PART-I
(Basics of DBMS)

Meaning of DBMS:

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know.

Basic meaning, operation applied on Database:

- Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS.
- Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database.
- Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes in the mini world, and generating reports from the data.
- Sharing a database allows multiple users and programs to access the database concurrently.

Other important functions provided by the DBMS:

It includes *protecting* the database and *maintaining* it over a long period of time. Protection includes both *system protection* against hardware or software malfunction (or crashes), and *security protection* against unauthorized or malicious access.

The DBMS is hence a *general-purpose software system* that facilitates the processes of *defining, constructing, manipulating,* and *sharing* databases among various users and applications.

To complete our initial definitions, we will call the database and DBMS software together a database system.

The primary goal of a DBMS:

DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information.

Additional Features of DBMS:

In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access. If data are to

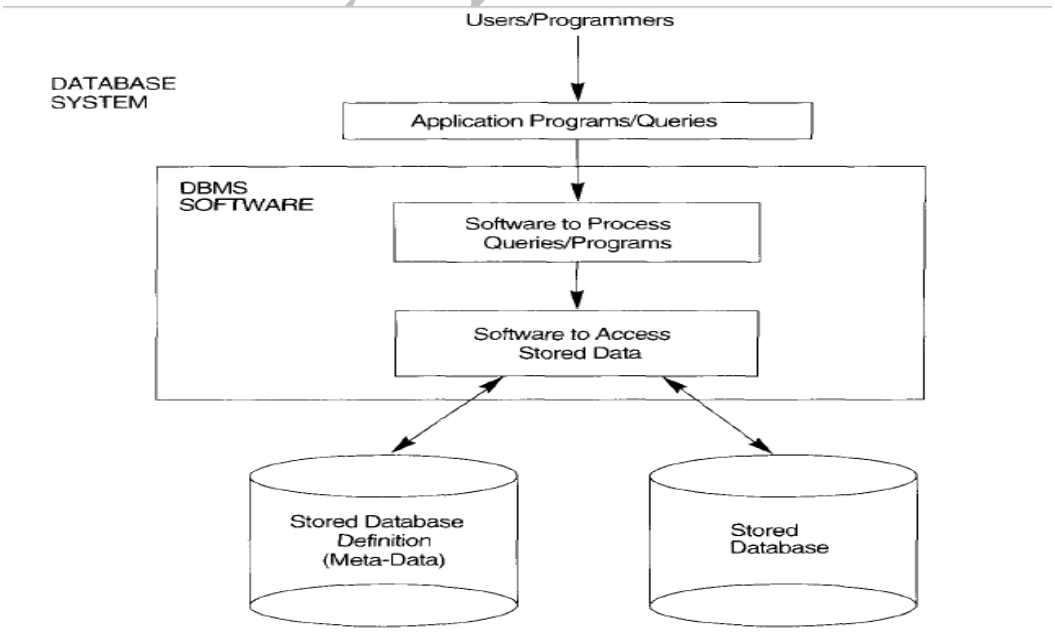
be shared among several users, the System must avoid possible anomalous results.

Database System Applications:

Databases are widely used. Here are some representative applications:

• *Banking*: For customer information, accounts, and loans, and banking transactions.
• *Airlines*: For reservations and schedule information.
• *Universities*: For student information, course registrations, and grades.
• *Credit card transactions*: For purchases on credit cards and generation of monthly statements.
• *Telecommunication*: For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
• *Finance*: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
• *Sales*: For customer, product, and purchase information.
• *Manufacturing*: For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
• *Human resources*: For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

A simplified database system environment:



A simplified database system environment.

Database Systems versus File Systems:

The **file-processing system** is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before database management systems (DBMSs) came along, organizations usually stored information in such systems.

Ex: Consider part of a savings-bank enterprise that keeps information about all customers and savings accounts. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including

- A program to debit or credit an account
- A program to add a new account
- A program to find the balance of an account
- A program to generate monthly statements

System programmers wrote these application programs to meet the needs of the bank. New application programs are added to the system as the need arises. Thus, as time goes by, the system acquires more files and more application programs.

Disadvantages of File System:

Keeping organizational information in a file-processing system has a number of major disadvantages:

• Data redundancy:

Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). This redundancy leads to higher storage and access cost.

For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records.

. Data inconsistency:

In addition, redundancy may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree for updating.

For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

• **Difficulty in accessing data**:

Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Suppose that one of the bank officers needs to find out the names of all customers who live within a particular postal-code area. But there is only an application program to generate the list of *all* customers. The bank officer has now two choices: either obtains the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory.

• **Data isolation**:

Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

• **Integrity problems:**

The data values stored in the database must satisfy certain types of **consistency constraints**. For example, the balance of a bank account may never fall below a prescribed amount (say, $25). Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them.

• **Atomicity problems**:

In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $50 from account *A* to account *B*. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account *A* but was not credited to account *B*, resulting in an inconsistent database state.
That is, the funds transfer must be *atomic.*

• **Concurrent-access anomalies**:

Many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates may result in inconsistent data. Consider bank account *A*, containing $500. If two customers withdraw funds (say $50 and $100 respectively) from
account *A* at about the same time, the result of the concurrent executions may leave the account in an incorrect (or inconsistent) state. Depending on which one writes the value last, the account may contain either $450 or $400, rather than the correct value of $350.

• **Security problems**:

Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only information about the various bank employees but they can also access to information about customer accounts.

