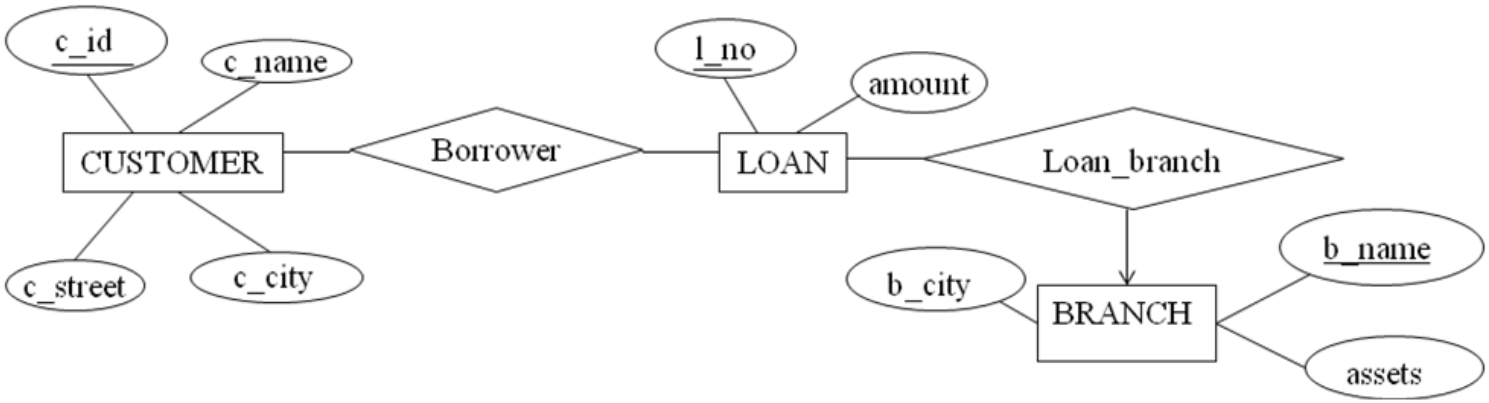
In the real world situations, sometimes it is difficult to select the property as an attribute or an entity set

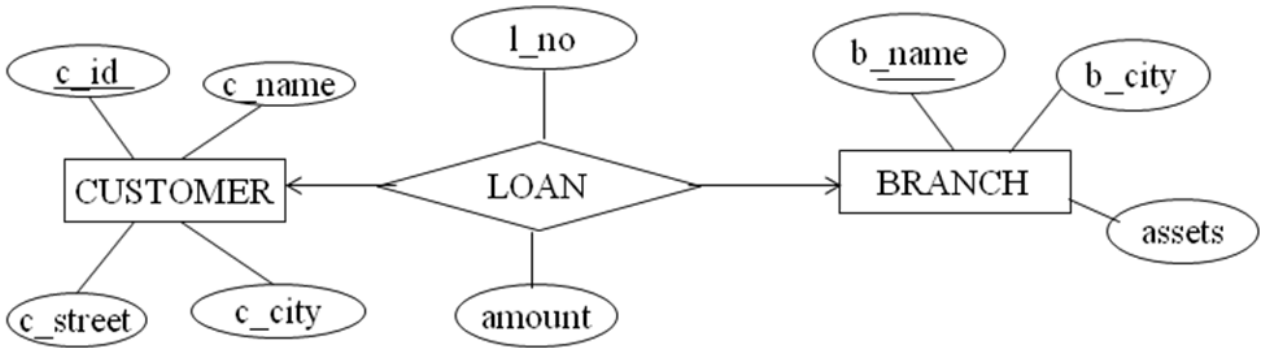


ER Design Issues...

2. Use of Entity sets vs. Relationship sets

Sometimes, an entity set can be better expressed in relationship set. Thus, it is not always clear whether an object is best expressed by an entity set or a relationship set

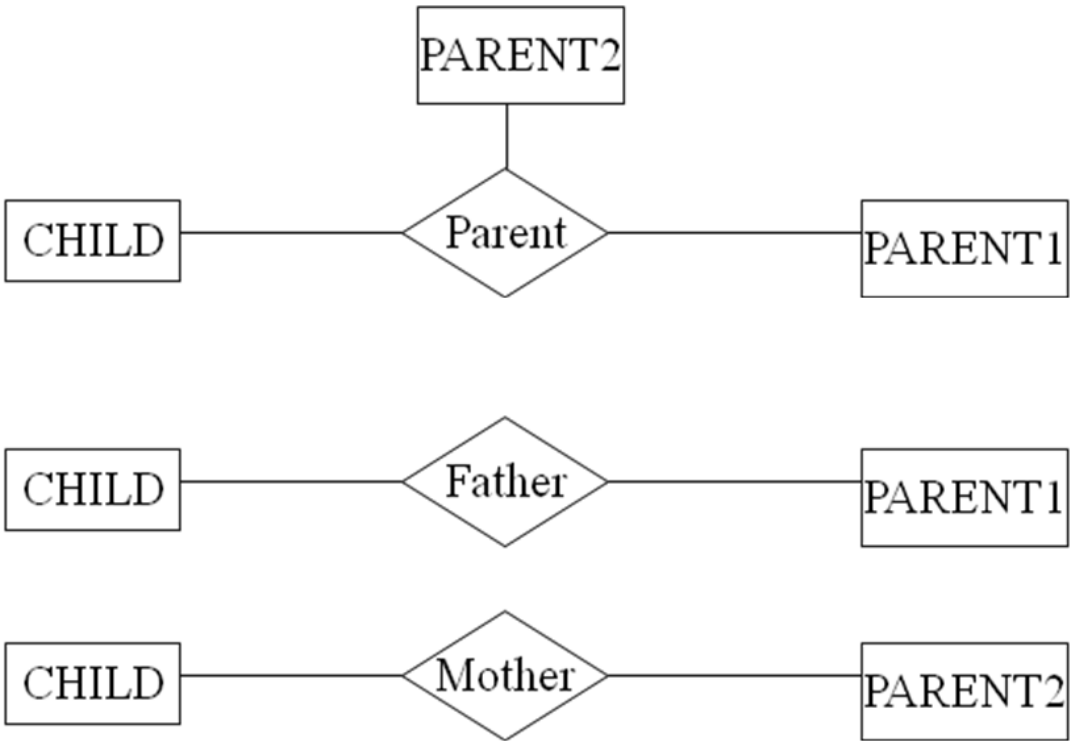




ER Design Issues...

3. Binary vs. n-ary relationship sets

Relationships in databases are often binary. Some relationships that appear to be non-binary could actually be better represented by several binary relationships



ER Design Issues...

3. Binary vs. n-ary relationship sets...

It is always possible to replace a non-binary relationship set by a number of distinct binary relationship sets. For example, consider a ternary relationship R associated with three entity sets A, B and C. We can replace the relationship set R by an entity set E and create three relationship sets as:

- R_A, relating E and A
- R_B, relating E and B
- R_C, relating E and C

If the relationship set R had any attributes, these are assigned to entity set E. A special identifying attribute is created for E

ER Design Issues...

4. Placement of Relationship Attributes

The cardinality ratio of a relationship can affect the placement of relationship attributes:

- **One-to-Many**: Attributes of 1:M relationship set can be repositioned to only the entity set on the many side of the relationship
- **One-to-One**: The relationship attribute can be associated with either one of the participating entities
- **Many-to-Many**: Here, the relationship attributes can not be represented to the entity sets; rather they will be represented by the entity set to be created for the relationship set

ER Design Methodologies

ER Design Methodologies

The guidelines that should be followed while designing an ER diagram are discussed below:

- Recognize entity sets
- Recognize relationship sets and participating entity sets
- Recognize attributes of entity sets and attributes of relationship sets
- Define binary relationship types and existence dependencies
- Define general cardinality, constraints, keys, and discriminators
- Design diagram

Database Management System 8

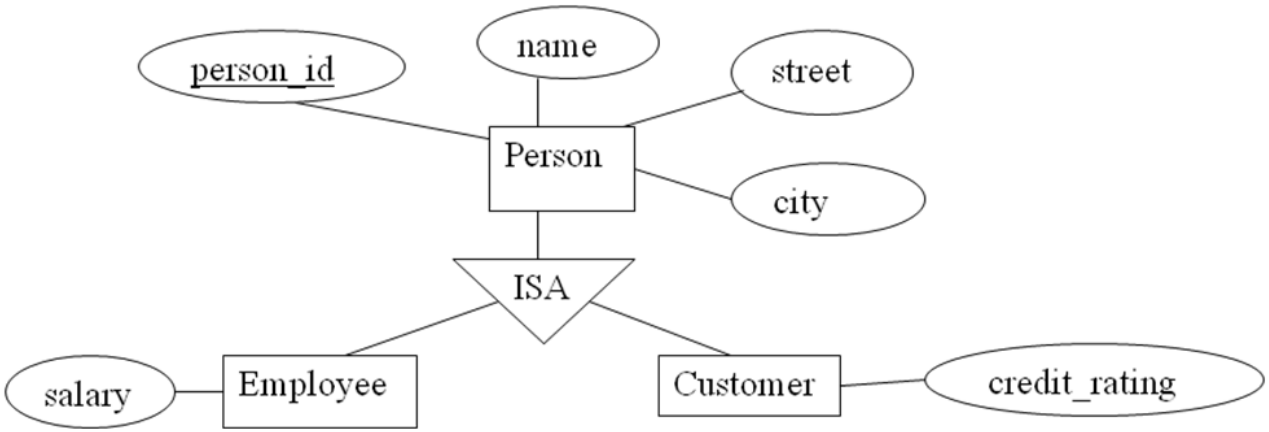
Enhanced ER-Model

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Specialization

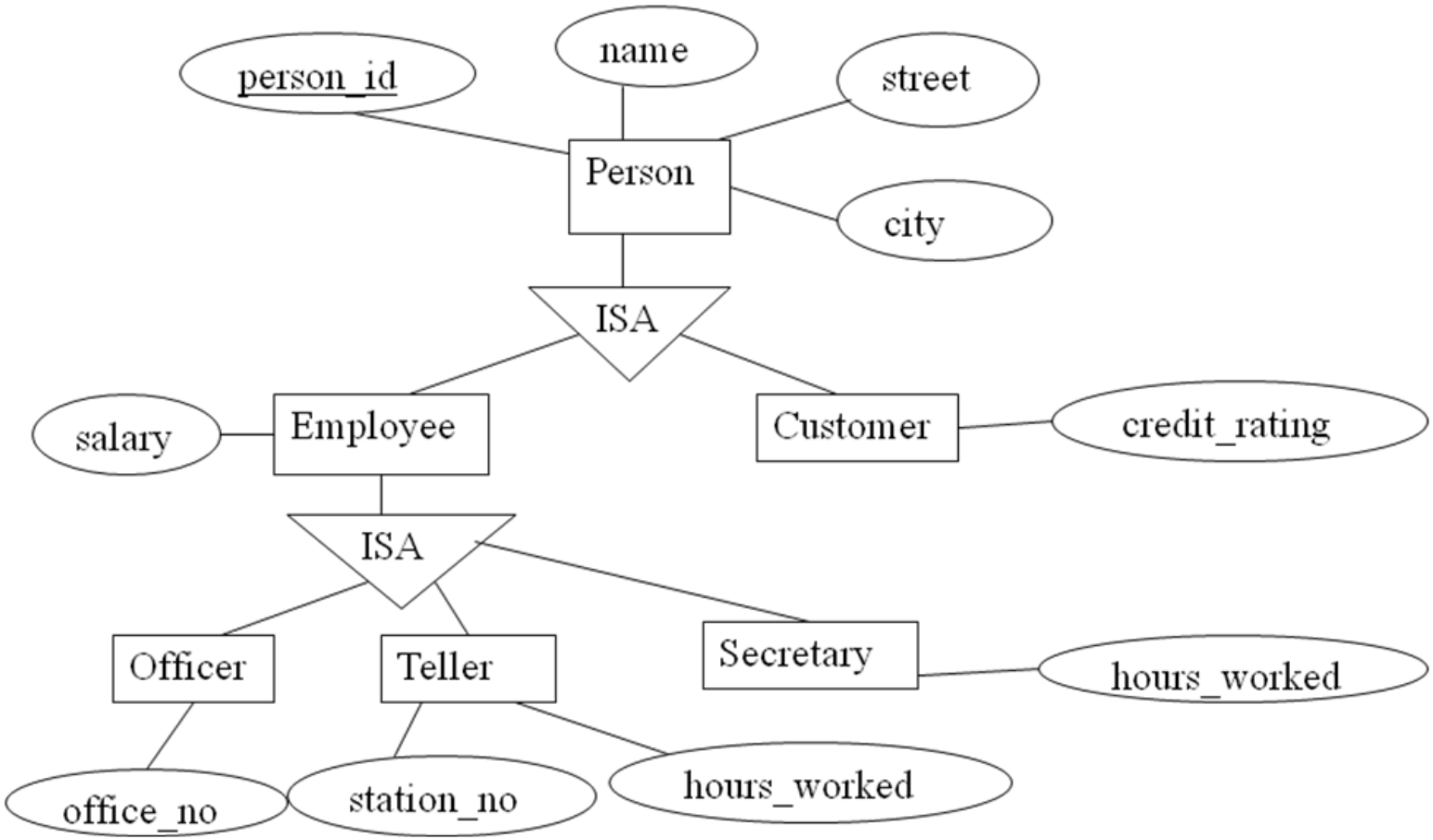
Specialization

The process of designating sub groupings within an entity set is called Specialization. An entity set may be specialized by more than one distinguishing features. In ER-design, specialization is depicted by a Triangle component labeled ISA (is a)



Specialization...

We can apply specialization repeatedly to refine a design scheme



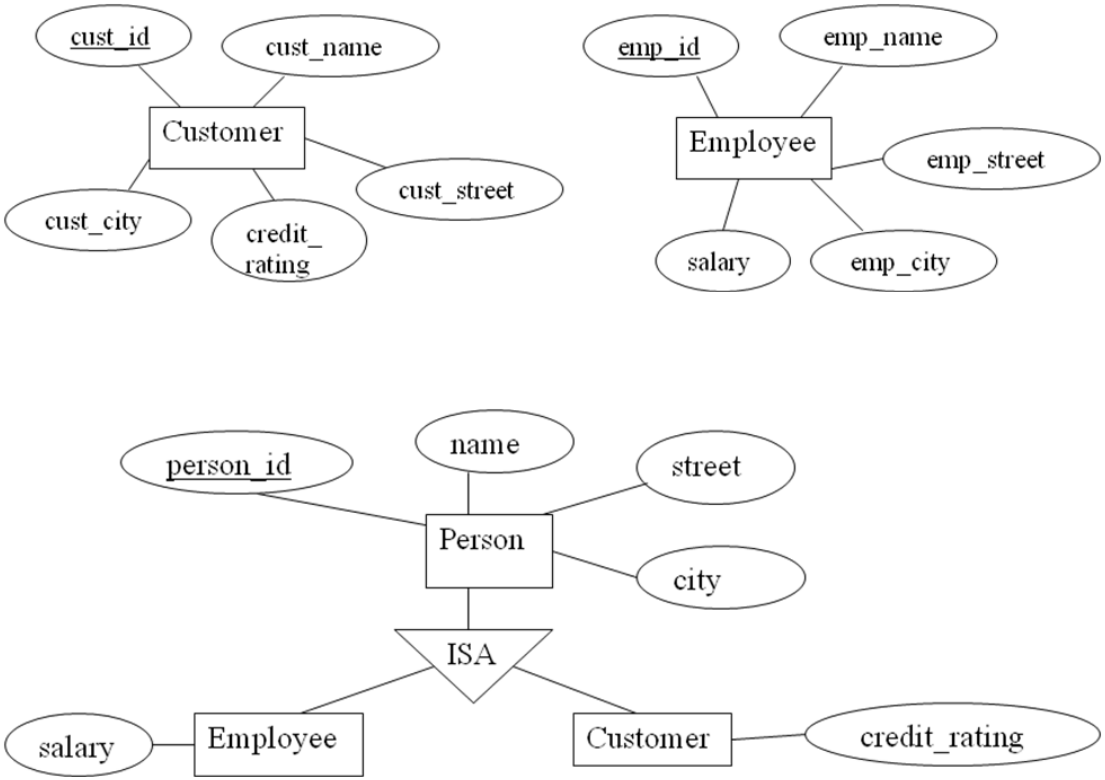
Generalization

Generalization

The commonality can be expressed by Generalization, which is a containment relationship that exists between a higher-level entity set and one or more low-level entity sets

- To create a generalization, the attributes must given a common name and represented with the higher-level entity
- Generalization is a simple inversion of specialization
- Specialization adopts *top-down* approach, while Generalization adopts *bottom-up* approach
- A crucial property of the higher-level and lower-level entities created by specialization and generalization is attribute inheritance
- A lower-level entity set (or subclass) also inherits participation in the relationship sets in which its higher-level entity (or superclass) participates

Generalization...



Constraints on Generalization/Specialization

a. Condition defined or not

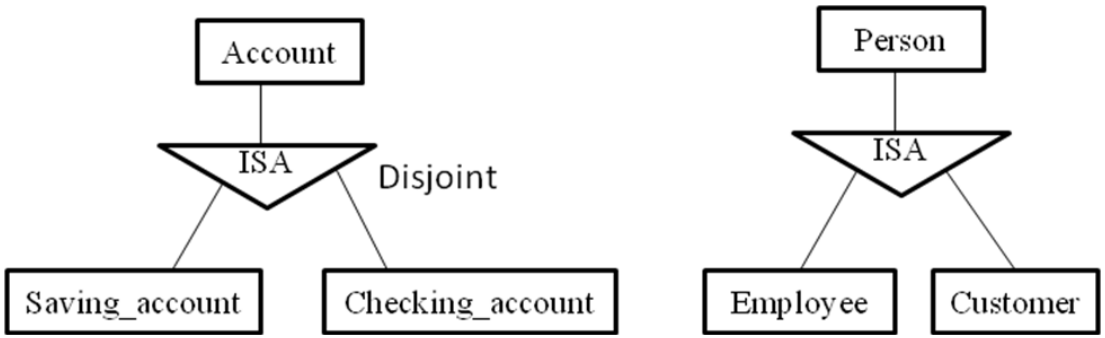
- **Condition-defined**: In condition defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate. Since all the lower-level entities are evaluated on the basis of the same attribute, this type of generalization is also said to be attribute-defined

- **User-defined**: User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set

Constraints on Generalization/Specialization...

b. Disjoint or Overlapping

- **Disjoint**: A Disjointness constraint requires that an entity belong to only one lower-level entity set
- **Overlapping**: In overlapping generalizations, the same entity may belong to more than one lower-level entity set within a single generalization
- *Lower-level entity overlap is the default case*. A disjointness constraint must be placed explicitly on a generalization. This is done by adding the word disjoint next to the ISA symbol



Constraints on Generalization/Specialization...