These difficulties, among others, prompted the development of database systems.

Characteristics of the Database Approach:

The main characteristics of the database approach versus the file-processing approach are the following:

• **Self-describing nature of a database system-**

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the DBMS **catalog** called **meta-data.** In traditional file processing, data definition is typically part of the application programs themselves.

• **Insulation between programs and data, and data abstraction-**

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require *changing all programs* that access this file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this Property **program-data independence.**

A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented called data abstraction.

• **Support of multiple views of the data-**

A view may be a subset of the database or it may contain **virtual data** that is derived from the database files but is not explicitly stored.

• **Sharing of data and multiuser transaction processing-**

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct. The atomicity property ensures that either all the database operations in a transaction are executed or none are.

Actors on the Scene:

Many persons are involved in the design, use, and maintenance of a large database with hundreds of users. The people whose jobs involve the day-to-day use of a large database; we call them the "actors on the scene" and people who may be called "workers behind the scene"-those who work to maintain the database system environment but who are not actively interested in the database itself.

1. **Database Administrators**
2. **Database Designers**
3. **End Users**
4. **System Analysts and Application Programmers (Software Engineers)**

Database Administrators:

In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, for coordinating and monitoring its use, and for acquiring software and hardware resources as needed.

Database Designers:

Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data. These tasks are mostly undertaken before the database is actually implemented and populated with data.

End Users:

End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:

• Casual end users: Occasionally access the database, but they may need different information each time. They use a sophisticated database query language to specify their requests and are typically manager's rank.

• Naive or parametric end users: Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates. The tasks that such users perform are varied: Bank tellers check account balances and post withdrawals and deposits.

• Sophisticated end users: Include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS so as to implement their applications to meet their complex requirements.

• Stand-alone users: Maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces.

System Analysts and Application Programmers (Software Engineers):

System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for canned transactions that meet these requirements.

Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers- commonly referred to as software engineers.

WORKERS BEHIND THE SCENE:

Others are associated with the design, development, and operation of the DBMS *software and system environment but t*hese persons are typically not interested in the database itself. We call them the "workers behind the scene," and they include the following categories.

• DBMS system designers and implementers are persons who design and implement the DBMS modules and interfaces as a software package like - implementing the catalog, processing query language, processing the interface, accessing and buffering data, controlling concurrency, and handling data recovery and security.

• Tool developers include persons who design and implement tools-the software packages that facilitate database system design and use and that help improve performance.

• Operators and maintenance personnel who are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Although these categories of workers behind the scene are instrumental in making the database system available to end users, they typically do not use the database for their own purposes.

---X-----X---

PART-II
(Three Level Schema Architecture of Database)

Meaning of Schemas:

The description of a database is called the database schema, which is specified during database design.

Features:

- A schema diagram displays only *some aspects of a schema, such as the names of record,* types and data items, and some types of constraints.
- It is the logical structure of the database. Similar to types in programming languages.
- It is not expected to change frequently.
- The schema is sometimes called the intension.
- The DBMS stores the descriptions of the schema constructs and constraints-also called the meta-data.

Ex:

```
CREATE TABLE stdinfo
(  id NUMBER(3) PRIMARY KEY,
   name VARCHAR(15) NOT NULL
);
```

Here stdinfo is called Schema.

Meaning of Instances:

The actual content of the database at a particular point in a time. Similar to variables in programming languages.

Ex: INSERT INTO stdinfo VALUES(101,'xyz');

It creates one instance wrt stdinfo schema.

Meaning of Database State:

The data in the database at a particular moment in time is called a database state or snapshot. It is also called the *current set of occurrences or instances in the database.*

*Like SELECT * FROM stdinfo;*

It create one snapshot or database state.

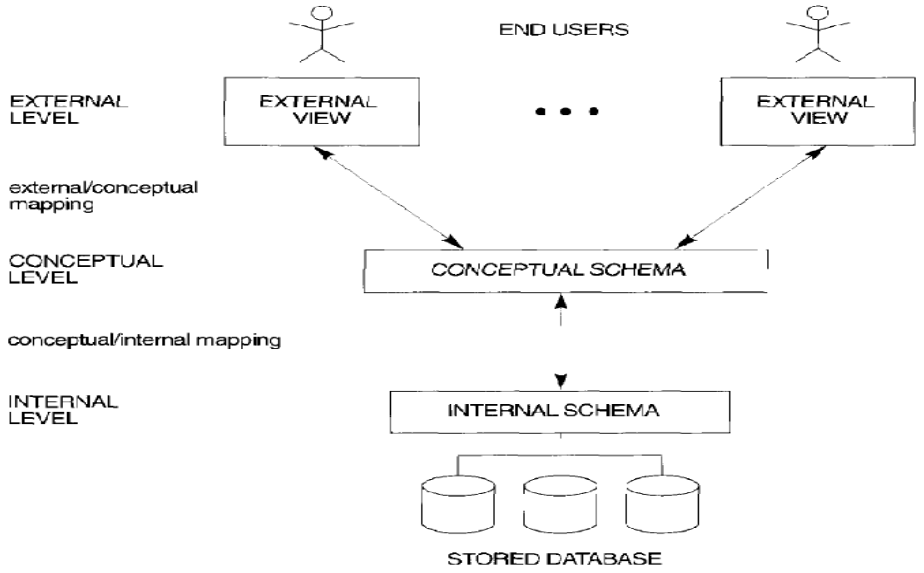
- Every time we insert or delete a record or change the value of a data item in a record, we change one state of the database into another state.
- Database state also called an extension of the schema.
- When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state with no data.*

Three Level Schema Architecture:

- The goal of the three-schema architecture, is to separate the user applications and the physical database.
- A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data is stored and maintained

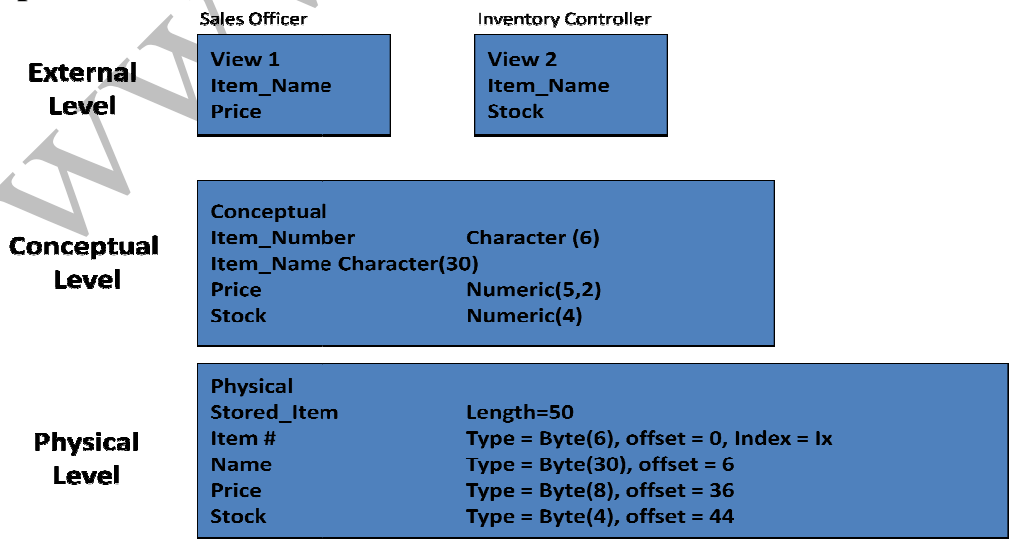
Three level of Schema are:

- *The internal/ Physical level*
- *Conceptual / Logical Level*
- *External / User View Level*



The three-schema architecture.

Example of three level architecture of DBMS :



The internal/ Physical level

- *The internal level is closest to physical storage.*
- It has an internal schema, which describes the physical storage structure of the database. *It means how data are actually stored on the storage medium.*
- The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database. Like various types of stored record, specifies what indexes exists, how files are represented, etc

Conceptual / Logical 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, user operations, and constraints.
- *The users of this level are not concerned with how these logical data structures will be implemented at the physical level, rather they just are concerned about what information is to be kept in the database.*

External / User View Level

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

Data independence:

- Data independence means when we bring out changes in lower levels then upper level does not feel any affect. i.e changing internal level does not effect conceptual level.
- The three-schema architecture can make it easier to achieve true data independence, both physical and logical.

It is of two types:-

1. Physical data independence
2. Logical data independence

Physical Data Independence:

- Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.
- Changes to the internal schema may be needed because some physical files had to be reorganized.

- for example: by creating additional access structures-to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

Logical data independence:

- Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs.
- We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
- In the last case, external schemas that refer only to the remaining data should not be affected.
- The logical data independence ensures that the application programs remain the same.

Physical & Logical Data Independence:

It is more difficult to achieve logical data independence than the physical data independence. The reason being that the application programs are heavily dependent on the logical structure of the database.

Mappings:

- The process to convert a request and the result between view levels is called mapping.
- The mapping defines the correspondence between three view levels.
- The mapping description is also stored in data dictionary.
- The DBMS is responsible for mapping between these three types of schemas.

There are two types of mapping.
(i) External-Conceptual mapping
(ii) Conceptual-Internal mapping

External-Conceptual mapping :

- An external-conceptual mapping defines the correspondence between a particular external view and the conceptual view.
- The external-conceptual mapping tells the DBMS which objects on the conceptual level correspond to the objects requested on a particular user's external view.
- If changes are made to either an external view or conceptual view, then mapping must be changed accordingly.

Conceptual-Internal mapping:

- The conceptual-internal mapping defines the correspondence between the conceptual view and the internal view, i.e. database stored on the physical storage device.
- It describes how conceptual records are stored and retrieved to and from the storage device.
- This means that conceptual-internal mapping tells the DBMS that how the conceptual! records are physically represented.
- If the structure of the stored database is changed, then the mapping must be changed accordingly. It is the responsibility of DBA to manage such changes.

Data Dictionary:

It contains information about the structures in the database. For Example, Each "object" in the database Tables, Views, Types, Procedures, Functions, Columns …, Who created it, Its Definition, What it uses etc.

- Data dictionary also called- System Catalog, Meta Data, Data Repository.
- DBMS Use a Data Dictionary for-Security, Integrity, View Definition, Parsing SQL, Optimizing SQL.
- Data Dictionary used by DBA, Programmer.

Data Models

A data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

Database models may be grouped into two categories:

1) Conceptual model *focuses on the logical nature of the data representation and is concerned* with *what is represented in the database.* Conceptual models include:

- Entity Relationship (E-R) model
- Object-Oriented model.