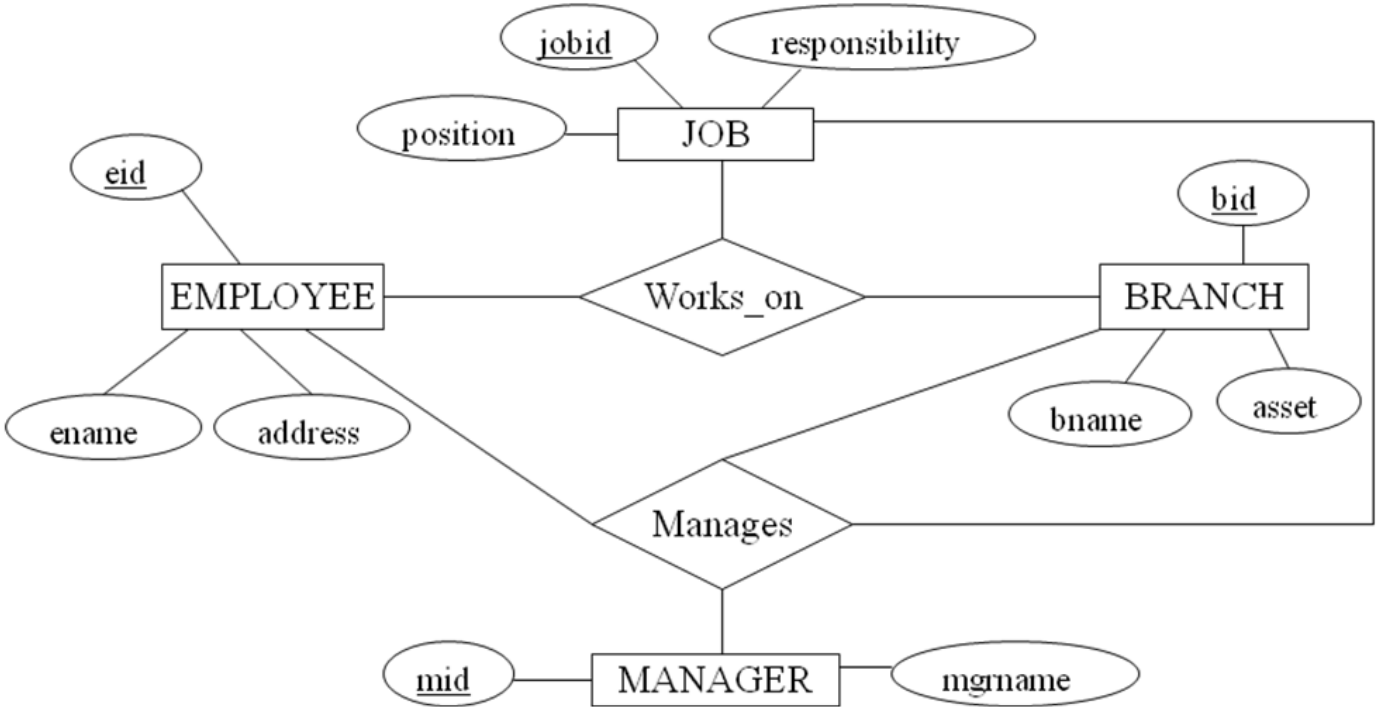
c. Completeness Constraint

Completeness constraint on a generalization/specialization specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization

- **Total generalization/specialization**: Each higher-level entity must belong to a lower-level entity set
- **Partial generalization/specialization**: Some higher-level entities may not belong to any lower-level entity set
- *Partial generalization is the default*. Total generalization in an ER diagram can be specified by using a double line to connect the box representing the higher-level entity set to the triangle symbols

Aggregation

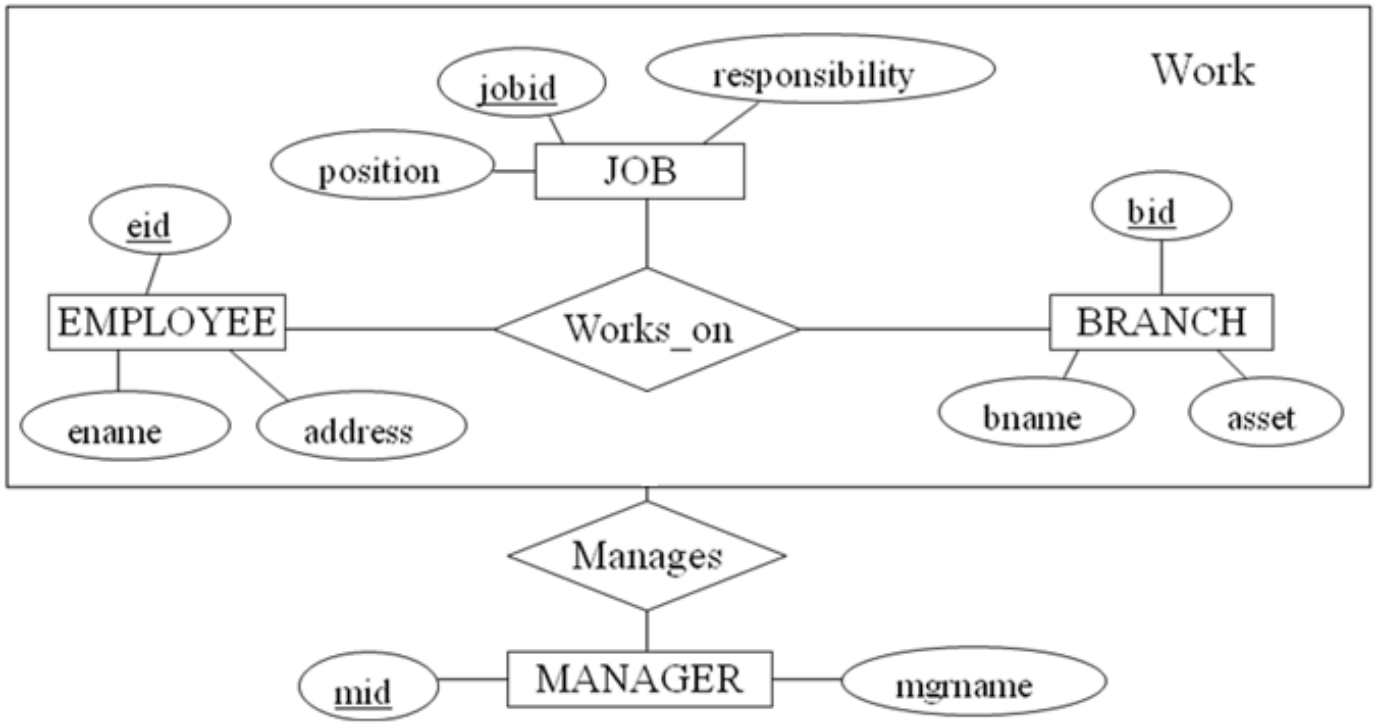
One limitation of the ER model is that it can not express relationship among relationships



Aggregation...

Aggregation

Aggregation *is an abstraction through which relationships are treated as higher-level entities*. Thus, aggregation allows us to treat a relationship set as an entity set for the purposes of participation in (other) relationships



Database Management System 9

Relational Model

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Relational Model

Relational Model

- Relational data model is the primary data model for commercial data- processing applications
- A relational database consists of a collection of tables, each of which is assigned a unique name
- A row in a table represents a relationship among a set of values. Thus, a table is an entity set and a row is an entity
- The columns or properties are called attributes
- For each attribute, there is a set of permitted values, called the domain of that attribute. Same domain can be shared by more than one attribute
- Degree is the number of attributes in the relation/ table, where as Cardinality is the number of tuples or rows in the relation/table
- The attribute values are required to be atomic, i.e. indivisible

Relational Model...

Relational Model...

- Let D_1, D_2, and D_3 are the domains. Any row of the table consists of a 3-tuple (v_1, v_2, v_3) where $v_1 \in D_1$, $v_2 \in D_2$ and $v_3 \in D_3$. Thus, the table will contain only a subset of the set of all possible rows. Therefore, the table is a subset of D_1 x D_2 x D_3
- Each attribute of a relation has a unique name
- NULL Value is a domain value which is a member of any possible domain
- Database Schema is the logical design of the database. If (a_1, a_2 ...a_n) be the attributes, then the relation schema will be R=(a_1,a_2 ...a_n)
- Database Instance is the snapshot of the data in the database at a given instant of time
- *Relation is denoted by lower case names and Relation Schema is the name beginning with an uppercase letter*

Relational Database

Relational Database

Relational database is a database consisting of multiple relations or tables. The information about an enterprise is broken up into parts, with each relation storing one part of the information

The normalization process deals with how to design relational schemas

Relational Data Integrity

Relational Data Integrity

Candidate key is an attribute or set of attributes that can uniquely identify a row or tuple in a table. Let R be the relation with attributes a_1, a_2 ... a_n. The set of attributes of R is said to be a candidate key of R iff the following two properties holds:

- **Uniqueness**: At any given time, no two distinct tuples or rows of R have the same value for a_i, the same value for a_j ...a_n
- **Minimality**: No proper subset of the set (a_i, a_j ... a_n) has the uniqueness property

The major types of integrity constraints are:

1. Domain Constraints

- All the values that appear in a column of a relation must be taken from the same domain
- This constraint can be applied by specifying a particular data type to a column

Relational Data Integrity...

2. Entity Integrity

- The entity integrity rule is designed to assure that every relation has a primary key, and that the data values for that primary key are all valid
- Usually, the primary key of each relation is the first column
- Entity integrity guarantees that every primary key attribute is NOT NULL
- Primary key performs the unique identification function in a relational model

3. Referential Integrity

- In relational data model, associations between tables are defined by using foreign keys
- A referential integrity constraint is a rule that maintains consistency among the rows of two relations

Relational Data Integrity...

3. Referential Integrity...

- The rule states that if there is a foreign key in one relation, either each foreign key value must match a primary key value in the other table or else the foreign key value must be NULL
- A foreign key that references its own relation is known as recursive foreign key
- The linking between the foreign key and primary key allows a set of relations to form an integrated database

4. Operational Constraints

- These are the constraints enforced in the database by the business rules or real world limitations

Database Languages

DDL (Data Definition Language)

- DDL is used to define the conceptual schema. The definition includes the information of all the entity sets and their associated attributes as well as the relationships between the entity sets
- The data values stored in the database must specify certain consistency constraints. The database systems check these constraints every time the database is updated
- The output of the DDL is placed in the Data Dictionary which contains the metadata (data about data)
- The data dictionary is considered to be a special type of table, which can only be accessed and updated by the database system itself
- The database system consults the data dictionary, before querying or modifying the actual data, for the validation purpose
- *CREATE, ALTER, DROP, RENAME & TRUNCATE*

Database Languages...

DML (Data Manipulation Language)

- DML is used to manipulate data in the database
- A query is a statement in the DML that requests the retrieval of data from the database
- *SELECT, INSERT, UPDATE & DELETE*

DCL (Data Control Languages)

- DCL allows in changing the permissions on database structures
- *GRANT & REVOKE*

TCL (Transaction Control Language)

- TCL allows permanently recording the changes made to the rows stored in a table or undoing such changes
- *COMMIT, ROLLBACK & SAVEPOINT*

CODD's Rules

Codd's rules are a set of 12 rules proposed by E. F. Codd designed to define what is required from a database management system in order for it to be considered relational, i.e. RDBMS. Any database that satisfies even six rules may be categorized as RDBMS

Rule0

A relational system should be able to manage databases, entirely through its relational capabilities

Rule1: Information representation

The entire information is explicitly and logically represented by the data values of the tables in the relational data model

Rule2: Guaranteed access

In relational model, at each cell, i.e. the interaction of each row and column, it will have one and only one value of data (or NULL value). Each value of data must be addressable via the combination of a table name, primary key value and the column name

CODD's Rules...

Rule3: Systematic treatment of NULL values

NULL values are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way independent of data type

Rule4: Database description rule

The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data. This means, the RDBMS should have a data dictionary

Rule5: Comprehensive data sub-language

The RDBMS should have its own extension of SQL. The SQL should support Data Definition, View Definition, Data Manipulation, Integrity Constraint, and Authorization

CODD's Rules...

Rule6: Views updation

All views that are theoretically updatable are also updatable by the system. Similarly, the views which are theoretically non-updatable are also non-updatable by the database system

Rule7: High-level update, insert, deletes

A RDBMS should not only support retrieval of data as relational sets, but should also support insertion, updation and deletion of data as a relational set

Rule8: Physical data independence

Application programs and terminal activities are not disturbed if any changes are made either in storage representations or access methods

Rule9: Logical data independence

User programs and the user should not be aware of any changes to the structure of the tables such as the addition of extra columns

CODD's Rules...

Rule10: Distribution independence

A relational DBMS has distribution independence. The RDBMS may spread across more than one system and across several networks. However to the end-user, the tables should appear no different to those that are local

Rule11: Integrity rule

Integrity rules must be supported by the relational data sub-language; they can be stored in the catalogue and not in the application program. **Entity integrity**: no component of a primary key may have a NULL value. **Referential integrity**: for every unique non-null 'foreign key' values in the database, there should be a matching primary key value from the same domain

Rule12: Data integrity cannot be subverted

If a relational system has a low-level language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language

Database Management System 10

Conversion of ER model to Relational Model

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

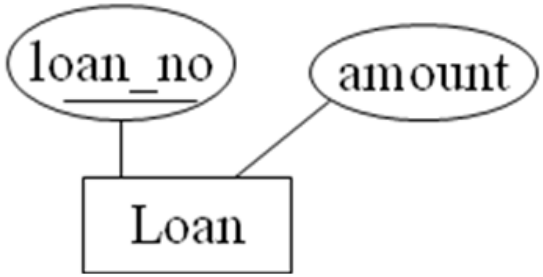
Conversion of ER model to Relational Model

Conversion of ER model to Relational Model

A database that conforms to an ER diagram schema can be represented by a collection of relational schemas. Both the ER model and Relational data model are abstract, logical representations of real-world enterprises

1. Representation of Strong Entity sets

A strong entity set reduces to a schema with the same attributes. The primary key of the entity set serves as the primary key of the resulting schema

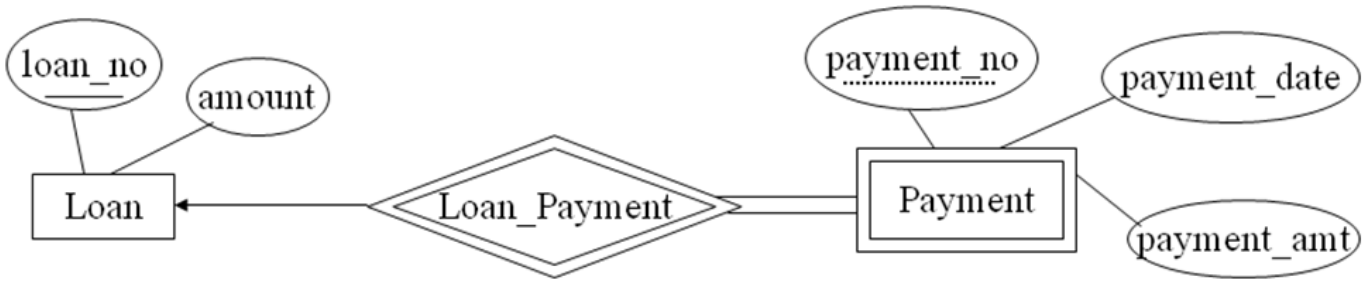


Loan = (loan_no, amount)

Conversion of ER model to Relational Model...

2. Representation of Weak Entity sets

A weak entity set becomes a table that includes a column for the primary key of the identifying strong entity set. The primary key is constructed by the collection of foreign key and partial key



Loan = (loan_no, amount)

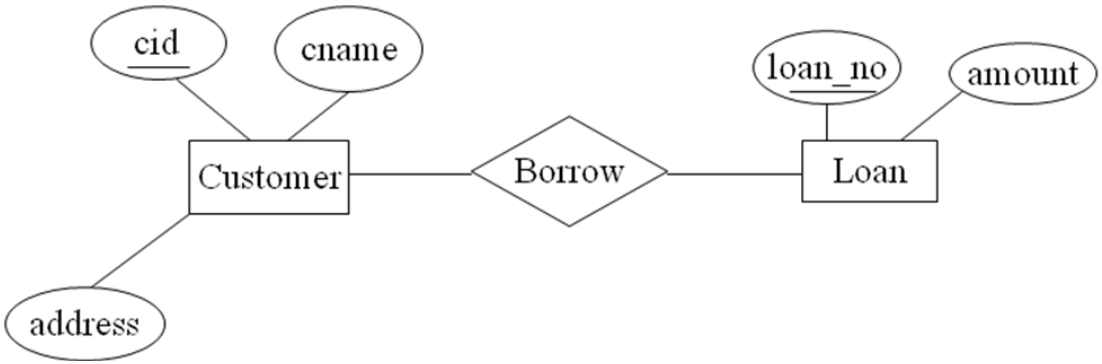
Payment = (loan_no, payment_no, payment_date, payment_amt)

Conversion of ER model to Relational Model...

3. Representation of Relationship sets

3.a. Binary M:N

Union of the primary key attributes from the participating entity sets becomes the primary key of the relationship



Customer = (cid, cname, address)
Loan = (loan_no, amount)
Borrow = (cid, loan_no)

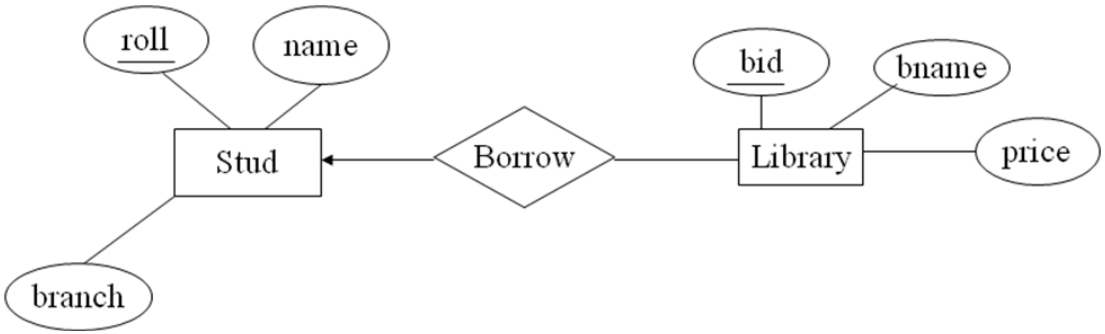
If borrow_date is mentioned as descriptive attribute, then

Borrow = (cid, loan_no, borrow_date)

Conversion of ER model to Relational Model...