2) Implementation model *emphases on how information is represented in the database or on how* the data structures are implemented to represent what is modeled. Implementation models include:

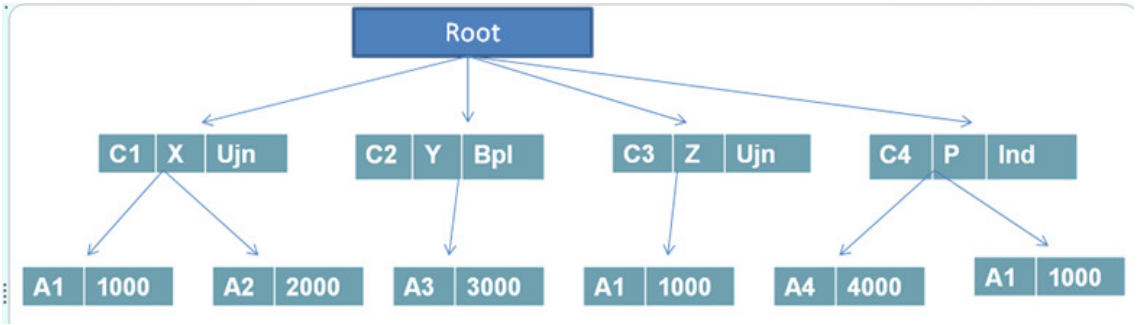
- Hierarchical database model
- Network database model
- Relational database model.

Within DBMS technology, as the hierarchical is the oldest DBMS data model, and the object-oriented being the newest DBMS data model.

Hierarchical database model:

In the hierarchical model, data is organized as an inverted tree. Each entity has only one parent but can have several children. At the top of the hierarchy, there is one entity, which is called the root.

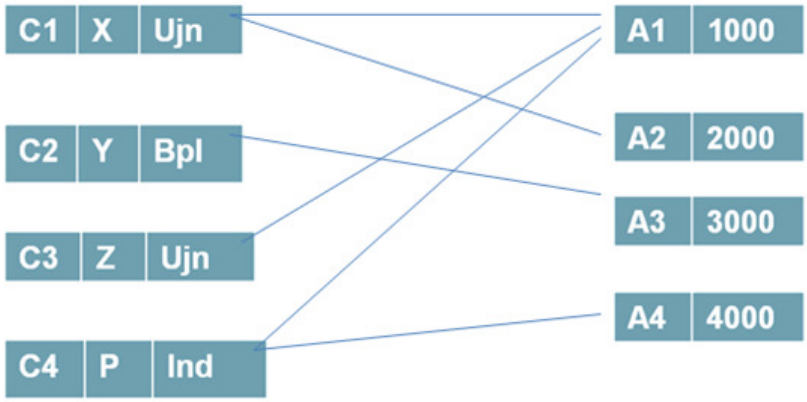
Ex: Hierarchical Model



Network database model:

In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths

Ex: Network Model



Relational database model:

In the relational model, data is organized in two-dimensional tables called relations. The tables or relations are, however, related to each other, as we will see shortly.

Ex: Relational Model

C-ID	Name	City
C1	X	Ujn
C2	Y	Bpl
C3	Z	Ujn
C4	P	Ind

Customer-Info

A-ID	Balance
A1	1000
A2	2000
A3	3000
A4	4000

Account-Info

C-ID	A-ID
C1	A1
C1	A2
C2	A3
C3	A1
C4	A1
C4	A4

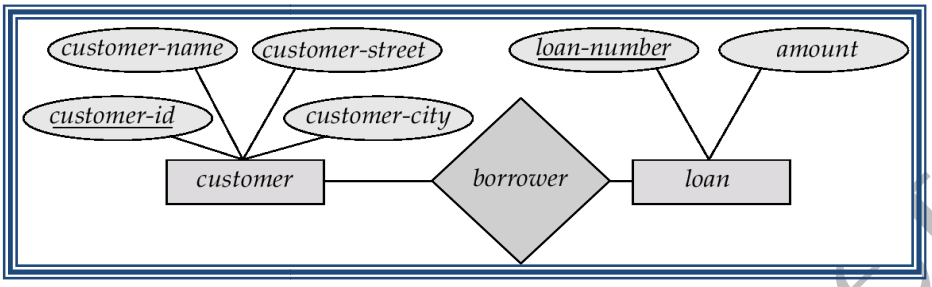
Cust-Account Info

Comparisons of these three model:

S.No	Hierarchical Data Model	Network Data Model	Relational Data Model
1.	Relationship between records is of the parent child type	Relationship between records is expressed in the form of pointers or links	Relationship between records is represented by a relation that contains a key for each record involved in the relationship
2.	Many to many relationship cannot be expressed in this model	Many to many relationship can also be implemented	Many to many relationship can be easily implemented
3.	It is a simple, straightforward and natural method of implementing record relationships	Record relationship implementation is very complex due to the use of pointers	Relationship implementation is very easy through the use of a key or composite key field(s)
4.	This type of model is useful only when there is some hierarchical character in the database.	Network model is useful for representing such records which have many to many relationships	Relational model is useful for representing most of the real world objects and relationships among them
5.	In order to represent links among records, pointers are used. Thus relations among records are physical.	In Network model also the record relations are physical.	Relational model does not maintain physical connection among records. Data is organized logically in the form of rows and columns and

Entity-Relationship Model:

It is a data model in which information are stored in the database is viewed as Entity Set and Relationship Set among entity.
Ex:



- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Lines link attributes to entity sets and entity sets to relationship sets.
- Ellipses represent attributes
- Underline indicates primary key attributes.

Entity Sets:

An *entity* is an object that exists and is distinguishable from other objects. Like- specific person, company, event, plant.

An *entity set* is a set of entities of the same type that share the same properties. Like- set of all persons, companies, trees, holidays.

Ex: Entity Sets for *customer* and *loan*

customer-id	customer-name	customer-street	customer-city
321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

customer

loan-number	amount
L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

loan

Attributes:

An entity is represented by a set of attributes that is *descriptive properties* possessed by all members of an entity set.

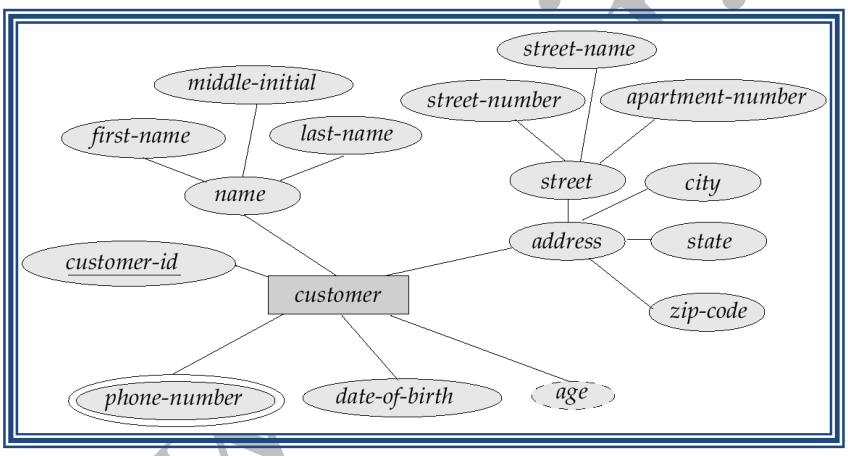
Ex: *customer = (customer-id, customer-name, customer-street, customer-city)*
loan = (loan-number, amount)

Domain – the set of permitted values for each attribute. For Example- List of customer-name only.

Types of Attribute:

- *Simple* and *composite* attributes. Ex: name, address are composite type.
- *Single-valued* and *multi-valued* attributes. Ex: Phone number is multi valued.
- *Derived* attributes: Can be computed from other attributes. Ex. *age*, given date of birth

E-R Diagram With Composite, Multivalued, and Derived Attributes

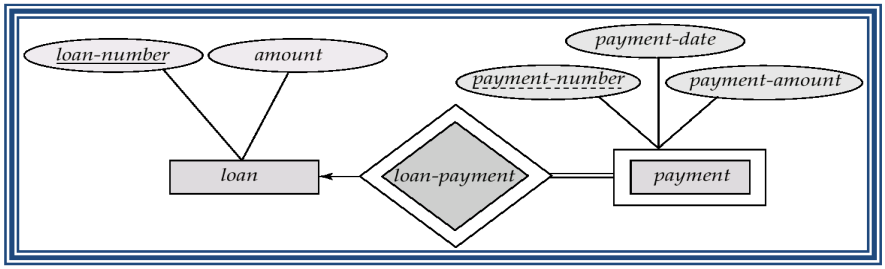


Ellipses represent attributes
- Double ellipses represent multi valued attributes.
- Dashed ellipses denote derived attributes.

Strong Entity v/s Weak Entity Set:

- An entity set which have a primary key is termed as a strong entity set; where as an entity set does not have sufficient attributes to form a primary key called weak entity.
- The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent.

Example:



Here-

- Identifying relationship depicted using a double diamond.
- We depict a weak entity set by double rectangles.
- Underline the discriminator of a weak entity set with a dashed line.
- *payment-number* – discriminator of the *payment* entity set.

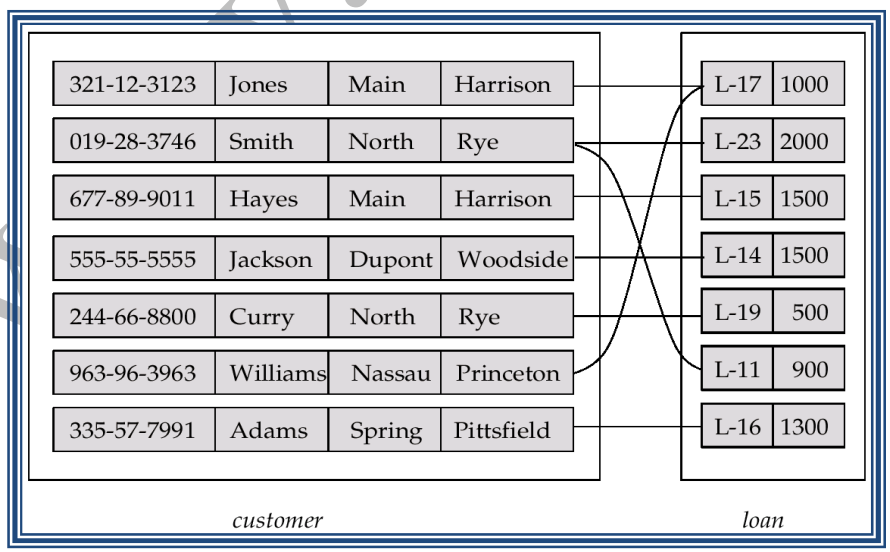
Primary key for *payment* – (*loan-number, payment-number*)