3.b. Binary M:1/1:M

Construct two tables, one for the entity set at 1 side and another for entity set at M side, add the descriptive attributes and a reference of the primary key of 1 side to the entity set at M side



Stud = (roll, name, branch)

Library = (bid, bname, price, roll)

The foreign key can be represented by specifying the name as:

Library = (bid, bname, price, borrowing_roll)

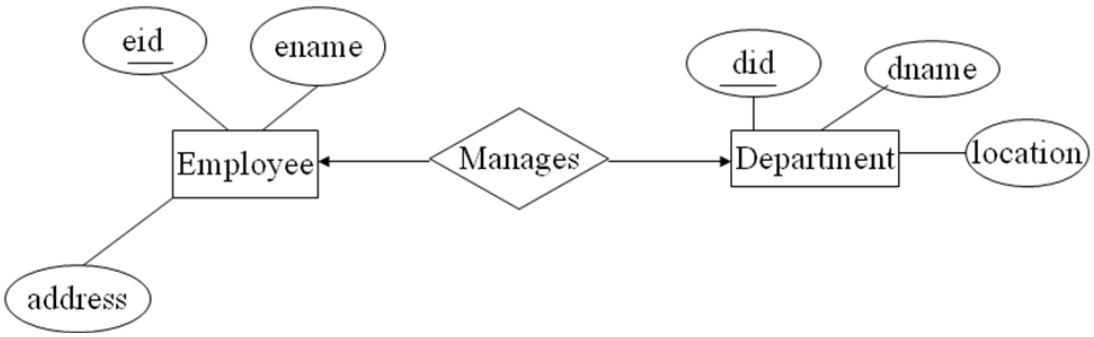
If borrow_date is the descriptive attribute, then

Library = (bid, bname, price, borrowing_roll, borrow_date)

Conversion of ER model to Relational Model...

3.c. Binary 1:1

Construct two tables. In this case, either side can be chosen to act as the many side. That is, extra attributes can be added to either of the tables corresponding to the two entity sets, but not at the same time

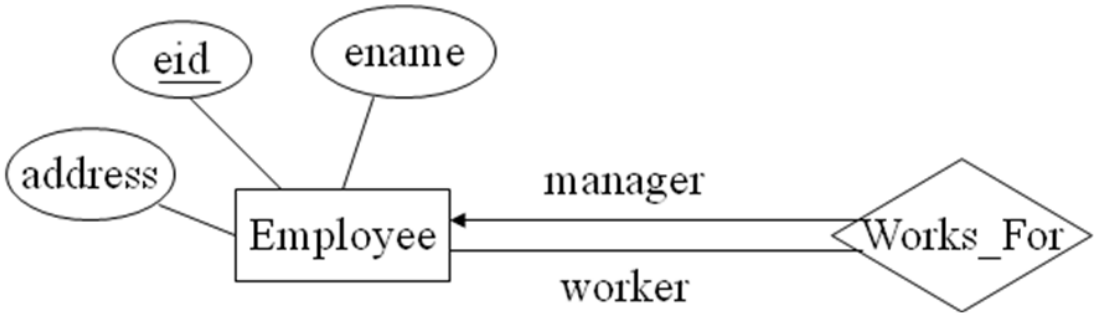


Employee = (<u>eid</u>, ename, address, did)
Department = (<u>did</u>, dname, location)
If it is required to mention the relationship name, then
Employee = (<u>eid</u>, ename, address, manager_did)
If department entity set will be considered as many side, then
Employee = (<u>eid</u>, ename, address)
Department = (<u>did</u>, dname, location, manager_eid)

Conversion of ER model to Relational Model...

4. Representation of Recursive Relationship sets

Two tables will be constructed; one for entity set and one for relationship set



Employee = (<u>eid</u>, ename, address)

Works_for = (<u>mgrid, workerid</u>)

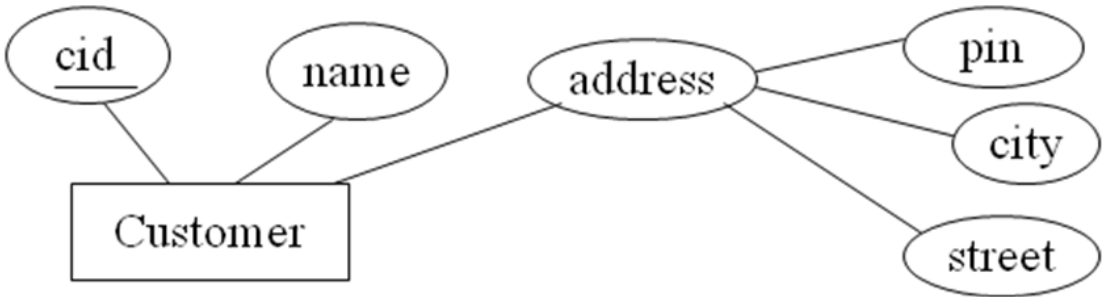
This ER diagram can also be represented by using a single relation schema. In such cases, the schema contains a foreign key for each tuple in the original entity set

Employee = (<u>eid</u>, ename, address, manager_id)

Conversion of ER model to Relational Model...

5. Representation of Composite attributes

The composite attributes are flattened out by creating a separate attribute for each of its parts

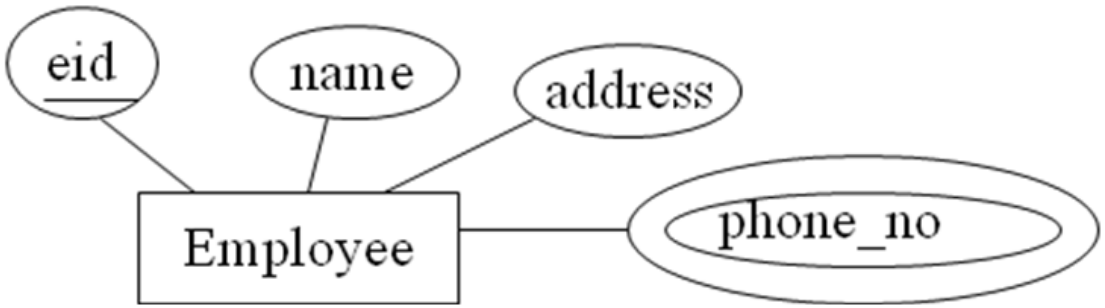


Customer = (cid, name, address_street, address_city, address_pin)

Conversion of ER model to Relational Model...

6. Representation of Multi-valued attributes

A multi-valued attribute M of an entity set E is represented by a separate schema E_M as E_M(primary key of E,M)

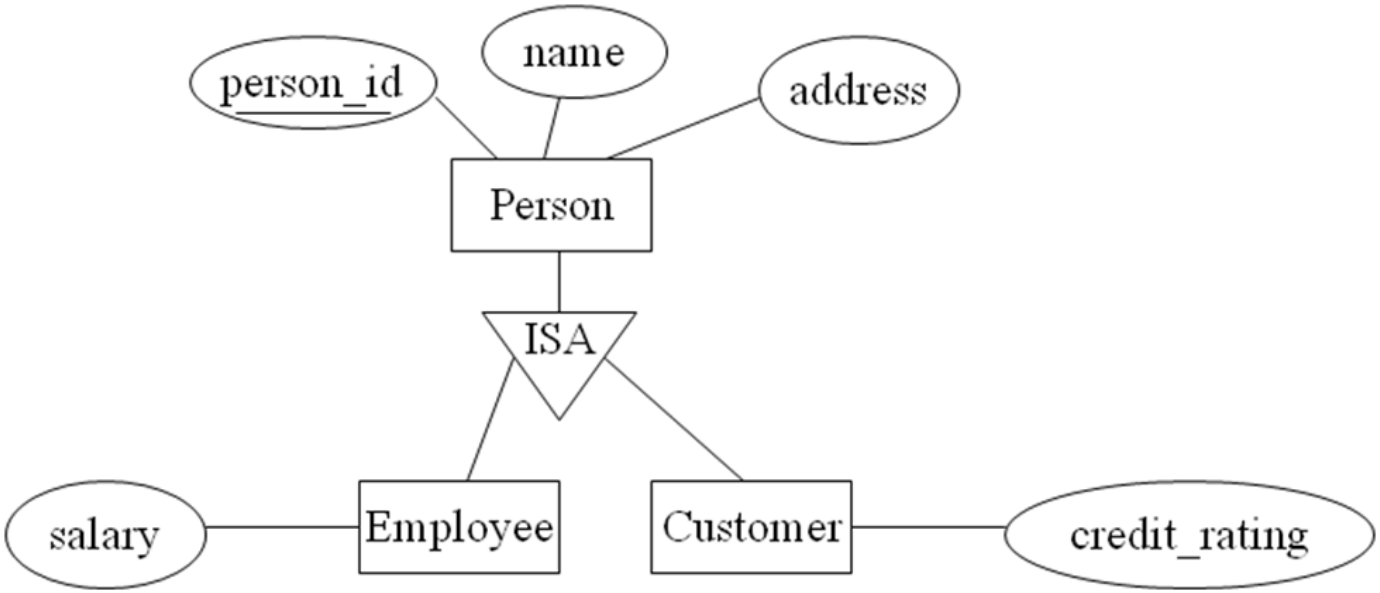


Employee = (eid, name, address)
Employee_phone_no = (eid, phone_no)

Conversion of ER model to Relational Model...

7. Representation of Generalization/Specialization

In case of generalization/specialization-related ER diagram, one schema will be constructed for the generalized entity set and the schemas for each of the specialized entity sets



Person = (person_id, name, address)
Employee = (person_id, salary)
Customer = (person_id, credit_rating)

Conversion of ER model to Relational Model...

Representation of Generalization/Specialization...

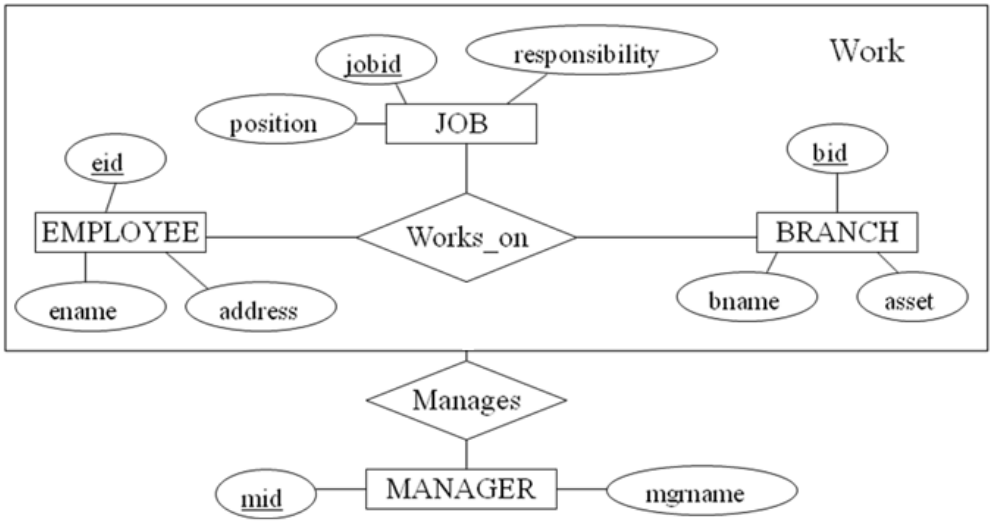
When the generalization/specialization is a disjointness case, the schemas are constructed only for the specialized entity sets

Employee = (employee_id, name, address, salary)
Customer = (customer_id, name, address, credit_rating)

Conversion of ER model to Relational Model...

8. Representation of Aggregation

To represent aggregation, create a schema containing the primary key of the aggregated relationship, primary key of the associated entity set and descriptive attributes (if any)



Employee = (eid, name, address)
Branch = (bid, bname, asset)
Job = (jobid, position, responsibility)
Works_on = (eid, bid, jobid)
Manager = (mid, mgrname)
Manages = (eid, bid, jobid, mid)

Database Management System 11

Relational Algebra

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Query Language

Query Language

Language in which user requests information from the database are:

- Procedural language
- Nonprocedural language

The categories of different languages are:

- SQL
- Relational Algebra
- Relational Calculus
  - Tuple Relational Calculus
  - Domain Relational Calculus

Relational Algebra

Relational Algebra

Relational algebra is a procedural language for manipulating relations. Relational algebra operations manipulate relations. That is, these operations use one or two existing relations to create a new relation

- Fundamental operators
  - Unary: SELECT, PROJECT, RENAME
  - Binary: UNION, SET DIFFERENCE, CARTESIAN PRODUCT
- Secondary operators
  - INTERSECTION, NATURAL JOIN, DIVISION, and ASSIGNMENT

Relational Algebra...

The relational schemas used for different operations are:

- **Customer(cust_name, cust_street, cust_city)**
  - used to store customer details
- **Branch(branch_name, branch_city, assets)**
  - used to store branch details
- **Account(acc_no, branch_name, balance)**
  - stores the account details
- **Loan(loan_no, branch_name, amount)**
  - stores the loan details
- **Depositor(cust_name, acc_no)**
  - stores the details about the customers' account
- **Borrower(cust_name, loan_no)**
  - used to store the details about the customers' loan

SELECT Operator(σ)

SELECT Operator(σ)

SELECT operation is used to create a relation from another relation by selecting only those tuples or rows from the original relation that satisfy a specified condition. It is denoted by sigma (σ) symbol. The predicate appears as a subscript to σ

The argument relation is in parenthesis after the σ. The result is a relation that has the same attributes as the relation specified in <relation-name>. The general syntax of select operator is:
$\sigma_{<selection-condition>}$ **(<relation name>)**

Query: Find the details of the loans taken from 'Bhubaneswar Main' branch.

$\sigma_{branch\_name='BhubaneswarMain'}$ (Loan)

- The operators used in selection predicate may be: =, $\neq$, <, $\leq$, >, $\geq$.
- Different predicates can be combined into a larger predicate by using the connectors like: AND($\wedge$), OR($\vee$), NOT($\neg$)

SELECT Operator(σ)...

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

$\sigma_{branch\_name='BhubaneswarMain'}$ (Loan)

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00

$\sigma_{branch\_name='BhubaneswarMain' AND amount > 10{,}000{,}000}$ (Loan)

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00

PROJECT Operator(π)

PROJECT Operator(π)

PROJECT operation can be thought of as eliminating unwanted columns. It eliminates the duplicate rows. It is denoted by pie(π) symbol. The attributes needed to be appeared in the resultant relation appear as subscript to π.

The argument relation follows in parenthesis. The general syntax of project operator is:

$\pi_{<attribute-list>}$ **(<relation name>)**

Query: Find the loan numbers and respective loan amounts.

$\pi_{loan\_no,amount}$ (Loan)

PROJECT Operator(π)...

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

$\pi_{loan\_no, amount}$ (Loan)

loan_no	amount
L201	50,000,000.00
L202	5,000,000.00
L203	100,000,000.00
L204	60,000,000.00

Composition of Relational Operators

Composition of Relational Operators

Relational algebra operators can be composed together into a relational algebra expression to answer the complex queries

Q:Find the name of the customers who live in Bhubaneswar

Customer

cust_name	cust_street	cust_city
Rishi	India Gate	New Delhi
Sarthak	M. G. Road	Bangalore
Manas	Shastri Nagar	Bhubaneswar
Ramesh	M. G. Road	Bhubaneswar
Mahesh	Juhu	Mumbai

$\pi_{\ cust\_name}\ (\sigma_{\ cust\_city = 'Bhubaneswar'}\ (\text{Customer}))$

cust_name
Manas
Ramesh

RENAME Operator(ρ)

RENAME Operator(ρ)

The results of relational algebra expressions do not have a name that can be used to refer them. It is useful to be able to give them names; the rename operator is used for this purpose. It is denoted by rho(ρ) symbol.

The general syntax of rename operator is:
$\rho_{\ x}$ **(E)**

Assume E is a relational-algebra expression with arity n. The second form of rename operation is: $\rho_{\ x(b_1,b_2,\ldots b_n)}$ **(E)**

$\pi_{\ cust\_name}\ (\sigma_{\ cust\_city='Bhubaneswar'}(\text{Customer}))$ can be written as:

1. $\rho_{\ Customer\_Bhubaneswar}\ (\sigma_{\ cust\_city='Bhubaneswar'}(\text{Customer}))$

2. $\pi_{\ cust\_name}$ (Customer_Bhubaneswar)

RENAME Operator(ρ)...

RENAME Operator(ρ)...

The different forms of the rename operation for renaming the relation are:

a. ρ_{S} **(R)**

b. $\rho_{S(b_1,b_2,\ldots b_n)}$ **(R)**

c. $\rho_{(b_1,b_2,\ldots b_n)}$ **(R)**

For example, the attributes of Customer (cust_name, cust_street, cust_city) can be renamed as:

$\rho_{(name,street,city)}$ **(Customer)**

Union Compatibility

Union Compatibility

To perform the set operations such as UNION, DIFFERENCE and INTERSECTION, the relations need to be union compatible for the result to be a valid relation

Two relations $R_1(a_1, a_2, \ldots a_n)$ and $R_2(b_1, b_2, \ldots b_m)$ are union compatible iff:

- n = m, i.e. both relations have same arity
- $\text{dom}(a_i) = \text{dom}(b_i)$ for $1 \leq i \leq n$

UNION Operator(∪)

UNION Operator(∪)

The union operation is used to combine data from two relations. It is denoted by union(∪) symbol. The union of two relations $R_1(a_1, a_2, \ldots a_n)$ and $R_2(b_1, b_2, \ldots b_n)$ is a relation R_3 $(c_1, c_2, \ldots c_n)$ such that:

dom(c_i) = dom(a_i) ∪ dom(b_i), 1 ≤ i ≤ n

$R_1 \cup R_2$ is a relation that includes all tuples that are either present in R_1 or R_2 or in both without duplicate tuples

Depositor

cust_name	acc_no
Manas	A101
Ramesh	A102
Rishi	A103
Mahesh	A104
Mahesh	A105

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

π_{cust_name} (Depositor) ∪ π_{cust_name} (Borrower)

cust_name
Manas
Ramesh
Rishi
Mahesh

DIFFERENCE Operator(-)

DIFFERENCE Operator(-)

The difference operation is used to identify the rows that are in one relation and not in another. It is denoted as (-) symbol. The difference of two relations $R_1(a_1, a_2, \ldots a_n)$ and $R_2(b_1, b_2, \ldots b_n)$ is a relation R_3 $(c_1, c_2, \ldots c_n)$ such that:

dom(c_i) = dom(a_i) - dom(b_i), 1 $\leq$ i $\leq$ n

R_1 - R_2 is a relation that includes all tuples that are in R_1, but not in R_2

Depositor

cust_name	acc_no
Manas	A101
Ramesh	A102
Rishi	A103
Mahesh	A104
Mahesh	A105

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

π *cust_name* (Depositor) - π *cust_name* (Borrower)

cust_name
Manas

Cartesian Product Operator($\times$)

Cartesian Product Operator($\times$)

The Cartesian product of two relations $R_1(a_1, a_2, \ldots a_n)$ with cardinality **i** and $R_2(b_1, b_2, \ldots b_m)$ with cardinality **j** is a relation R_3 with

- degree **k = n + m**,
- cardinality **i*j** and
- attributes $(a_1, a_2, \ldots a_n, b_1, b_2, \ldots b_m)$

$R_1 \times R_2$ is a relation that includes all the possible combinations of tuples from R_1 and R_2. The Cartesian product is used to combine information from any two relations
It is not a useful operation by itself; but is used in conjuction with other operations

Cartesian Product Operator($\times$)...

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

Borrower $\times$ Loan

cust_name	Borrower.loan_no	Loan.loan_no	branch_name	amount
Ramesh	L201	L201	Bhubaneswar Main	50,000,000.00
Ramesh	L201	L202	Bhubaneswar Main	5,000,000.00
Ramesh	L201	L203	Mumbai Main	100,000,000.00
Ramesh	L201	L204	Juhu	60,000,000.00
Ramesh	L202	L201	Bhubaneswar Main	50,000,000.00
Ramesh	L202	L202	Bhubaneswar Main	5,000,000.00
Ramesh	L202	L203	Mumbai Main	100,000,000.00
Ramesh	L202	L204	Juhu	60,000,000.00
Mahesh	L203	L201	Bhubaneswar Main	50,000,000.00
Mahesh	L203	L202	Bhubaneswar Main	5,000,000.00
Mahesh	L203	L203	Mumbai Main	100,000,000.00
Mahesh	L203	L204	Juhu	60,000,000.00
Rishi	L204	L201	Bhubaneswar Main	50,000,000.00
Rishi	L204	L202	Bhubaneswar Main	5,000,000.00
Rishi	L204	L203	Mumbai Main	100,000,000.00
Rishi	L204	L204	Juhu	60,000,000.00

Cartesian Product Operator($\times$)...

Query: Find out the customer and their loan details taken from Bhubaneswar Main branch.

Ans: $\sigma_{branch\_name='BhubaneswarMain' AND Borrower.loan\_no=Loan.loan\_no}$ (Borrower $\times$ Loan)

cust_name	Borrower.loan_no	Loan.loan_no	branch_name	amount
Ramesh	L201	L201	Bhubaneswar Main	50,000,000.00
Ramesh	L202	L202	Bhubaneswar Main	5,000,000.00

Intersection Operator(∩)

Intersection Operator(∩)

The intersection operation is used to identify the rows that are common to two relations. It is denoted by (∩) symbol. The intersection of two relations $R_1(a_1, a_2, \ldots a_n)$ and $R_2(b_1, b_2, \ldots b_n)$ is a relation R_3 $(c_1, c_2, \ldots c_n)$ such that:

dom(c_i) = dom(a_i) ∩ dom(b_i), 1 ≤ i ≤ n

$R_1 \cap R_2$ is a relation that includes all tuples that are present in both R_1 and R_2

The intersection operation can be rewritten by a pair of set difference operations as **R ∩ S = R - (R - S)**

Depositor

cust_name	acc_no
Manas	A101
Ramesh	A102
Rishi	A103
Mahesh	A104
Mahesh	A105

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

π_{cust_name} (Depositor) ∩ π_{cust_name} (Borrower)

cust_name
Ramesh
Mahesh
Rishi

JOIN Operator($\bowtie$)

JOIN Operator($\bowtie$)

The join is a binary operation that is used to combine certain selections and a Cartesian product into one operation. It is denoted by join ($\bowtie$) symbol.

The join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relations, and finally removes the duplicate attributes

Query: Find the names of customers who have a loan at the bank, along with the loan number and the loan amount.
Ans:This query can be solved by using the PROJECT, SELECT and CARTESIAN PRODUCT operators as:

$\pi_{cust\_name, Loan.loan\_no, amount}(\sigma_{Borrower.loan\_no = Loan.loan\_no}(Borrower \times Loan))$

This same expression can be simplified by using the JOIN as:

$\pi_{cust\_name, loan\_no, amount}(Borrower \bowtie Loan))$

JOIN Operator($\bowtie$)...

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

$\pi_{cust\_name, loan\_no, amount}$ (Borrower $\bowtie$ Loan))

cust_name	loan_no	amount
Ramesh	L201	50,000,000.00
Ramesh	L202	5,000,000.00
Mahesh	L203	100,000,000.00
Rishi	L204	60,000,000.00

Division Operator($\div$)

Division Operator($\div$)

The division operation creates a new relation by selecting the rows in one relation that match every row in another relation. The division operation requires that we look at an entire relation at once. It is denoted by division ($\div$) symbol

Let A, B, C are three relations and we desire B $\div$ C to give A as the result. This operation is possible iff:

- The columns of C must be a subset of the columns of B. The columns of A are all and only those columns of B that are not columns of C
- A row is placed in A if and only if it is associated with B and with every row of C

The division operation is the reverse of the Cartesian product operation as: B = (B $\times$ C) $\div$ C

Division operator is suited to queries that include the phrase every or all as part of the condition

Division Operator(÷)...

Depositor

cust_name	acc_no
Manas	A101
Ramesh	A102
Rishi	A103
Mahesh	A104
Mahesh	A105

Account

acc_no	branch_name	balance
A101	Bhubaneswar Main	100,000.00
A102	Shastri Nagar	50,000.00
A103	India Gate	5,000,000.00
A104	Juhu	600,000.00
A105	Mumbai Main	10,000,000.00

Branch

branch_name	branch_city	assets
Bhubaneswar Main	Bhubaneswar	Gold
Shastri Nagar	Bhubaneswar	Mines
India Gate	New Delhi	Gold
Juhu	Mumbai	Sea Shore
Mumbai Main	Mumbai	Movie

Query: Find all the customers who have an account at all the branches located in Mumbai

$$\pi_{cust\_name, branch\_name}(\text{Depositor} \bowtie \text{Account}) \div \pi_{branch\_name}(\sigma_{branch\_city = 'Mumbai'}(\text{Branch}))$$

cust_name
Mahesh

Assignment Operator(←)

Assignment Operator(←)

It works like assignment in a programming language. In relational algebra, the assignment operator gives a name to a relation. It is denoted by (←) symbol
Assignment must always be made to a temporary relation variable. The result of the right of the ← symbol is assigned to the relation variable on the left of the ← symbol
With the assignment operator, a query can be written as a sequential program consisting of:

- a series of assignment,
- followed by an expression whose value is displayed as a result of the query

$\pi_{cust\_name,branch\_name}$ (Depositor ⋈ Account) ÷ π_{branch_name} ($\sigma_{branch\_city='Mumbai'}$ (Branch))
can be simplified as:
Temp1 ← $\pi_{cust\_name,branch\_name}$ (Depositor ⋈ Account)
Temp2 ← π_{branch_name} ($\sigma_{branch\_city='Mumbai'}$ (Branch))
Result = Temp1 ÷ Temp2

Database Management System 12

JOIN

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Generalized Projection

Generalized Projection

The generalized-projection operation extends the projection operation by allowing arithmetic functions to be used in the projection list. The general form of generalized-projection is:

$\pi_{F_1, F_2 \ldots F_n}(E)$

Ex:Emp=(ssn, salary, deduction, years_service) be a relation. A report may be required to show net_salary=salary-deduction, bonus=2000*years_service and tax=0.25*salary

$$REPORT \leftarrow \rho_{(ssn, net\_salary, bonus, tax)}(\pi_{ssn, salary - deduction, 2000 * years\_service, 0.25 * salary}(Emp))$$

Aggregate Functions(g)

Aggregate Functions(g)

Aggregate functions take a collection of values and return a single value as a result. NULL value will not participate in the aggregate functions. The general form of aggregate function is:
$grouping\_attribute\ g\ aggregate\_functions\ (R)$

Let Works = (emp_id, ename, salary, branch_name)
Query: Find the total sum of salaries of all the employees
Ans: $g_{SUM(salary)}(Works)$

Query: Find the total sum of salaries of all the employees in each branch
Ans: $branch\_name\ g_{SUM(salary)}(Works)$

Query: Find the maximum salary for the employees at each branch, in addition to the sum of the salaries
Ans: $branch\_name\ g_{SUM(salary),MAX(salary)}(Works)$

Query: Find the number of employees working
Ans: $g_{COUNT(emp\_id)}(Works)$

Join

Join

The join operation is used to connect data across relations. Tables are joined on columns that have the same **datatype** and **data width** in the tables

Join operation joins two relations by **merging** those tuples from two relations that satisfy a given condition. The condition is defined on attributes belonging to relations to be joined

Different categories of join are:

- Inner Join
- Outer Join
- Self Join

Inner Join

Inner Join

In the inner join, tuples with NULL valued join attributes do not appear in the result. Tuples with NULL values in the join attributes are also eliminated. The different types of inner join are:

- Theta Join
- Equi Join
- Natural Join

Theta Join($\bowtie_\theta$)

Theta Join($\bowtie_\theta$)

The theta join is a join with a specified condition involving a column from each relation. This condition specifies that the two columns should be compared in some way

The comparison operator can be any of the six: <, $\leq$, >, $\geq$, = and $\neq$

Theta join is denoted by ($\bowtie_\theta$) symbol. The general form of theta join is:
$R \bowtie_\theta S = \pi_{all} (\sigma_\theta (R \times S))$

- Degree (Result) = Degree (R) + Degree (S)
- Cardinality (Result) $\leq$ Cardinality(R) $\times$ Cardinality(S)

Theta Join($\bowtie_\theta$)...

Account

acc_no	branch_name	balance
A101	Bhubaneswar Main	100,000.00
A102	Shastri Nagar	50,000.00
A103	India Gate	5,000,000.00
A104	Juhu	600,000.00
A105	Mumbai Main	10,000,000.00

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

Q: Find the account details as well as loan details for the situations where depositing balance is greater than or equal to the borrowing amount

Account $\bowtie_{balance \geq amount}$ Loan

acc_no	branch_name	balance	loan_no	branch_name	amount
A103	India Gate	5,000,000.00	L202	Bhubaneswar Main	5,000,000.00
A105	Mumbai Main	10,000,000.00	L202	Bhubaneswar Main	5,000,000.00

Equi Join($\bowtie_{=}$)

Equi Join($\bowtie_{=}$)

The equi join is the theta join based on equality of specified columns. That means the equi join is the special type of theta join where the comparison operator is =

The general form of theta join is:
$R \bowtie_{=} S = \pi_{all} (\sigma_{=} (R \times S))$

- Degree (Result) = Degree (R) + Degree (S)
- Cardinality (Result) $\leq$ Cardinality(R) $\times$ Cardinality(S)

Equi Join($\bowtie_{=}$)...

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

Q: Find the customer name and their loan details

Borrower $\bowtie_{Borrower.loan\_no=Loan.loan\_no}$ Loan

cust_name	Borrower.loan_no	Loan.loan_no	branch_name	amount
Ramesh	L201	L201	Bhubaneswar Main	50,000,000.00
Ramesh	L202	L202	Bhubaneswar Main	5,000,000.00
Mahesh	L203	L203	Mumbai Main	100,000,000.00
Rishi	L204	L204	Juhu	60,000,000.00

Natural Join($\bowtie$)

Natural Join($\bowtie$)

To perform natural join on two relations, they should contain at least one common attributes. It is just like the equi join with the elimination of the common attributes. The natural join is denoted by ($\bowtie$) symbol

The general form of theta join is:
$R \bowtie S = \pi_{all-common\_attributes} (\sigma_{=} (R \times S))$

- Degree (Result) = Degree (R) + Degree (S) - Degree ($R \cap S$)
- Cardinality (Result) $\leq$ Cardinality(R) $\times$ Cardinality(S)

The general form of the natural join can also be represented as:
$R \bowtie S = \pi_{all} (R \bowtie S)$

Natural Join($\bowtie$)...

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

Loan

loan_no	branch_name	amount
L201	Bhubaneswar Main	50,000,000.00
L202	Bhubaneswar Main	5,000,000.00
L203	Mumbai Main	100,000,000.00
L204	Juhu	60,000,000.00

Q: Find the customer name and their loan details

Borrower $\bowtie$ Loan

cust_name	loan_no	branch_name	amount
Ramesh	L201	Bhubaneswar Main	50,000,000.00
Ramesh	L202	Bhubaneswar Main	5,000,000.00
Mahesh	L203	Mumbai Main	100,000,000.00
Rishi	L204	Juhu	60,000,000.00

Outer Join

Outer Join

It is an extension of the natural join operation to deal with the missing information. The outer join consists of two steps:

- First, a natural join is executed
- Then if any record in one relation does not match a record from the other relation in the natural join, that unmatched record is added to the join relation, and the additional columns are filled with NULLs

The different types of outer join are:

- Left Outer Join
- Right Outer Join
- Full Outer Join

Left Outer Join

Left Outer Join(⟕)

The left outer join preserves all tuples in left relation. The left outer join is denoted by symbol:

⟕

All information from the left relation is present in the result of the left outer join

Left Outer Join...

Customer

cust_name	cust_street	cust_city
Rishi	India Gate	New Delhi
Sarthak	M. G. Road	Bangalore
Manas	Shastri Nagar	Bhubaneswar
Ramesh	M. G. Road	Bhubaneswar
Mahesh	Juhu	Mumbai

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

Q: Find out the customer details who have taken loans as well as who have not taken loans

$$Customer \bowtie Borrower$$

cust_name	cust_street	cust_city	loan_no
Rishi	India Gate	New Delhi	L204
Ramesh	M. G. Road	Bhubaneswar	L201
Ramesh	M. G. Road	Bhubaneswar	L202
Mahesh	Juhu	Mumbai	L203
Sarthak	M. G. Road	Bangalore	NULL
Manas	Shastri Nagar	Bhubaneswar	NULL

Right Outer Join

Right Outer Join(⟖)

The right outer join preserves all tuples in right relation. The right outer join is denoted by symbol:

⟖

All information from the right relation is present in the result of the right outer join

Right Outer Join...

Borrower

cust_name	loan_no
Ramesh	L201
Ramesh	L202
Mahesh	L203
Rishi	L204

Customer

cust_name	cust_street	cust_city
Rishi	India Gate	New Delhi
Sarthak	M. G. Road	Bangalore
Manas	Shastri Nagar	Bhubaneswar
Ramesh	M. G. Road	Bhubaneswar
Mahesh	Juhu	Mumbai

Q: Find out the customer details who have taken loans as well as who have not taken loans

$$Borrower \bowtie\!\!\!\!\!\!\subset Customer$$

cust_name	loan_no	cust_street	cust_city
Rishi	L204	India Gate	New Delhi
Ramesh	L201	M. G. Road	Bhubaneswar
Ramesh	L202	M. G. Road	Bhubaneswar
Mahesh	L203	Juhu	Mumbai
Sarthak	NULL	M. G. Road	Bangalore
Manas	NULL	Shastri Nagar	Bhubaneswar

Full Outer Join

Full Outer Join(⟗)

The full outer join preserves all tuples in both relations. The full outer join is denoted by symbol:

⟗

All information from both the relations is present in the result of the full outer join

Self Join

Self Join

The self join is similar to the theta join. It joins a relation to itself by a condition. The self join can be viewed as a join of two copies of the same relation
The general form of self join is:
$R \bowtie_{\theta} R = \pi_{all} (\sigma_{\theta} (R \times R))$

Thus, the self join creates two alias or copies of the same relation; then performs the theta join by a condition based on the attributes of these two copies

Self Join...

Customer

cust_name	cust_street	cust_city
Rishi	India Gate	New Delhi
Sarthak	M. G. Road	Bangalore
Manas	Shastri Nagar	Bhubaneswar
Ramesh	M. G. Road	Bhubaneswar
Mahesh	Juhu	Mumbai

Q: Find out the customer details as well as the others' staying in the same cust_city

$$C1 \bowtie_{C1.cust\_city = C2.cust\_city} C2$$

C1.cust_name	C1.cust_street	C1.cust_city	C2.cust_name	C2.cust_street	C2.cust_city
Manas	Shastri Nagar	Bhubaneswar	Ramesh	M. G. Road	Bhubaneswar
Ramesh	M. G. Road	Bhubaneswar	Manas	Shastri Nagar	Bhubaneswar

Database Management System 13

Query Using Relational Algebra

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Query Using Relational Algebra

Employee Database

Emp(empNo,name)

Project(projectNo,pName,manager)

Assigned_To(projectNo,empNo)

Query: Find empNo of employees working on project 'comp01'

$\pi_{empNo}(\sigma_{projectNo='comp01'}(\text{Assigned\_To}))$

Query: Find details of employees working on project 'comp01'

$\pi_{empNo,name}(\sigma_{projectNo='comp01'}(\text{Emp} \bowtie \text{Assigned\_To}))$

Query: Obtain the details of employees working on the database project

$\pi_{empNo,name}(\sigma_{pName='database'}(\text{Emp} \bowtie \text{Assigned\_To} \bowtie \text{Project}))$

Query Using Relational Algebra...

Query: Find the details of employees working on the 'comp01' and 'comp02' projects

$\pi_{empNo,name}(\sigma_{projectNo='comp01' \wedge projectNo='comp02'}(\text{Emp} \bowtie \text{Assigned\_To}))$

Query: Find the empNo who don't work on project 'comp01'

$\pi_{empNo}(\text{Assigned\_To}) - \pi_{empNo}(\sigma_{projectNo='comp01'}(\text{Assigned\_To}))$

This query can be solved as:

$\pi_{empNo}(\sigma_{projectNo\neq'comp01'}(\text{Assigned\_To}))$

Query Using Relational Algebra...

Sailor Database

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Query: Find the names of sailors who've reserved boat 105

$\pi_{sname}(\sigma_{bid=105}(\text{Reserves} \bowtie \text{Sailors}))$

This query can also be written as:

$\pi_{sname}(\sigma_{bid=105}(\text{Reserves}) \bowtie \text{Sailors})$

Query: Find the names of sailors who've reserved a green boat

$\pi_{sname}(\sigma_{color='green'}(\text{Boats} \bowtie \text{Reserves} \bowtie \text{Sailors}))$

This query can also be written as:

$\pi_{sname}((\sigma_{color='green'}(\text{Boats})) \bowtie \text{Reserves} \bowtie \text{Sailors})$

Query Using Relational Algebra...