Relationship Sets:

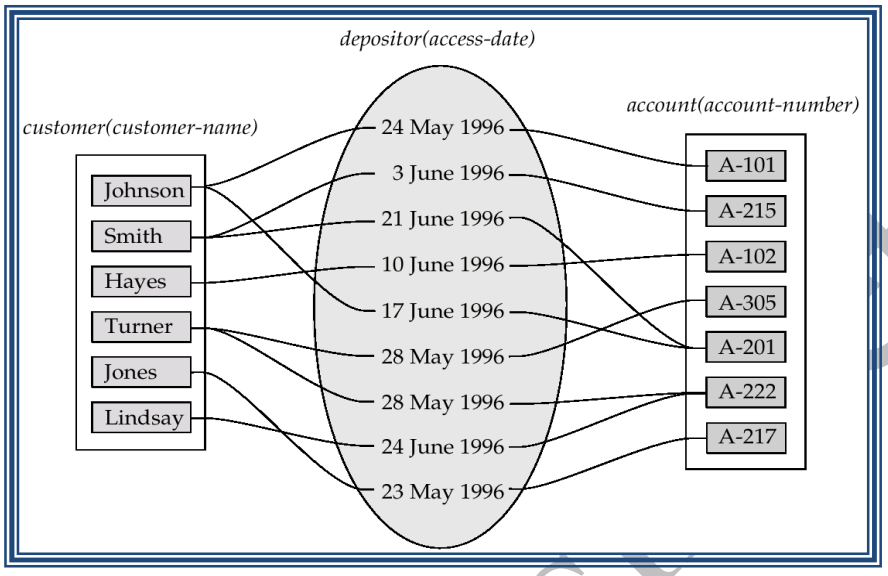
A relationship is an association among several entities. For example, we can define a relationship that associates customer Hayes with loan L-15. This relationship specifies that Hayes is a customer with loan number L-15.

A relationship set is a set of relationships of the same type. Consider the two entity sets *customer and loan . We define the relationship* set *borrower to denote the association between customers and the bank loans* that the customers have.

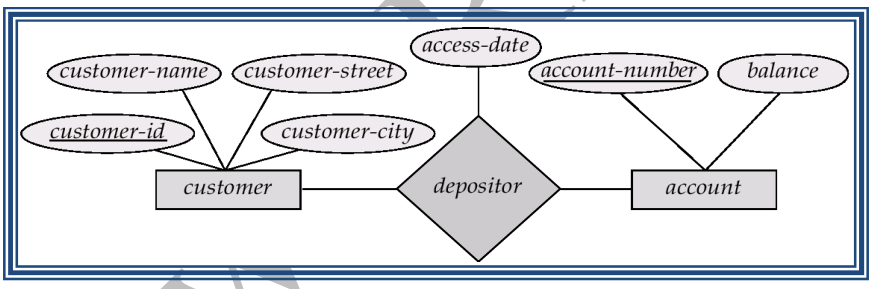
Relationship Set *borrower*



An *attribute* can also be property of a relationship set. For instance, the *depositor* relationship set between entity sets *customer* and *account* may have the attribute *access-date.*



Relationship Sets with Attributes



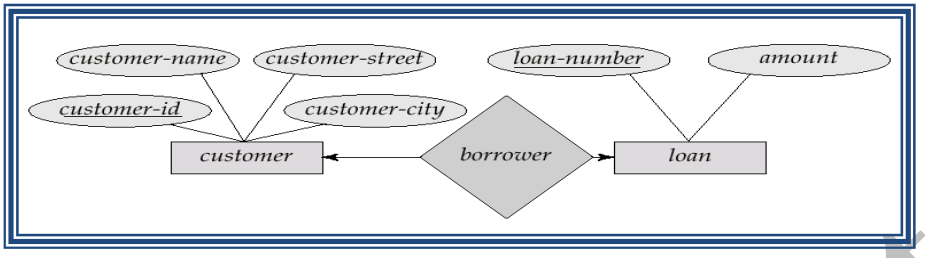
Cardinality Constraints

(Types of Relationship)

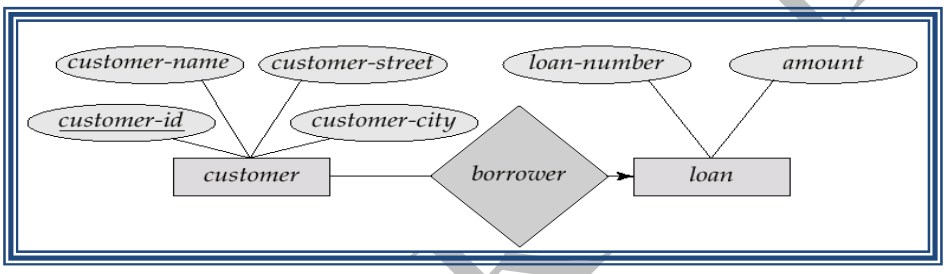
We express cardinality constraints by drawing either a directed line (→), signifying "one," or an undirected line (—), signifying "many," between the relationship set and the entity set.
It can be classified into 3 categories.