Query: Find the sailor ids of the sailors who've reserved all boats

$\pi_{sid,bid}$ (Reserves) $\div$ π_{bid} (Boats)

Query: Find the names of sailors who've reserved all boats

1. ρ_{Temp} ($\pi_{sid,bid}$ (Reserves) $\div$ π_{bid} (Boats))
2. π_{sname} (Temp $\bowtie$ Sailors)

This query can also be written as:
$\pi_{sname,bid}$ (Sailors $\bowtie$ Reserves) $\div$ π_{bid} (Boats)

Query Using Relational Algebra...

Query: Find the colors of boats reserved by Akash

$\pi_{color}((\sigma_{sname='Akash'}(\text{Sailors})) \bowtie \text{Reserves} \bowtie \text{Boats})$

Query: Find all sailor id's of sailors who have a rating of at least 10 or reserved boat 105

$\pi_{sid}(\sigma_{rating \geq 10}(\text{Sailors})) \cup \pi_{sid}(\sigma_{bid=105}(\text{Reserves}))$

Query: Find the names of sailors who have not reserved a green boat

$\pi_{sname}((\pi_{sid}(\text{Sailors}) - \pi_{sid}(\sigma_{color='green'}(\text{Boats}) \bowtie \text{Reserves})) \bowtie \text{Sailors})$

Query: Find the sailor id's of sailors with age over 20 who have not reserved a green boat

$\pi_{sid}(\sigma_{age>20}(\text{Sailors})) - \pi_{sid}(\sigma_{color='green'}(\text{Boats}) \bowtie \text{Reserves})$

Query Using Relational Algebra...

Query: Find the names of sailors who have reserved at least two boats

$\pi_{sname}(\sigma_{Reserves.sid=Reserves2.sid \wedge Reserves.bid \neq Reserves2.bid}(Reserves \times \rho_{Reserves2}(Reserves)) \bowtie Sailors)$

Query: Find the sailor id's of sailors whose rating is better than some sailor called Bobby

$\pi_{Sailors2.sid}(\sigma_{Sailors2.rating > Sailors.rating}(\rho_{Sailors2}(Sailors) \times \sigma_{sname='Bobby'}(Sailors)))$

Query: Find the sailor id's of sailors whose rating is better than every sailor called Bobby

$\pi_{sid}(Sailors) - \pi_{Sailors2.sid}(\sigma_{Sailors2.rating \leq Sailors.rating}(\rho_{Sailors2}(Sailors) \times \sigma_{sname='Bobby'}(Sailors)))$

Query: Find the sailor id's of sailors with the highest rating

$\pi_{sid}(Sailors) - \pi_{Sailors2.sid}(\sigma_{Sailors2.rating < Sailors.rating}(\rho_{Sailors2}(Sailors) \times (Sailors))$

Query Using Relational Algebra...

Shipment Database

Customer(cust_id, cust_name, annual_revenue)
Truck(truckno, driver_name)
City(city_name, population)
Shipment(shipment_no, cust_id, weight, truckno, destination_city)

Query: Find the list of shipment numbers for shipments weighing over 20 pounds

$\pi_{shipment\_no}(\sigma_{weight>20pound}(\text{Shipment}))$

Query: Find the names of customers with more than $10 million in annual revenue

$\pi_{cust\_name}(\sigma_{annual\_revenue>\$10million}(\text{Customer}))$

Query Using Relational Algebra...

Query: Find the driver of truck 45

$\pi_{driver\_name}(\sigma_{truckno=45}(\text{Truck}))$

Query: Find the names of cities which have received shipments weighing over 100 pounds

$\pi_{destination\_city}(\sigma_{weight>100pounds}(\text{Shipment}))$

Query: Find the name and annual revenue of customers who have sent shipments weighing over 100 pounds

$\pi_{cust\_name,annual\_revenue}(\sigma_{weight>100pounds}(\text{Customer} \bowtie \text{Shipment}))$

Query: Find the truck numbers of trucks which have carried shipments weighing over 100 pounds

$\pi_{truckno}(\sigma_{weight>100pounds}(\text{Shipment}))$

Query Using Relational Algebra...

Query: Find the names of drivers who have delivered shipments weighing over 100 pounds

$\pi_{driver\_name}(\sigma_{weight>100pounds}(\text{Shipment} \bowtie \text{Truck}))$

Query: List the cities which have received shipments from customers having over $15 million in annual revenue

$\pi_{destination\_city}(\sigma_{annual\_revenue>\$15million}(\text{Customer} \bowtie \text{Shipment}))$

Query: List the customers having over $5 million in annual revenue who have sent shipments weighing greater than 1 pound

$\pi_{cust\_name}(\sigma_{annual\_revenue>\$5million}(\text{Customer}) \bowtie \sigma_{weight>1pound}(\text{Shipment}))$

This query can also be written as:

$\pi_{cust\_name}(\sigma_{annual\_revenue>\$5million \wedge weight>1pound}(\text{Customer} \bowtie \text{Shipment}))$

Query Using Relational Algebra...

Query: List the customers whose shipments have been delivered by truck driver Ramesh

$\pi_{\textit{cust\_name}}(\sigma_{\textit{driver\_name}='\textit{Ramesh}'}(\text{Customer} \bowtie \text{Shipment} \bowtie \text{Truck}))$

Query: Find the customers having over $5 million in annual revenue who have sent shipments weighing less than 1 pound or have sent a shipment to Bhubaneswar

$\pi_{\textit{cust\_name}}(\sigma_{\textit{annual\_revenue}>\$5\textit{million}}(\text{Customer}) \bowtie \sigma_{\textit{weight}>1\textit{pound} \vee \textit{destination\_city}='\textit{Bhubaneswar}'}(\text{Shipment}))$

Query: Find the customers who have sent shipments to every city with population over 500000

$\pi_{\textit{cust\_name},\textit{destination\_city}}(\text{Customer} \bowtie \text{Shipment}) \div \pi_{\textit{city}}(\sigma_{\textit{population}>500000}(\text{City}))$

Query Using Relational Algebra...

Query: List the drivers who have delivered shipments for customers with annual revenue over $20 million to cities with population over 1 million

$\pi_{driver\_name}(\sigma_{annual\_revenue>20million}(\text{Customer}) \bowtie \text{Shipment} \bowtie \text{Truck} \bowtie (\sigma_{population>1million}(\text{City})))$

This query can also be written as:

$\pi_{driver\_name}(\sigma_{annual\_revenue>20million \wedge population>1million}(\text{Customer} \bowtie \text{Shipment} \bowtie \text{Truck} \bowtie \text{City}))$

Query: Find the cities which have received shipments from every customer

$\pi_{destination\_city, cust\_id}(\text{Shipment}) \div \pi_{cust\_id}(\text{Customer})$

Query: Find the drivers who have delivered shipments to every city

$\pi_{driver\_name, destination\_city}(\text{Truck} \bowtie \text{Shipment}) \div \pi_{city\_name}(\text{City})$

Database Management System 14

Relational Calculus

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Relational Calculus

Relational Calculus

Relational calculus is non-procedural

In relational calculus, a query is solved by defining a solution relation in a single step

Relational calculus is mainly based on the well-known propositional calculus, which is a method of calculating with sentences or declarations

Various types of relational calculus are:

- Tuple Relational Calculus (TRC)
- Domain Relational Calculus (DRC)

Tuple Relational Calculus (TRC)

Tuple Relational Calculus (TRC)

A tuple variable is a variable that takes on tuples of a particular relation schema as values

A tuple relational calculus query has the form:
{T/ P(T)}
The result of this query is the set of all tuples t for which the formula P(T) evaluates to TRUE with T = t
Sailors (sid, sname, rating, age)

Query: Find all the sailors with a rating above 4
$\{S/S \in Sailors \wedge S.rating > 4\}$

Tuple Relational Calculus (TRC)...

Tuple Relational Calculus (TRC)

Let Rel $\rightarrow$ be a relation name, R, S $\rightarrow$ be the tuple variables, a $\rightarrow$ an attribute of R, b $\rightarrow$ an attribute of S, op $\rightarrow$ operator in the set {<, $\leq$, >, $\geq$, =, $\neq$}. An Atomic formula is one of the following:

- R $\in$ Rel
- R.a op S.b
- R.a op Constant or Constant op R.a

To represent the join and division of relational algebra by relational calculus, we need quantifiers such as: ***existential for join*** *and* ***universal for division***

A quantifier quantifies or indicates the quantity of something

The existential quantifier **($\exists$)** states that at least one instance of a particular type of thing exist
Similarly, the universal quantifier**($\forall$)** states that some condition applies to all or to every row of some type

Tuple Relational Calculus (TRC)...

Tuple Relational Calculus (TRC)

A formula is recursively defined by using the following rules:

- Any atomic formula
- If p and q are formulae, then $\neg$ p, p $\wedge$ q, p $\vee$ q, or p $\Rightarrow$ q are also formulae
- If p is a formula that contains T as a variable, then $\exists$ T(p) and $\forall$ T(p) are also formulae

The quantifiers $\exists$ and $\forall$ are said to **bind** the tuple variable R; whereas a variable is said to be **free** in a formula if the formula does not contain an occurrence of a quantifier that binds it

In most of the queries, the output is shown by using the free variables

Safe Expressions

Safe Expressions

Whenever we use universal quantifiers or existential quantifiers in a calculus expression, we must make sure that the resulting expression makes sense

A safe expression in relational calculus is one that is guaranteed to yield a finite number of tuples as its result; otherwise, the expression is called unsafe

That means, an expression is said to be safe if all values in its result are from the domain of the expression

Queries

Sailor Database

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Query: Find the names & ages of sailors with a rating above 4

{T/∃S ∈ Sailors (S.rating >4 ∧ T.sname=S.sname ∧ T.age= S.age)}

Query: Find the sailor name, boat id & reservation date for each reservation

{T/∃R ∈ Reserves ∃S ∈ Sailors (R.sid = S.sid ∧ T.sname=S.sname ∧ T.bid=R.bid ∧ T.day=R.day)}

Query: Find the names of sailors who have reserved boat 111

{T/∃R ∈ Reserves ∃S ∈ Sailors (R.sid = S.sid ∧ R.bid=111 ∧ T.sname=S.sname)}

Queries...

Query: Find the names of sailors who have reserved a green boat

$\{T/\exists S \in Sailors\ \exists R \in Reserves(R.sid = S.sid \wedge T.sname=S.sname \wedge \exists B \in Boats\ (B.bid=R.bid \wedge B.color='green'))\}$

This query can also be written as:
$\{T/\exists S \in Sailors\ \exists R \in Reserves\ \exists B \in Boats(R.sid = S.sid \wedge B.bid=R.bid \wedge B.color='green' \wedge T.sname=S.sname)\}$

Query: Find the names of sailors who have reserved at least 2 boats

$\{T/\exists S \in Sailors\ \exists R1 \in Reserves\ \exists R2 \in Reserves(S.sid=R1.sid \wedge R1.sid= R2.sid \wedge R1.bid \neq R2.bid \wedge T.sname=S.sname)\}$

Queries...

Query: Find the names of sailors who have reserved all boats

$\{T/\exists S \in Sailors\ \forall B \in Boats(\exists R \in Reserves\ (S.sid=R.sid \wedge R.bid= B.bid \wedge T.sname= S.sname))\}$

Query: Find sailors who have reserved all green boats

$\{S/S \in Sailors \wedge \forall B \in Boats\ (B.color='green' \Rightarrow(\exists\ R \in Reserves\ (S.sid=R.sid \wedge R.bid= B.bid))\}$

Domain Relational Calculus (DRC)

Domain Relational Calculus (DRC)

In tuple relational calculus, the variables range over the tuples whereas in domain relational calculus, the variables range over the domains

The domain variables are the ones which range over the underlying domains instead of over the relations

The domain relational calculus query has the form:
$\{<x_1, x_2, \ldots x_n> \mid P(x_1, x_2, \ldots x_n)\}$
where x_i is either a domain variable or a constant and $P(x_1, x_2, \ldots x_n)$ is the domain relational calculus formula whose only free variables are the variables among the xi, $1 \leq i \leq n$

The result of this query is the set of all tuples $<x_1, x_2, \ldots x_n>$ for which the formula evaluates to TRUE

Domain Relational Calculus (DRC)...

Domain Relational Calculus (DRC)

Let Rel $\rightarrow$ be a relation name, X, Y $\rightarrow$ be the domain variables, op $\rightarrow$ an operator in the set {<, $\leq$, >, $\geq$, =, $\neq$}. An Atomic formula in domain relational calculus is one of the following:

- $<x_1, x_2, \ldots x_n> \in$ Rel
- X op Y
- X op Constant or Constant op X

A formula is recursively defined by using the following rules:

- Any atomic formula
- If p and q are formulae, then $\neg$p, p $\wedge$ q, p $\vee$ q, or p $\Rightarrow$ q are also formulae
- If p is a formula that contains X as a domain variable, then $\exists$X(p) and $\forall$ X(p) are also formulae

The quantifiers $\exists$ & $\forall$ are said to bind the domain variable X. Whereas a variable is said to be free in a formula if the formula does not contain an occurrence of a quantifier that binds it

Queries

Sailor Database

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Query: Find all sailors with a rating above 7

{<I, N, T, A> / <I, N,T, A> $\in$ Sailors $\wedge$ T >7}

Query: Find the names of sailors who reserved boat 111

{<N> /$\exists$I, T , A (<I, N, T, A> $\in$ Sailors $\wedge$ $\exists$<Ir, Br, D> $\in$ Reserves (Ir=I $\wedge$ Br=111))}

This query can also be written as:
{<N> /$\exists$I, T , A (<I, N, T, A> $\in$ Sailors $\wedge$ $\exists$<Ir, Br, D> $\in$ Reserves (Ir=I $\wedge$ Br=111))}
or
{<N> /$\exists$I, T , A (<I, N, T, A> $\in$ Sailors $\wedge$ $\exists$D(<I,111, D> $\in$ Reserves))}

Queries...

Query: Find the names of sailors who have reserved a green boat

{<N>/∃I, T, A (<I, N, T, A> ∈ Sailors ∧ <I, Br, D> ∈ Reserves ∧ ∃<Br, Bn, 'green'> ∈ Boats)}

Query: Find the names of sailors who have reserved at least 2 boats

{<N>/∃I, T, A (<I, N, T, A> ∈ Sailors ∧ ∃Br1, Br2, D1, D2(<I, Br1, D1> ∈ Reserves ∧ <I, Br2, D2> ∈ Reserves ∧ Br1 ≠ Br2))}

Query: Find the names of sailors who have reserved all boats

{<N>/∃I, T, A(<I, N, T, A> ∈ Sailors ∧ ∀<B, Bn, C> ∈ Boats (∃<Ir, Br, D> ∈ Reserves (I=Ir ∧ Br=B)))}

Query: Find sailors who have reserved all green boats

{<I,N,T,A>/<I,N,T,A> ∈ Sailors ∧ ∀<B, Bn, C> ∈ Boats (C='green' ⇒ ∃<Ir, Br, D> ∈ Reserves (I=Ir ∧ Br=B)))}

Database Management System 15

Database Design

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Database Design

Database Design

- First characterize fully the data requirements of the prospective database users, which usually involves in textual descriptions
- Next, choose ER model to translate these requirements into a conceptual schema of the database
- In the logical design phase, map the high level conceptual schema onto the implementation data model of the database system that will be used. The implementation data model is typically the Relational data model
- Finally, use the resulting system specific database schema in the subsequent physical design phase, in which the physical features of the database are specified
- In designing a database schema, the major pitfalls which should be avoided are:
  - redundancy: it means repetition of the information
  - incompleteness: it means certain aspects of the enterprise may not be modeled due to difficulty or complexity

Bad Database Design/Concept of Anomalies

Bad Database Design/Concept of Anomalies

- Database anomalies are the problems in relations that occur due to redundancy in the relations. These anomalies affect the process of inserting, deleting and updating data in the relations
- The intension of relational database theory is to eliminate anomalies from occurring in a database

Student database

Name	Course	Phone_no	Major	Prof	Grade
Mahesh	353	1234	Comp sc	Alok	A
Nitish	329	2435	Chemistry	Pratap	B
Mahesh	328	1234	Comp sc	Samuel	B
Harish	456	4665	Physics	James	A
Pranshu	293	4437	Decision sc	Sachin	C
Prateek	491	8788	Math	Saurav	B
Prateek	356	8788	Math	Sunil	In prog
Mahesh	492	1234	Comp sc	Paresh	In prog
Sumit	379	4575	English	Rakesh	C

Bad Database Design/Concept of Anomalies...

Insertion Anomaly

It is the anomaly in which the user cannot insert a fact about an entity until he/she has an additional fact about another entity. In other words, there are circumstances in which certain facts can not be recorded at all

Ex: *We cannot record a new prof details without assigning a course to him*

Deletion Anomaly

It is the anomaly in which the deletion of facts about an entity automatically deleted the fact of another entity

Ex: *If we want to delete the information about course 293, automatically the information of prof Sachin will be deleted which is not our interest*

Bad Database Design/Concept of Anomalies...

Updation Anomaly

It is the anomaly in which the modification in the value of specific attribute requires modification in all records in which that value occurs. In other words, the same data can be expressed in multiple rows. Therefore, updates to the table may result in logical inconsistencies

Ex: *If the updation to the phone_no of Mahesh is done in a single row only, then the updation process will put the database in an inconsistent state so that the phone_no of Mahesh will give conflicting answers*

Functional Dependency(FD)

Functional Dependency(FD)

- Functional Dependency is the building block of normalization principles
- Attribute(s) *A* in a relation schema *R* functionally determines another attribute(s) *B* in *R* if for a given value a_1 of *A*; there is a single, specific value b_1 of *B* in relation *r* of *R*
- The symbolic expression of this FD is:
  A$\rightarrow$**B**
  where A(LHS of FD) is known as the determinant and B(RHS of FD) is known as the dependent
- *In other words, if A functionally determines B in R, then it is invalid to have two or more tuples that have the same A value, but different B values in R*

Functional Dependency(FD)...