1. One to one
2. One to many
3. Many to many

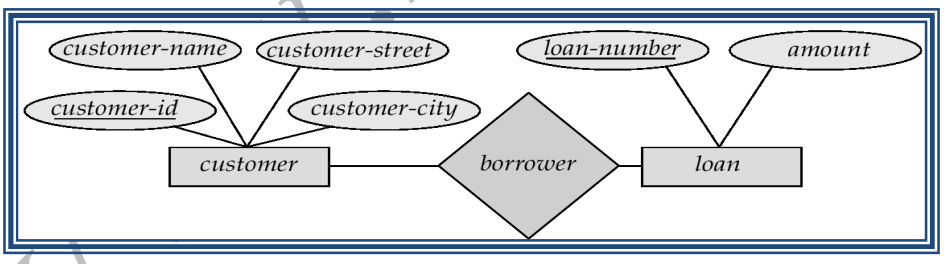
1. ***One-to-one relationship:*** A customer is associated with at most one loan via the relationship *borrower*. A loan is associated with at most one customer via *borrower*



2. ***Many-To-One Relationships:*** A loan is associated with several customers via *borrower*; a customer is associated with at most one loan via *borrower*.



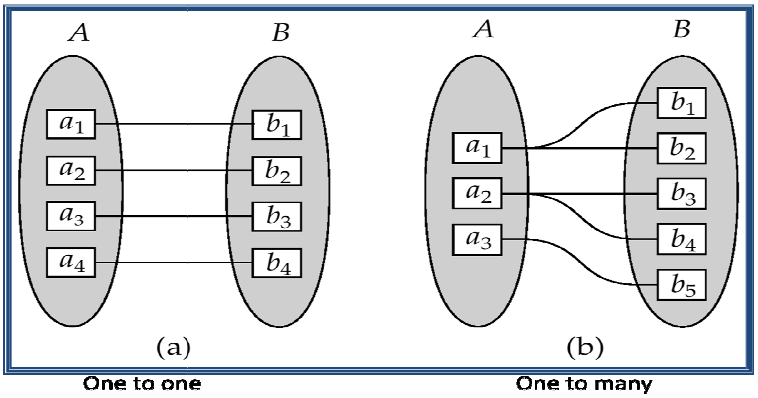
3. ***Many-To-Many Relationship:*** A customer is associated with several (possibly 0) loans via borrower. A loan is associated with several (possibly 0) customers via borrower.



Mapping Cardinalities:

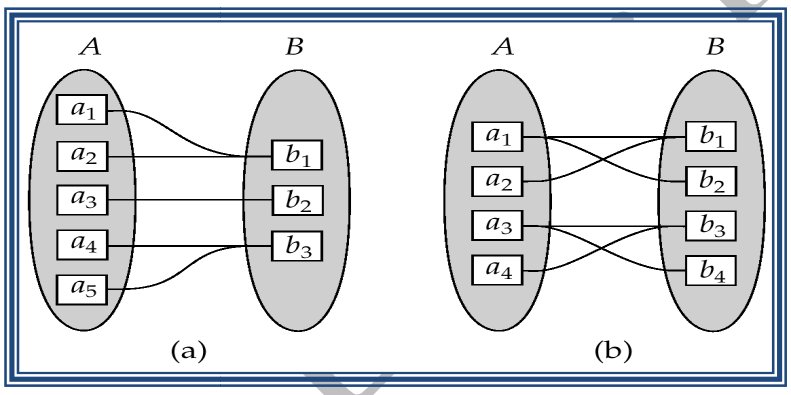
Express the number of entities to which another entity can be associated via a relationship set. Most useful in describing binary relationship sets.

Mapping cardinality between One to One and One to Many relationship set can be expressed as -



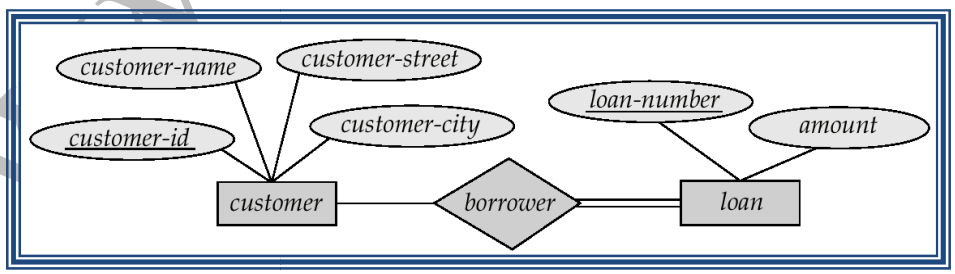
Note: Some elements in A and B may not be mapped to any elements in the other set

Mapping cardinality between Many to One and Many to Many relationship set can be expressed as-



Participation of an Entity Set in a Relationship Set:

1. Total participation *(indicated by double line)*
2. Partial participation

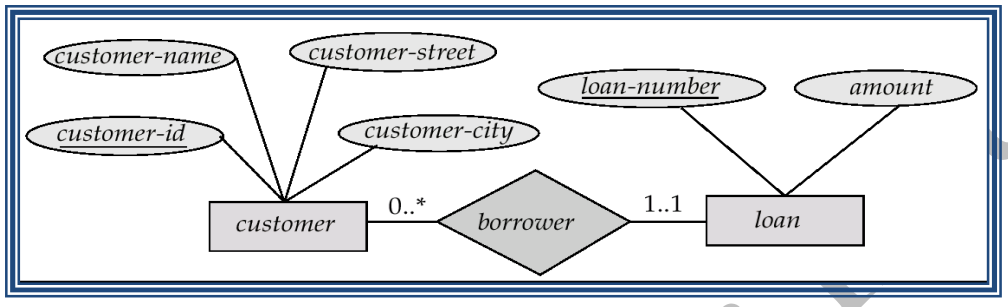


Total participation: When every entity in the entity set participates in at least one relationship in the relationship set. E.g. participation of *loan* in *borrower* is total. Here every loan must have a customer associated to it via borrower.