Functional Dependency(FD)

- From Student schema, we can infer that Name$\rightarrow$Phone_no because all tuples of Student with a given Name value also have the same Phone_no value
- Likewise, it can also be inferred that Prof$\rightarrow$Grade. At the same time, notice that Grade does not determine Prof
- When the determinant or dependent in an FD is a composite attribute, the constituent atomic attributes are enclosed by braces as shown in the following example: {Name, Course}$\rightarrow$Phone_no

- *FD is a constraint between two sets of attributes in a relation from a database*

Trivial FDs and Non-Trivial FDs

Trivial FDs

- A functional dependency X$\rightarrow$Y is a trivial functional dependency if Y is a subset of X
- For example, {Name, Course}$\rightarrow$Course. If two records have the same values on both the Name and Course attributes, then they obviously have the same Course
- *Trivial dependencies hold for all relation instances*

Non-Trivial FDs

- A functional dependency X$\rightarrow$Y is called as non-trivial type if Y$\cap$X =Φ
- For example, Prof$\rightarrow$Grade
- *Non-trivial FDs are given implicitly in the form of constraints when designing a database*

Armstrong's Inference Axioms

The inference axioms or rules allow users to infer the FDs that are satisfied by a relation

Let R(X, Y, Z, W) where X, Y, Z, and W are arbitrary subsets of the set of attributes of a universal relation schema R

The three fundamental inference rules are:

- ***Reflexivity Rule***: If Y is a subset of X, then X$\rightarrow$Y (Trivial FDs). Ex:*{Name, Course}$\rightarrow$Course*
- ***Augmentation Rule***: If X$\rightarrow$Y, then {X, Z}$\rightarrow${Y, Z}. Ex: *as Prof$\rightarrow$Grade, therefore {Prof, Major}$\rightarrow${Grade, Major}*
- ***Transitivity Rule***: If X$\rightarrow$Y and Y$\rightarrow$Z, then X$\rightarrow$Z. Ex: *as Course$\rightarrow$Name and Name$\rightarrow$Phone_no functional dependencies are present, therefore Course$\rightarrow$Phone_no*

Armstrong's Inference Axioms...

Armstrong's Inference Axioms

The four secondary inference rules are:

- ***Union or Additive Rule***: If X→Y and X→Z, then X→{Y, Z}. Ex:*as Prof→Grade and Prof→Course FDs are present; therefore, Prof→{Grade, Course}*

- ***Decomposition Rule***: If X→{Y, Z}, then X→Y and X→Z. Ex: *if Prof→{Grade, Course}, then this FD can be decomposed as Prof→Grade and Prof→Course*

- ***Composition Rule***: If X→Y and Z→W, then {X, Z}→{Y, W}. Ex: *if Prof→Grade and Name→Phone_no, then the FDs can be composed as {Prof, Name}→{Grade, Phone_no}*

- ***Pseudotransitivity Rule***: If X→Y and {Y, W}→Z, then {X, W}→Z. Ex: *if Prof→Grade and {Grade, Major}→Course, then the FD {Prof, Major}→Course is valid*

Logical Implication

- Given a relation schema R and a set of functional dependencies F. Let FD X→Y is not in F. *F can be said to logically imply X→Y if for every relation r on the relation schema R that satisfies the FD in F, the relation r also satisfies X→Y*

- F logically implies X→Y is written as:
  F⊨X→Y
- Let R = (A, B, C, D) and F = {A→B, A→C, BC→D}

  F⊨A→D

  Given F = {A→B, C→D} with C⊆B, show that F⊨A→D

Closure of a Set of Functional Dependencies

Closure of a Set of Functional Dependencies

Given a set F of functional dependencies for a relation schema R, we define F^+, the closure of F, to be the set of all functional dependencies that are logically implied by F. Mathematically,

$$F^+ = \{X \rightarrow Y / F \models X \rightarrow Y\}$$

To generate all FDs that can be derived from F, the steps are:

- First, apply the inference axioms to all single attributes and use the FDs of F whenever it is applicable
- Second, apply the inference axioms to all combinations of two attributes and use the functional dependencies of F whenever it is applicable
- Next apply the inference axioms to all combinations of three attributes and use the FDs of F when necessary
- Proceed in this manner for as many different attributes as there are in F

Closure of a Set of Functional Dependencies...

Closure of a Set of Functional Dependencies...

Let R=(A, B, C) and F={A→B, A→C}

F^+ = {A→A, B→B, C→C, A→B, A→C, A→BC, A→AB, A→AC, AB→A, AB→B, AB→AB, AC→A, AC→C, AC→AC, BC→B, BC→C, BC→BC, ABC→AB, ABC→AC, ABC→BC, ABC→ABC}

Ex: R=(W, X, Y) and F={W→X, X→Y, W→XY}

Uses of set of Functional Dependency Closure:

- ***Computing if two sets of functional dependencies F and G are equivalent***: When $F^+=G^+$, then the functional dependencies sets F and G are equivalent

Ex: F={W→X, X→Y, W→XY} and G= {W→X, W→Y, X→Y}

Closure of a Set of Attributes

Closure of a Set of Attributes

Given a set of attributes X and a set of functional dependencies F, then the closure of the set of attributes X under F, denoted as X^+, is the set of attributes A that can be derived from X by applying the Armstrong's Inference Axioms to the functional dependencies of F

The closure of X is always a non empty set

R = (A, B, C, D) and F = {A$\rightarrow$C, B$\rightarrow$D}

$\{A\}^+$ = {A, C}, $\{B\}^+$ = {B, D}, $\{C\}^+$={C}, $\{D\}^+$={D},
$\{A, B\}^+$ = {A, B, C, D}, $\{A, C\}^+$ = {A, C}, $\{A, D\}^+$ = {A, C, D},
$\{B, C\}^+$ = {B, C, D}, $\{B, D\}^+$ = {B, D}, $\{C, D\}^+$ = {C, D},
$\{A, B, C\}^+$ = {A, B, C, D}, $\{A, B, D\}^+$ = {A, B, C, D},
$\{B, C, D\}^+$ = {B, C, D}, $\{A, B, C, D\}^+$ = {A, B, C, D}

Ex: R = (X, Y, Z)and F = {X$\rightarrow$Y, Y$\rightarrow$Z}

Closure of a Set of Attributes...

Uses of Attribute Closure:

- ***Testing for key***: To test whether X is a key or not, X^+ is computed. X is a key iff X^+ contains all the attributes of R. X is a candidate key if none of its subsets is a key
- ***Testing functional dependencies***: To check whether a functional dependency X$\rightarrow$Y holds or not, just check if Y$\subseteq$ X^+

Given R=(A, B, C, D) and F={AB$\rightarrow$C, B$\rightarrow$D, D$\rightarrow$B}, find the candidate keys of the relation. How many candidate keys are in this relation?

Redundancy of FDs

Redundancy of FDs

Given a set of functional dependencies F, a functional dependency A→B of F is said to be redundant with respect to the FDs of F if and only if A→B can be derived from the set of FDs F - {A→B}
Eliminating redundant functional dependencies allows us to minimize the set of FDs
Ex: A→C is redundant in {A→B, B→C, A→C}

Redundant attribute on RHS

In a functional dependency, some attributes in the RHS may be redundant
Ex: F= {A→B, B→C, A→{C, D}} can be simplified to {A→B, B→C, A→D}

Redundant attribute on LHS

In a functional dependency, some attributes in the LHS may be redundant
Ex: F = {A→B, B→C, {A, C}→D} can be simplified to {A→B, B→C, A→D}

Canonical Cover/Minimal Cover

For a given set F of FDs, a canonical cover, denoted by F_c, is a set of FDs where the following conditions are satisfied:

- F and F_c are equivalent
- Every FD of F_c is simple. That is, the RHS of every functional dependency of F_c has only one attribute
- No FD in F_c is redundant
- The determinant or LHS of every FD in F_c is irreducible

R = (A, B, C) and F = {A$\rightarrow${B, C}, B$\rightarrow$C, A$\rightarrow$B, {A, B}$\rightarrow$C}

F_c= {A$\rightarrow$B, B$\rightarrow$C}

As canonical cover contains the functional dependencies without any redundancy, therefore finding the key of the relation becomes efficient

Database Management System 16

Normalization

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Normalization

Normalization is the process of decomposing or breaking a relation schema R into fragments (i.e. smaller schemas) R_1, R_2, ... R_n such that the following conditions hold:

- ***Lossless decomposition***: The fragments should contain the same information as the original relation
- ***Dependency preservation***: All the functional dependencies should be preserved within each fragment R_i
- ***Good form***: Each fragment R_i should be free from any type of redundancy

In other words, normalization is the process of refining the relational data model. It is used because of the following reasons:

- It improves database design
- It ensures minimum redundancy of data
- It removes anomalies for database activities

Lossless Decomposition

Lossless Decomposition

The decomposition of a base relation is said to be lossless if the original relation can be recovered back by joining the fragment relations
Let R be the base relation, which is decomposed into R_1, R_2, ... R_n. This decomposition is lossless iff R=$R_1 \bowtie R_2 \bowtie ... \bowtie R_n$. In other words, the consecutive fragments are interrelated by the primary key and foreign key relationships

R

A	B
a	1
a	2
b	1
b	2

R_1 R_2

A	B
a	1
b	2

$R_1 \bowtie R_2$

A	B
a	1
a	2
b	1
b	2

As $R_1 \bowtie R_2 = R$, the decomposition is lossless

Lossless Decomposition...

R

A	B
a	1
a	2
b	1

R_1

A
a
b

R_2

B
1
2

$R_1 \bowtie R_2$

A	B
a	1
a	2
b	1
b	2

As $R_1 \bowtie R_2 \neq R$, the decomposition is lossy

Lossless Decomposition...

Lossless Decomposition...

When the base relation schema is decomposed into the fragmented relation schemas, the consecutive relations should be related by primary key - foreign key pair on the common column; so that natural join be possible on the common column. Thus, we have to check whether the common column is the key of any relation or not:

- If the common column is the key, the decomposition is lossless
- If the common column is not the key, the decomposition is lossy

Ex: R(A, B, C, D) with F={A$\rightarrow$B, B$\rightarrow$C, C$\rightarrow$D} is decomposed to R_1(A, B, C) and R_2(C, D) with F_1={A$\rightarrow$B, B$\rightarrow$C} and F_2={C$\rightarrow$D} respectively

Ex: R(A, B, C, D) with F={A$\rightarrow$B, B$\rightarrow$C, D$\rightarrow$C} is decomposed to R_1(A, B, C) and R_2(C, D) with F_1={A$\rightarrow$B, B$\rightarrow$C} and F_2={D$\rightarrow$C} respectively

Lossless - Join Algorithm

Lossless - Join Algorithm

This algorithm is used to check whether the decomposition of a relation is lossless or lossy type. The steps are:

- Construct a table with n columns, where n is the number of attributes in the original relation and k rows, where k is the number of decomposed relations. Label the columns as A_1, A_2, ... A_n
- Fill the entries in this table as follows: for each attribute A_i, check if this attribute is one of the attributes of the relation schema R_j
  - If attribute A_i is in the Relation R_j, then the entry (A_i, R_j) of the table will be a_i
  - If attribute A_i is not an attribute in the relation R_j, then the entry (A_i, R_j) of the table will be b_{ij}
- For each of FD X$\rightarrow$Y of F, do the following until it is not possible to make any more changes to the table. When the table no longer changes, continue with step4

Lossless - Join Algorithm...

Lossless - Join Algorithm...

- .
  - If there are two or more rows with the same value under the attribute or attributes of the determinant X, then make equal their entries under attribute Y (i.e. RHS of FD)
    - When making equal two or more entries under any column, if one of them is a_i, then make all of them a_i
    - If none of them is a_i, then make all the entries uniform by considering one b term. If they are b_{ij} and b_{kl}, chose one of these two values as the representative value and make the other values equal to it. That means, make all the entries as either b_{ij} or b_{kl}. Continue with step3
  - If there are no two rows with the same value under the attribute or attributes of the determinant X, continue with step3
- Check the rows of the table. If there is a row with its entries equal to a_1, a_2 ... a_n, then the decomposition is lossless. Otherwise, the decomposition is lossy

Lossless - Join Algorithm...

Lossless - Join Algorithm...

R=(X, Y, Z), F={X$\rightarrow$Y, Z$\rightarrow$Y }, R_1=(X, Y), R_2=(Y, Z)

	A_1(X)	A_2(Y)	A_3(Z)
R_1			
R_2			

	A_1(X)	A_2(Y)	A_3(Z)
R_1	a_1	a_2	b_{31}
R_2	b_{12}	a_2	a_3

For X$\rightarrow$Y, the table remains unchanged

For Z$\rightarrow$Y, the table remains unchanged

Decomposition is lossy type

Lossless - Join Algorithm...

Lossless - Join Algorithm...

R=(X, Y, Z), F={X$\rightarrow$Y, Y$\rightarrow$Z }, R_1=(X, Y), R_2=(Y, Z)

	A_1(X)	A_2(Y)	A_3(Z)
R_1			
R_2			

	A_1(X)	A_2(Y)	A_3(Z)
R_1	a_1	a_2	b_{31}
R_2	b_{12}	a_2	a_3

For X$\rightarrow$Y, the table remains unchanged

For Y$\rightarrow$Z, the table will be modified

	A_1(X)	A_2(Y)	A_3(Z)
R_1	a_1	a_2	a_3
R_2	b_{12}	a_2	a_3

Decomposition is lossless type

Dependency Preservation

Dependency Preservation

The decomposition of a relation schema R with FDs F is a set of fragment relations R_i with FDs F_i, where F_i is the subset of dependencies in F^+ that include only attributes in R_i. The decomposition is dependency preserving if and only if

$(\cup_i F_i)^+ = F^+$

R = (A, B, C) and F={A$\rightarrow$B, B$\rightarrow$C, A$\rightarrow$C}

R

A	B	C
1	2	3
2	2	3
3	2	3
4	3	4

Key=A

Dependency Preservation...

R_1 = (A, B) and F={A→B}

A	B
1	2
2	2
3	2
4	3

R_2 = (A, C) and F={A→C}

A	C
1	3
2	3
3	3
4	4

The decomposition is lossless because $R_1 \bowtie R_2$ = R

But, the decomposition is not dependency preserving because $(F_1 \cup F_2)^+$={A→B, A→C}

Thus, $(F_1 \cup F_2)^+ \neq F^+$ as we lost the FD B→C

Dependency Preservation...

R_1 = (A, B) and F={A→B}

A	B
1	2
2	2
3	2
4	3

R_2 = (B, C) and F={B→C}

B	C
2	3
3	4

The decomposition is lossless because $R_1 \bowtie R_2$ = R

Similarly, the decomposition is dependency preserving because $(F_1 \cup F_2)^+$={A→B, B→C, A→C}

Thus, $(F_1 \cup F_2)^+ = F^+$

Guidelines followed in Designing Good Database

Guideline 1

Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity sets and relationship sets into a single relation

If a relation schema corresponds to one entity set or one relationship set, then the meaning tends to be simple and clear. Otherwise, the relation corresponds to a mixture of multiple entities and relationships and hence becomes complex and unclear

Only foreign keys should be used to refer to other entities. Entity and relationship attributes should be kept apart as much as possible

Bottom Line: *design the schemas that can be explained easily relation by relation. In such cases, the semantics of attributes should be easy to interpret*

Guidelines followed in Designing Good Database...

Guideline 2

Design the base relation schemas in such a way that the anomalies such as insertion, deletion, or updation anomalies are removed from the relations

If any anomalies are present, note them clearly and make sure that the programs that modify (update) the database will operate correctly

Guideline 3

Avoid placing attributes in a base relation whose values may frequently be NULL

If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation

Attributes that are NULL frequently could be placed in separate relations (with the primary key)

Guidelines followed in Designing Good Database...

Guideline 4

Design the relation schemas so that they can be joined in a such a way that no spurious tuples are generated

Avoid relations that contain matching attributes that are not (foreign key and primary key) combinations, because joining on such attributes may produce spurious tuples

Database Management System 17

Normal Forms

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Normal Forms

Normal Forms

- Normal forms provide a stepwise progression towards the construction of normalized relation schemas, which are free from data redundancies
- A relation schema is said to be in a particular normal form if it is satisfying certain defined conditions

1NF(First Normal Form)

A relation is in 1NF iff the values in the relation are atomic and single-valued for every attribute in the relation

Course

Module	Dept	Lecturer	Text
M_1	D_1	L_1	T_1, T_2
M_2	D_1	L_1	T_1, T_3
M_3	D_1	L_2	T_4
M_4	D_2	L_3	T_1, T_5
M_5	D_2	L_4	T_6

- As the Text column values are not atomic, this relation is not present in 1NF
- To convert this non-1NF relation into a 1NF relation, split up the non-atomic values

1NF(First Normal Form)...

Course1

Module	Dept	Lecturer	Text
M_1	D_1	L_1	T_1
M_1	D_1	L_1	T_2
M_2	D_1	L_1	T_1
M_2	D_1	L_1	T_3
M_3	D_1	L_2	T_4
M_4	D_2	L_3	T_1
M_4	D_2	L_3	T_5
M_5	D_2	L_4	T_6

Course2

Module	Dept	Lecturer	$Text_1$	$Text_2$
M_1	D_1	L_1	T_1	T_2
M_2	D_1	L_1	T_1	T_3
M_3	D_1	L_2	T_4	
M_4	D_2	L_3	T_1	T_5
M_5	D_2	L_4	T_6	

Corollary: As the relation schema contains no data values, therefore all relation schemas are in 1NF

Anomalies in 1NF Relations:

- Insertion anomalies
- Updation anomalies
- Deletion anomalies

Partial FD

Partial FD

A FD A $\rightarrow$ B is a partial FD, if some attribute of A can be removed and the FD still holds. That means there is some proper subset of A, C $\subset$ A, such that C $\rightarrow$ B

- **Key attributes**: are the attributes which are part of some candidate key
- **Non-key attributes**: are the attributes which are not part of any candidate key

2NF(Second Normal Form)

2NF(Second Normal Form)

A relation is in 2NF iff the following two conditions are met simultaneously:

- It is in 1NF
- No non-key attribute is partially dependent on any key

A non-2NF relation can be decomposed into 2NF relations by:

- Create a new relation by using the attributes from the offending FD as the attributes in the new relation. The determinant of the FD becomes the primary key of the new relation
- The attribute on the RHS of the FD is then eliminated from the original relation
- If more than one FD prevents the relation from being 2NF, repeat steps 1 and 2 for each offending FD
- If the same determinant appears in more than one FD, place all the attributes functionally dependent on this determinant as non-key attributes in the relation having the determinant as the primary key

2NF(Second Normal Form)...

Course

Module	Dept	Lecturer	Text
M_1	D_1	L_1	T_1
M_1	D_1	L_1	T_2
M_2	D_1	L_1	T_1
M_2	D_1	L_1	T_3
M_3	D_1	L_2	T_4
M_4	D_2	L_3	T_1
M_4	D_2	L_3	T_5
M_5	D_2	L_4	T_6

F={Module$\rightarrow$Dept, Module$\rightarrow$Lecturer, Lecturer$\rightarrow$Dept, {Module, Text}$\rightarrow${Dept, Lecturer}}
Here, Key:{Module, Text}

2NF(Second Normal Form)...

Course1

Module	Dept	Lecturer
M_1	D_1	L_1
M_2	D_1	L_1
M_3	D_1	L_2
M_4	D_2	L_3
M_5	D_2	L_4

F_1={Module$\rightarrow${Dept, Lecturer}, Lecturer$\rightarrow$Dept}

Course2

Module	Text
M_1	T_1
M_1	T_2
M_2	T_1
M_2	T_3
M_3	T_4
M_4	T_1
M_4	T_5
M_5	T_6

F_2={{Module, Text}}$\rightarrow${{Module, Text}}

2NF(Second Normal Form)...

Corollary: If the primary key has a single attribute, then the relation is in 2NF

Anomalies in 2NF Relations:

- Insertion anomalies
- Updation anomalies
- Deletion anomalies

Q: R=(A, B, C, D, E), & F={A $\rightarrow$ {B, C, D, E}, {A, B} $\rightarrow$ {C, D, E}, C $\rightarrow$ E, D $\rightarrow$ E}

Q: R=(A, B, C, D, E), & F={{A, B} $\rightarrow$ {C, D, E}, B $\rightarrow$ C, A $\rightarrow$ D}

Transitive FD

Transitive FD

A FD A $\rightarrow$ C is a transitive FD, if there are some set of attributes B such that A $\rightarrow$ B and B $\rightarrow$ C are non-trivial FDs

A $\rightarrow$ B non-trivial means B is not a subset of A

3NF(Third Normal Form)

3NF(Third Normal Form)

A relation is in 3NF iff the following two conditions are satisfied simultaneously:

- It is in 2NF
- No non-key attribute is transitively dependent on the key

The process of decomposing the non-3NF relation into 3NF relations is similar to the process of decomposing the non-2NF relation to 2NF relations

Course

Module	Dept	Lecturer
M_1	D_1	L_1
M_2	D_1	L_1
M_3	D_1	L_2
M_4	D_2	L_3
M_5	D_2	L_4

F={Module→{Dept, Lecturer}, Lecturer→Dept}

This relation is not present in 3NF because Module → Lecturer and Lecturer → Dept

3NF(Third Normal Form)...

Course1

Module	Lecturer
M_1	L_1
M_2	L_1
M_3	L_2
M_4	L_3
M_5	L_4

F_1={Module$\rightarrow$Lecturer}

Course2

Lecturer	Dept
L_1	D_1
L_2	D_1
L_3	D_2
L_4	D_2

F_2={Lecturer$\rightarrow$Dept}

Corollary: A 2NF relation is in 3NF if no non-key attribute functionally determines any other non-key attribute

3NF(Third Normal Form)...

The 3NF helped us to get rid of the anomalies caused by dependencies of a non-key attribute on another non-key attribute

However, relations in 3NF are still susceptible to anomalies when the relations have two overlapping candidate keys or when non-key attribute functionally determines a key attribute. Overlapping candidate keys means composite candidate keys with at least one attribute in common among themselves

Note: A database should normally be in 3NF at least

Q: Lecturer = (lectid, lectname, courseid, coursename) & F={lectid $\rightarrow$ lectname, lectid $\rightarrow$ courseid, lectid $\rightarrow$ coursename, courseid $\rightarrow$ coursename}

Q: R=(B, C, E), F= {E $\rightarrow$ B, {B,C} $\rightarrow$ E}

Q: Store = (order, product, customer, address, qty, unitprice) & F= {order $\rightarrow$ {customer, address}, customer $\rightarrow$ address, product $\rightarrow$ unitprice, {order, product} $\rightarrow$ qty}

BCNF (Boyce-Codd Normal Form)

BCNF (Boyce-Codd Normal Form)

A relation is in BCNF iff the following two conditions are satisfied simultaneously:

- It is in 3NF
- If for every non-trivial functional dependency, the determinant is a key

The process of decomposing the non-BCNF relation into BCNF relations is a simple process. For each non-trivial FD where the determinant is not the key, construct new relations

Student

Student	Course	Time
Rahul	Database	12 : 00
Pratik	Database	12 : 00
Praveen	Database	15 : 00
Praveen	Programming	10 : 00
Rajib	Programming	10 : 00
Shivam	Programming	13 : 00

F={{Student, Course} $\rightarrow$ Time, Time $\rightarrow$ Course, {Student, Time} $\rightarrow$ Course}
Key={Student, Course} and {Student, Time}

BCNF (Boyce-Codd Normal Form)...

BCNF (Boyce-Codd Normal Form)...

This relation is not present in BCNF as in FD Time $\rightarrow$ Course; the determinant {Time} is not a key

After the conversion of this relation to BCNF, create a new relation R_1=(Time, Course) with set of FDs F_1={Time $\rightarrow$ Course}

The original relation is changed to R=(Student, Time) as {Student, Time} set is also the key of the relation

Here, we have lost the functional dependency {Student, Course} $\rightarrow$ Time

BCNF (Boyce-Codd Normal Form)...

Corollary: If a relation has only one candidate key, then 3NF and BCNF are same. That means if a relation is in 3NF having only one candidate key, then it is also present in BCNF

Note: Normalization to 3NF is always lossless and dependency preserving. But, normalization to BCNF is lossless, but may not preserve all the functional dependencies

Q: R =(A, B, C, D, E), F={A $\rightarrow$ {B, E}, C $\rightarrow$ D}. Decompose the relation to BCNF

Q: R=(A, B, C, D), F={{A, B} $\rightarrow$ {C, D}, C $\rightarrow$ B}. Decompose the relation into BCNF

Q: R =(A, B, C, D, E, G), F={{A, B} $\rightarrow$ {C, D}, {B, C} $\rightarrow$ {D, A}, C $\rightarrow$ G, B $\rightarrow$ E}. Decompose this relation to BCNF

Q: R =(A, B, C, D) & F= {{A, C} $\rightarrow$ {B, D}, {B, C} $\rightarrow$ {D, A}, A $\rightarrow$ B, B $\rightarrow$ A}. Decompose this relation to BCNF

Database Management System 18

Advanced Normal Forms

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

MVD(Multi-Valued Dependency)

MVD(Multi-Valued Dependency)

A table involves a multi-valued dependency if it may contain multiple values for an entity

A multi-valued dependency A $\rightarrow\rightarrow$ B exists iff for every occurrence of A; there exists multiple occurrences of B

If A $\rightarrow\rightarrow$ B and A $\rightarrow\rightarrow$ C, then we have a single attribute A which multi-determines two other attributes, B and C

Multi-valued dependencies are also referred to as tuple generating dependencies

MVD(Multi-Valued Dependency)...

Employee

Name	Project	Hobby
Asis	Microsoft	Reading
Asis	Oracle	Music
Asis	Microsoft	Music
Asis	Oracle	Reading
Bikash	Intel	Movies
Bikash	Sybase	Riding
Bikash	Intel	Riding
Bikash	Sybase	Movies

MVDs are: Name $\rightarrow\rightarrow$ Project and Name $\rightarrow\rightarrow$ Hobby

MVD

An MVD X $\rightarrow\rightarrow$ Y in relation R is called a trivial MVD if:

- Y is a subset of X, or
- X $\cup$ Y = R

An MVD that satisfies neither the first nor the second condition is called a nontrivial MVD

Normally, MVDs exist in pair

4NF(Fourth Normal Form)

4NF(Fourth Normal Form)

A relation is in 4NF iff the following two conditions are satisfied simultaneously:

- It is in 3NF
- It contains no multiple MVDs

In other words, a relation is in 4NF iff:

- There are no nontrivial MVDs in the relation, or
- The determinant of any nontrivial MVD in the relation is a key

The previous relation is not in 4NF

Employee_Project

Name	Project
Asis	Microsoft
Asis	Oracle
Bikash	Intel
Bikash	Sybase

Name $\rightarrow\rightarrow$ Project

Employee_Hobby

Name	Hobby
Asis	Reading
Asis	Music
Bikash	Riding
Bikash	Movies

Name $\rightarrow\rightarrow$ Hobby

These relations are present in 4NF because the MVDs are

JD(Join Dependency)

JD(Join Dependency)

A relation R satisfies join dependency (R_1, R_2 ... R_n) iff:

- R is equal to the join of R_1, R_2 ... R_n on the common attributes, where R_i is a subset of the relation R

That means R satisfies join dependency iff R= $R_1 \bowtie R_2 \bowtie ... \bowtie R_n$

In other words, a join dependency is said to hold over a relation R if R_1, R_2 ... R_n is a lossless-join decomposition of R

5NF(Fifth Normal Form)

5NF(Fifth Normal Form)

A relation is in 5NF iff the following two conditions are satisfied simultaneously:

- It is in 4NF
- Every join dependency is implied by the candidate keys

In other words, a relation is in 5NF if it is in 4NF and the decomposition is lossless type

Dealer

Dealer	Parts	Customer
D_1	P_1	C_1
D_1	P_1	C_2
D_1	P_2	C_1
D_2	P_1	C_1

Dealer $\rightarrow\!\!\rightarrow$ Parts, Dealer $\rightarrow\!\!\rightarrow$ Customer

Dealer_Parts

Dealer	Parts
D_1	P_1
D_1	P_2
D_2	P_1

Dealer $\rightarrow\!\!\rightarrow$ Parts

Dealer_Customer

Dealer	Customer
D_1	C_1
D_1	C_2
D_2	C_1

Dealer $\rightarrow\!\!\rightarrow$ Customer

This decomposition is not in 5NF

5NF(Fifth Normal Form)...

Dealer_Parts

Dealer	Parts
D_1	P_1
D_1	P_2
D_2	P_1

Dealer $\rightarrow\rightarrow$ Parts

Dealer_Customer

Dealer	Customer
D_1	C_1
D_1	C_2
D_2	C_1

Dealer $\rightarrow\rightarrow$ Customer

Parts_Customer

Parts	Customer
P_1	C_1
P_1	C_2
P_2	C_1

Parts $\rightarrow\rightarrow$ Customer

Dealer_Parts $\bowtie$ Parts_Customer $\bowtie$ Dealer_Customer = Dealer

Thus, decomposition of Dealer to Dealer_Parts, Parts_Customer and Dealer_Customer is in 4NF as well as in 5NF

Denormalization

Denormalization

- **Advantages of normalization**:
  - It removes data redundancy
  - It solves Insertion, Updation, and Deletion anomalies
  - This makes it easier to maintain in the database in a consistent state
- **Disadvantages of normalization**:
  - It leads to more tables in the database
  - For retrieving the records or information, these tables need to be joined back together, which is an expensive task
- Thus, sometimes it is worth denormalizing

Denormalization...

Denormalization...

Once a normalized database design has been achieved, adjustments can be made with the potential consequences (anomalies) in mind

Possible denormalization steps include the following:

- Recombining relations that were split to satisfy normalization rules
- Storing redundant data in tables
- Storing summarized data in tables

Denormalization is the opposite of Normalization

It is the process of increasing redundancy in the database either for convenience or to improve performance

Database Management System 19

Transactions

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

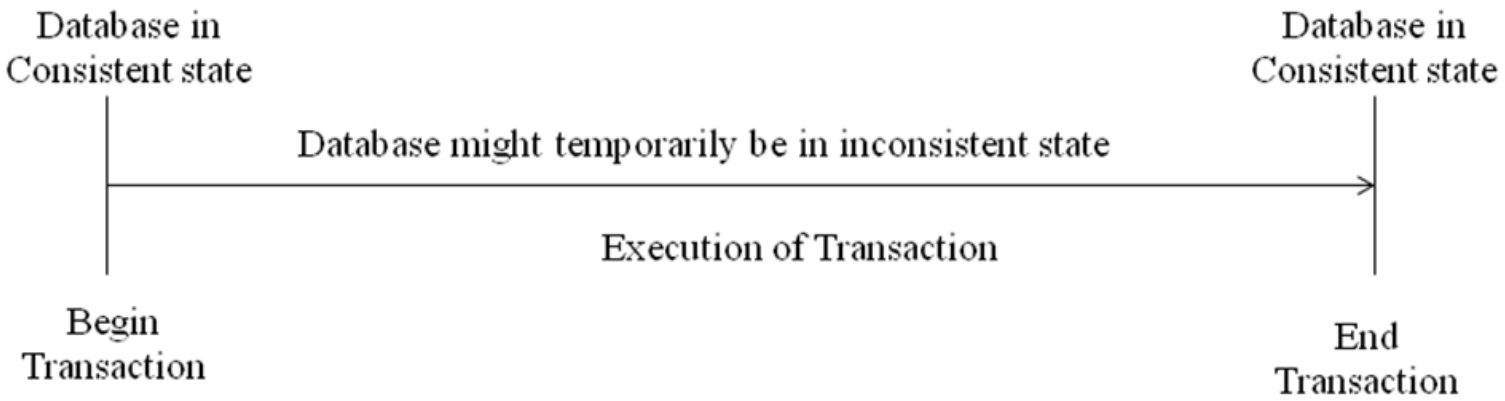
Transaction Concept

Transaction Concept

A transaction is a unit of program execution that accesses and possibly updates various data items

A transaction is a logical unit of work that contains one or more SQL statements. The effects of all the SQL statements in a transaction can be either all committed or all rolled back

A transaction that changes the contents of the database must alter the database from one consistent database state to another



ACID Properties

ACID Properties

- Atomicity: Either all operations of the transaction are reflected properly in the database, or none are. Atomicity requires that all operations of a transaction be completed; if not, the transaction is aborted by rolling back all the updates done during the transaction

- Consistency: Consistency means execution of a transaction should preserve the consistency of the database, i.e. a transaction must transform the database from one consistent state to another consistent state

- Isolation: Though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in the system

ACID Properties...

ACID Properties...

- Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. Durability ensures that once transaction changes are done or committed, they can't be undone or lost, even in the event of a system failure

- The transactions access data item X using the following two operations:
  - ***Read(X)***: It transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation
  - ***Write(X)***: It transfers the data item X from the local buffer of the transaction that executed the write back to the database

ACID Properties...

ACID Properties...

Let T_1 be a transaction that transfers \$100 from account A to account B

T_1
Read(A);
A:=A-100;
Write(A);
Read(B);
B:=B+100;
Write(B);

1. Atomicity: The database system keeps track of the old values of any data on which a transaction performs a write, and if the transaction does not complete its execution, the database system restores the old values to make it appear as though the transaction never executed

- Ensuring atomicity is the responsibility of the database system itself. It is handled by the transaction management component

ACID Properties...

ACID Properties...

2. Consistency: Sum of A and B be unchanged by the execution of the transaction

- Ensuring the consistency for an individual transaction is the responsibility of the application programmers who codes the transaction

3. Isolation: The database is temporarily inconsistent while the transaction to transfer funds from account A to B is executing. The solutions are:

- Execute transactions serially
- However, concurrent execution of transactions provides significant performance benefits such as increased throughputs
- Ensuring the isolation property is the responsibility of concurrency control component of the database system

4. Durability: Once a transaction completes successfully, all the updates that is carried out on the database persist, even if there is a system failure after the transaction completes execution

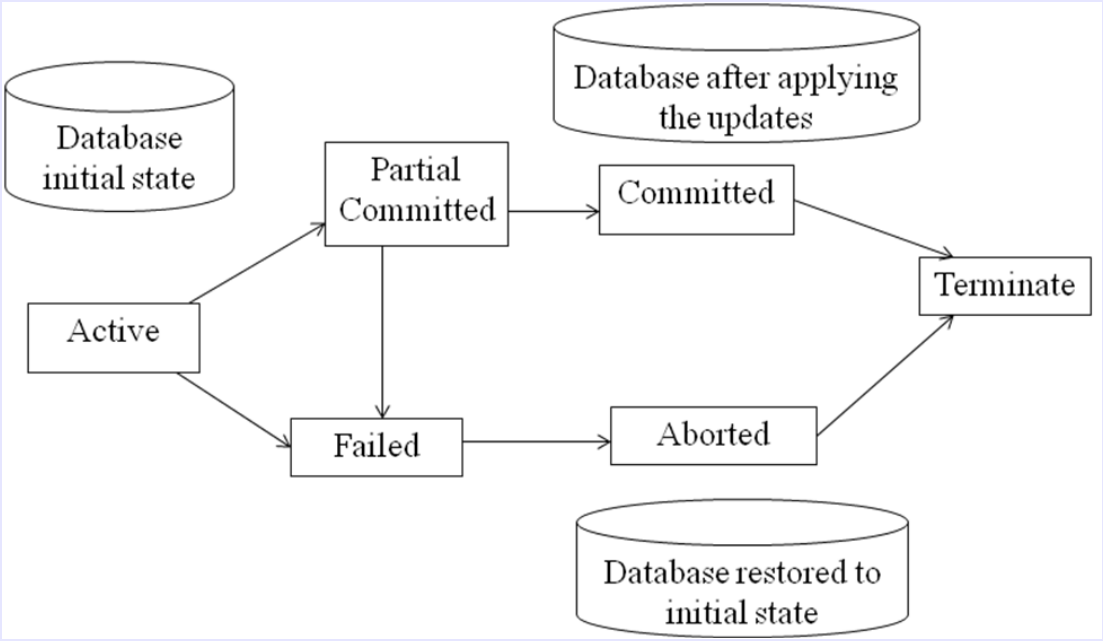
Transaction States

Transaction States

- **Active state**: This state is the initial state of a transaction. The transaction stays in this state while it is executing
- **Partial Committed state**: A transaction is partial committed after its final statement has been executed. A transaction enters this state immediately before the commit statement
- **Failed state**: A transaction enters the failed state after the discovery that normal execution can no longer proceed
- **Aborted state**: A transaction is aborted after it has been rolled back and the database is restored to its prior state before the transaction
- **Committed state**: Committed state occurs after successful completion of the transaction

Transaction States...

Transaction States...



When a transaction enters the aborted state, the system has two options:

- **Restart the transaction**: If the transaction was aborted as a result of a hardware failure or some software error (other than logical error), it can be restarted
- **Kill the transaction**: If the application program that initiated the transaction has some logical error

Database Management System 20

Concurrent Execution

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Concurrent Execution

Concurrent Execution

Concurrent execution of transactions means executing more than one transaction at the same time

In the serial execution, one transaction can start executing only after the completion of the previous

The advantages of using concurrent execution of transactions are:

- Improved throughput and resource utilization
- Reduced waiting time

The database system must control the interaction among the concurrent transactions to prevent them from destroying the consistency of the database. It does this through a variety of mechanisms called concurrency control schemes

Concurrent Execution...

T_1 transfers Dollar \$100 from account A to account B

T_1
Read(A);
A:=A-100;
Write(A);
Read(B);
B:=B+100;
Write(B);

T_2 transfers 20% of balance from account A to account B

T_2
Read(A);
Temp=0.2*A;
A:=A-Temp;
Write(A);
Read(B);
B:=B+Temp;
Write(B);

Schedules

Schedules

A schedule is a sequence that indicates the chronological order in which instructions of concurrent transactions are executed

A schedule for a set of transactions must consist of all instructions of those transactions

We must preserve the order in which the instructions appear in each individual transaction

Serial Schedule

Serial Schedule

A serial schedule is a schedule where all the instructions belonging to each transaction appear together

There is no interleaving of transaction operations. A serial schedule has no concurrency and therefore it does not interleave the actions of different transactions

For n transactions, there are exactly n! different serial schedules possible

Serial Schedule...

Schedule1 (T_1 followed by T_2)

T_1	T_2
Read(A); A:=A-100; Write(A); Read(B); B:=B+100; Write(B);	Read(A); Temp=0.2*A; A:=A-Temp; Write(A); Read(B); B:=B+Temp; Write(B);

Schedule2 (T_2 followed by T_1)

T_1	T_2
Read(A); A:=A-100; Write(A); Read(B); B:=B+100; Write(B);	Read(A); Temp=0.2*A; A:=A-Temp; Write(A); Read(B); B:=B+Temp; Write(B);

Concurrent Schedule

Concurrent Schedule

In concurrent schedule, operations from different concurrent transactions are interleaved

The number of possible schedules for a set of n transactions is much larger than n!

Schedule3

T_1	T_2
Read(A);	
A:=A-100;	
Write(A);	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
	Write(A);
Read(B);	
B:=B+100;	
Write(B);	
	Read(B);
	B:=B+Temp;
	Write(B);

Concurrent Schedule...

Schedule4

T_1	T_2
Read(A);	
A:=A-100;	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
	Write(A);
	Read(B);
Write(A);	
Read(B);	
B:=B+100;	
Write(B);	
	B:=B+Temp;
	Write(B);

Schedule5

T_1	T_2
Read(A);	
A:=A-100;	
Write(A);	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
Read(B);	
B:=B+100;	
	Write(A);
	Read(B);
	B:=B+Temp;
	Write(B);
Write(B);	

Serializability

Serializability

A concurrent schedule is serializable if it is equivalent to a serial schedule

Serial schedules preserve consistency as we assume each transaction individually preserves consistency

The database system must control concurrent execution of transactions to ensure that the database state remains consistent

Since the modifications are done in the local buffer, we can ignore the operations other than Read and Write instructions for easier understanding of the serializability

Serializability...

T_1	T_2
Read(A);	
Write(A);	
Read(B);	
Write(B);	
	Read(A);
	Write(A);
	Read(B);
	Write(B);
Schedule1	

T_1	T_2
	Read(A);
	Write(A);
	Read(B);
	Write(B);
Read(A);	
Write(A);	
Read(B);	
Write(B);	
Schedule2	

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
	Write(A);
Read(B);	
Write(B);	
	Read(B);
	Write(B);
Schedule3	

T_1	T_2
Read(A);	
	Read(A);
	Write(A);
	Read(B);
Write(A);	
Read(B);	
Write(B)	
	Write(B);
Schedule4	

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
Read(B);	
	Write(A);
	Read(B);
	Write(B);
Write(B);	
Schedule5	

Conflict Serializability

Conflict Serializability

Conflict Serializability consists of conflicting operations
Let us consider a schedule S in which there are two consecutive instructions, I_i and I_j of transactions T_i and T_j respectively (i≠j)

If I_i and I_j access different data items, then we can swap I_i and I_j without affecting the results of any transactions in the schedule. However, if I_i and I_j access the same data item Q, then the order of the two instructions may matter:

- **Case-1: I_i=Read(Q) and I_j=Read(Q)**:
  - Order of I_i and I_j does not matter
- **Case-2: I_i=Read(Q) and I_j=Write(Q)**:
  - Order of I_i and I_j matters in a schedule
- **Case-3: I_i=Write(Q) and I_j=Read(Q)**:
  - Order of I_i and I_j matters in a schedule
- **Case-1: I_i=Write(Q) and I_j=Write(Q)**:
  - Order of I_i and I_j matters in a schedule

Conflict Serializability...

Conflict Serializability...

Thus, I_i and I_j conflict if they are the instructions by different transactions on the same data item, and at least one of these instructions is a write operation

Let I_i and I_j be consecutive instructions of a schedule S. If I_i and I_j are instructions of different transactions and they do not conflict, then we can swap the order of I_i and I_j to produce a new schedule S'. Here, we expect S to be equivalent to S'

If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are conflict equivalent

A concurrent schedule S is conflict serializable if it is conflict equivalent to a serial schedule

Conflict Serializability...

T_1	T_2
Read(A);	
Write(A);	
Read(B);	
Write(B);	
	Read(A);
	Write(A);
	Read(B);
	Write(B);
Schedule3	

T_1	T_2
Read(A);	
	Read(A);
	Write(A);
Write(A);	
	Read(B);
Read(B);	
Write(B)	
	Write(B);
Schedule4	

T_1	T_2
Read(A);	
Write(A);	
Read(B);	
	Read(A);
	Write(A);
	Read(B);
	Write(B);
Write(B);	
Schedule5	

Conflict Serializability...

It is possible to have two schedules that produce the same outcome, but that are not conflict equivalent

T_1	T_5
Read(A); A:=A-100; Write(A); Read(B); B:=B+100; Write(B);	 Read(B); B:=B-200; Write(B); Read(A); A:=A+200; Write(A);
Schedule6	

T_1	T_5
Read(A); Write(A); Read(B); Write(B);	 Read(B); Write(B); Read(A); Write(A);
Schedule6	

Testing for Conflict Serializability

Testing for Conflict Serializability

Construct a directed graph, called a precedence graph from S. This graph consists of a pair G = (V, E), where V is a set of vertices and E is a set of edges

The set of vertices consists of all the transactions participating in the schedule

The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds:

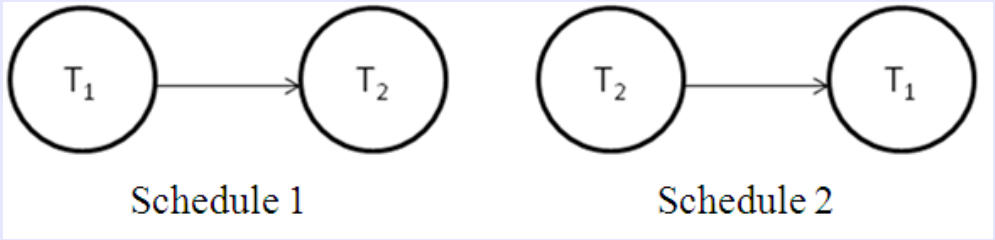
- T_i executes write(Q) before T_j executes read(Q)
- T_i executes read(Q) before T_j executes write(Q)
- T_i executes write(Q) before T_j executes write(Q)

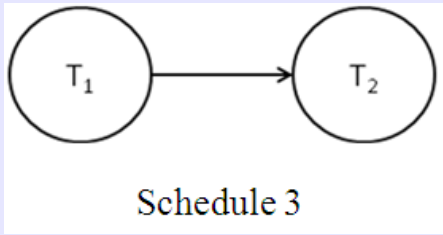
If an edge $T_i \rightarrow T_j$ exists in the precedence graph, then in any serial schedule S' equivalent to S, T_i must appear before T_j

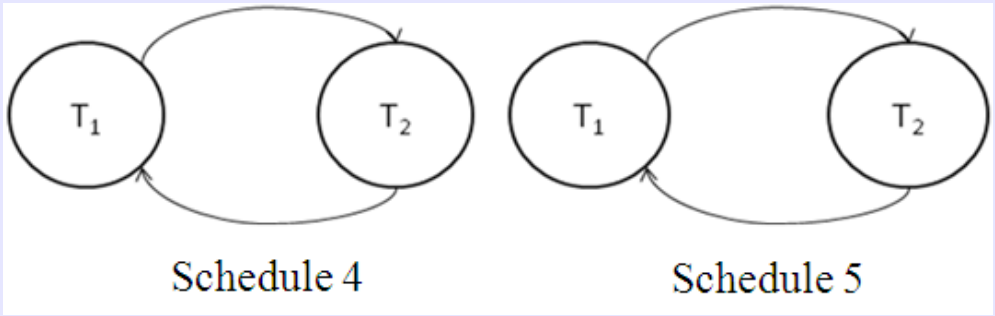
Testing for Conflict Serializability...

Testing for Conflict Serializability...

If the precedence graph for a concurrent schedule S has a cycle, then that schedule is not conflict serializable. If the graph contains no cycles, then the schedule S is conflict serializable







View Serializability

View Serializability

The schedules S and S_1 are said to be view equivalent if the following three conditions are met:

- For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then it must also read the initial value of Q in schedule S_1
- For each data item Q, the transaction that performs the final Write(Q) operation in schedule S must also perform the final Write(Q) operation in schedule S_1
- For each data item Q, if transaction T_i executes Read(Q) in schedule S, and if that value was produced by a Write(Q) operation executed by transaction T_j; then in schedule S_1, the Read(Q) operation of T_i must also read the value of Q that was produced by the same Write(Q) operation of transaction T_j

A schedule S is view serializable if it is view equivalent to a serial schedule

View Serializability...

View Serializability...

Schedule7

T_1	T_2	T_3
Read(Q);		
	Write(Q);	
Write(Q);		
		Write(Q);

- This schedule is view serializable
- This schedule is not conflict serializable
- *Every conflict serializable schedule is also view serializable, whereas all view serializable schedules are not conflict serializable*
- Every view serializable schedule, which is not conflict serializable, has blind writes

Testing for View Serializability

Testing for View Serializability

It can be tested using poly graph

- If the given schedule is conflict serializable, then it is also view serializable
- If the given schedule isn't conflict serializable and contains no blind writes, it is not view serializable
- If the given schedule isn't conflict serializable and contains blind writes, poly graph can be used to test

Database Management System 21

Concurrency Control

Chittaranjan Pradhan
School of Computer Engineering,
KIIT University

Need of Concurrency Control

Lost Update Problem

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect

T_1	T_2
Read(A);	
A:=A-100;	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
Write(A);	
	Write(A);
Read(B);	
B:=B+100;	
Write(B);	

Need of Concurrency Control...

Temporary Update(or Dirty Read) Problem

This problem occurs when one transaction updates a database item and then the transaction fails due to some reason. The updated item is accessed by another transaction before it is changed back to its original value

T_1	T_2
Read(A);	
A:=A-100;	
Write(A);	
	Read(A);
	Temp=0.2*A;
	A:=A-Temp;
	Write(A);
Read(B);	
B:=B+100;	

Need of Concurrency Control...

Incorrect Summary Problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated

T_1	T_2
	sum=0;
Read(A);	
A:=A-100;	
Write(A);	
	Read(A);
	sum:=sum+A;
	Read(B);
	sum:=sum+B;
Read(B);	
B:=B+100;	
Write(B);	

Lock-Based Protocols

Locking is a procedure used to control concurrent access to data. Locks enable a multi-user database system to maintain the integrity of transactions by isolating a transaction from others executing concurrently

Locking is one of the most widely used mechanisms to ensure serializability

Data items can be locked in two modes:

- **Shared lock or Read lock**: If a transaction T_i has obtained a shared mode lock(S) on data item Q, then T_i can only read the data item Q, but cannot write on Q
- **Exclusive lock or Write lock**: If a transaction T_i has obtained an exclusive mode lock(X) on data item Q, then T_i can both read and write Q

A transaction must obtain a lock on a data item before it can perform a read or write operation

Basic Rules for Locking

Basic Rules for Locking

- If a transaction has a read lock on a data item, it can only read the item; but cannot update its value
- If a transaction has a read lock on a data item, other transactions can obtain read locks on the same data item, but they cannot obtain any update lock on it
- If a transaction has a write lock on a data item, then it can both read and update the value of that data item
- If a transaction has a write lock on a data item, then other transactions cannot obtain either a read lock or a write lock on that data item

A transaction requests a shared lock on data item Q by executing the Lock-S(Q) instruction
Similarly, a transaction can request an exclusive lock through the Lock-X(Q) instruction
A transaction can unlock a data item Q by the Unlock(Q) instruction

Working of Locking

Working of Locking

- All transactions that need to access a data item must first acquire a read lock or write lock on the data item depending on whether it is a read only operation or not
- If the data item for which the lock is requested is not already locked, then the transaction is granted with the requested lock immediately
- If the item is currently locked, the database system determines what kind of lock is the current one. Also, it finds out which type of lock is requested:
  - If a read lock is requested on a data item that is already under a read lock, then the request will be granted
  - If a write lock is requested on a data item that is already under a read lock, then the request will be denied
  - Similarly; if a read lock or a write lock is requested on a data item that is already under a write lock, then the request is denied and the transaction must wait until the lock is released

Working of Locking...

Working of Locking...

- A transaction continues to hold the lock until it explicitly releases it either during the execution or when it terminates
- The effects of a write operation will be visible to other transactions only after the lock is released

A concurrent schedule, which is conflict serializable to a serial schedule, will always get the respective locks from the concurrency control manager

But, if the concurrent schedule is not conflict serializable, the requested locks will not be granted by the concurrency control manager

However, in case of Incorrect Summary Problem, all the requested locks will be granted resulting in incorrect values

Working of Locking...

Schedule3

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
	Write(A);
Read(B);	
Write(B);	
	Read(B);
	Write(B);

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
Write(A);		
Unlock(A)		
	Lock-X(A)	
		Grant-X(A, T_2)
	Read(A);	
	Write(A);	
	Unlock(A)	
Lock-X(B)		
		Grant-X(B, T_1)
Read(B);		
Write(B);		
Unlock(B)		
	Lock-X(B)	
		Grant-X(B, T_2)
	Read(B);	
	Write(B);	
	Unlock(B)	

Working of Locking...

Schedule4

T_1	T_2
Read(A);	
	Read(A);
	Write(A);
	Read(B);
Write(A);	
Read(B);	
Write(B);	
	Write(B);

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
	Lock-X(A)	

Working of Locking...

Schedule5

T_1	T_2
Read(A);	
Write(A);	
	Read(A);
Read(B);	
	Write(A);
	Read(B);
	Write(B);
Write(B);	

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
Write(A);		
Unlock(A)		
	Lock-X(A)	
		Grant-X(A, T_2)
	Read(A);	
Lock-X(B)		
		Grant-X(B, T_1)
Read(B);		
	Write(A);	
	Unlock(A)	
	Lock-X(B)	

Working of Locking...

T_1	T_2
Read(A);	Read(A);
A:=A-100;	Read(B);
Write(A);	Display(A+B);
Read(B);	
B:=B+100;	
Write(B);	

T_1	T_2
Lock-X(A);	Lock-S(A);
Read(A);	Read(A);
A:=A-100;	Unlock(A);
Write(A);	Lock-S(B);
Unlock(A);	Read(B);
Lock-X(B);	Unlock(B);
Read(B);	Display(A+B);
B:=B+100;	
Write(B);	
Unlock(B);	

Working of Locking...

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		
		Grant-X(A, T_1)
Read(A);		
A:=A-100;		
Write(A);		
Unlock(A)		
	Lock-S(A)	
		Grant-S(A, T_2)
	Read(A);	
	Unlock(A)	
	Lock-S(B)	
		Grant-S(B, T_2)
	Read(B);	
	Unlock(B)	
	Display(A+B);	
Lock-X(B)		
		Grant-X(B, T_1)
Read(B);		
B:=B+100;		
Write(B);		
Unlock(B)		

Though the concurrency control manager will not face any problem in granting the locks, the above schedule gives incorrect result for transaction T_2

Working of Locking...

To solve the previous discussed problem, different alternative solutions are possible. One solution can be by delaying the unlocking process. That means the unlocking is delayed to the end of the transaction

Unfortunately, this type of locking can lead to an undesirable situation

T_1	T_2
Lock-X(A);	Lock-S(A);
Read(A);	Read(A);
A:=A-100;	Lock-S(B);
Write(A);	Read(B);
Lock-X(B);	Display(A+B);
Read(B);	Unlock(A);
B:=B+100;	Unlock(B);
Write(B);	
Unlock(A);	
Unlock(B);	

Working of Locking...

T_1	T_2	Concurrency-Control Manager
Lock-X(A)		Grant-X(A, T_1)
Read(A);		
A:=A-100;		
Write(A);		
	Lock-S(A)	

Since T_1 is holding an exclusive-lock on A and T_2 is requesting a shared-lock on A, the concurrency control manager will not grant the lock permission to T_2. Thus, T_2 is waiting for T_1 to unlock A

T_3
Lock-X(B);
Read(B);
B:=B-100;
Write(B);
Lock-X(A);
Read(A);
A:=A+100;
Write(A);
Unlock(B);
Unlock(A);

Working of Locking...

T_3	T_2	Concurrency-Control Manager
Lock-X(B)		
		Grant-X(B, T_3)
Read(B);		
B:=B-100;		
Write(B);		
	Lock-S(A)	
		Grant-S(A, T_2)
	Read(A);	
	Lock-S(B);	
Lock-X(A)		

T_2 is waiting for T_3 to unlock B. Similarly, T_3 is waiting for T_2 to unlock A. Thus, this is a situation where neither of these transactions can ever proceed with its normal execution. This type of situation is called deadlock

If we do not use locking, or if we unlock data items as soon as possible after reading or writing them, we may get inconsistent states

On the other hand, if we do not unlock a data item before requesting a lock on another data item, deadlocks may occur

Locking Protocol

Locking Protocol

When a transaction requests a lock on a data item in a particular mode, and no other transaction has put a lock on the same data item in a conflicting mode, then the lock can be granted by the concurrency control manager

However, we must take some precautionary measures to avoid the following scenarios:

- Suppose a transaction T_1 has a shared-mode lock on a data item, and another transaction T_2 requests an exclusive-mode lock on that same data item. In this situation, T_2 has to wait for T_1 to release the shared-mode lock
- Suppose, another transaction T_3 requests a shared-mode lock on the same data item while T_1 is holding a shared lock on it. As the lock request is compatible with lock granted to T_1, so T_3 may be granted the shared-mode lock. But, T_2 has to wait for the release of the lock from that data item

Locking Protocol...

Locking Protocol

- At this point, T_1 may release the lock, but still T_2 has to wait for T_3 to finish. There may be a new transaction T_4 that requests a shared-mode lock on the same data item, and is granted the lock before T_3 releases it
- In such a situation, T_2 never gets the exclusive-mode lock on the data item. Thus, T_2 cannot progress at all and is said to be starved. This problem is called as the starvation problem

We can avoid starvation of transactions by granting locks in the following manner; when a transaction T_i requests a lock on a data item Q in a particular mode M, the concurrency-control manager grants the lock provided that:

- There is no other transaction holding a lock on Q in a mode that conflicts with M
- There is no other transaction that is waiting for a lock on Q and that made its lock request before T_i