Partial participation: When some entities may not participate in any relationship in the relationship set. E.g. participation of *customer* in *borrower* is partial

Alternative Notation for Cardinality Limits

Cardinality limits can also express participation constraints

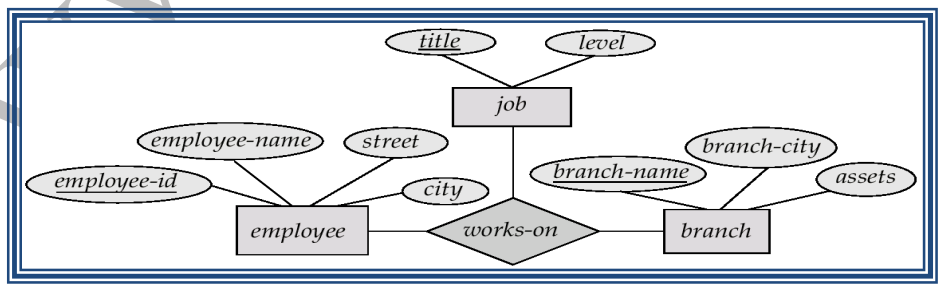


The edge between *loan and borrower has a cardinality* constraint of 1..1, *meaning the minimum and the maximum cardinality are* both 1. i.e. each loan must have exactly one associated customer.

The limit 0.. $*$ on the edge from *customer to borrower indicates that a customer can have zero or* more loans.

Degree of a Relationship Set: Refers to number of entity sets that participate in a relationship set.

- Relationship sets that involve two entity sets are *binary* (or degree two).
- Relationship sets may involve more than two entity sets like *ternary*.
- E.g.



Here, suppose employees of a bank may have jobs (responsibilities) at multiple branches, with different jobs at different branches. Then there is a ternary relationship set between entity sets *employee, job and branch.*

Relationships between more than two entity sets are rare. Most relationships are binary.

Meaning and Type of Key attributes

No two entities in an entity set are allowed to have exactly the same value for all attributes. A *key* allows us to identify a set of attributes that is enough to differentiate entities from each other.

super key: is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set.

For example, the *customer-id* attribute is sufficient to differ one *customer* entity from another. Thus, *customer-id* is a superkey. Similarly, the combination of *customer-name* and *customer-id* is a superkey for the entity set *customer*.

The *customer-name* attribute of *customer* is not a superkey, because several people might have the same name.

Candidate keys: Every attribute or combination of attributes that uniquely identifies an entity in entity set.

If *K* is a superkey, then superset of such *K contain one or more super keys*. Then we say each keys are candidate key. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called candidate keys.

Suppose that a combination of *customer-name* and *customer-street* is sufficient Then, both *{customer-id}* and *{customer-name*, *customer-street}* are candidate keys.

Although the attributes *customerid* and *customer-name* together does not form a candidate key, since the attribute *customer-id* alone is a candidate key.

primary key: a candidate key that is chosen for identifying entities within an entity set.

Composite key: A primary key that consist of more then one attributes.

Foreign key:An attributes in one entity set that serve as the primary key of another entity set. It is also called reference key.

DBMS Languages

Once the design of a database is completed and a DBMS is chosen to implement the database, then a set of instructions are required to specify various schemas for database, mappings between them, accessing and removing records etc. Thus format of such instructions called DBMS language.

DBMS language classified in two categories:

1) DDL – Data Definition Language

2) DML – Data Manipulation Language

DDL – (Data Definition Language): In DBMSs, a clear separation is maintained between the conceptual and internal levels, the DDL is used to specify the conceptual schema only where as Storage Definition Language (SDL), is used to specify the internal schema.

- The mappings between the two schemas may be specified in either one of these languages – DDL or SDL.
- We would need a third language, called View Definition Language (VDL), to specify user views and their mappings to the conceptual schema.
- In most DBMSs the DDL is used to define both conceptual and external schemas because there is no strict separation of levels are maintained.
- The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.
- DDL used by DBA and by database designers to define both schemas as conceptual and internal.

Example of DDL

How to create schema?

```
CREATE TABLE book_info
(
        bookid NUMBER(4),
        bookname VARCHAR2(15),
        author VARCHAR2(20),
        publisher VARCHAR2(15),
        price NUMBER(6,2),
);
```

DML – Data Manipulation Language

- Once the database schemas are compiled then operation on compiled database called manipulation.
- Manipulations include retrieval, insertion, deletion, and modification of the data.
- The DBMS provides a language to perform above operations called the Data Manipulation Language (DML) for these purposes.

Example of DML

- Insert new record-

INSERT INTO book_info VALUES(1001,'DBMS','KORTH','MC GRUE HILL',550.5);

It create only one instance of book_info schema.

Retrieval of data from database-

SELECT * FROM book_info.

Database Technology Trends

	1960s to Mid-1970s	1970s to Mid-1980s	Late 1980s	Future
Data Model	Network Hierarchical	Relational	Semantic Object-oriented Logic	Merging data models, knowledge representation, and programming languages
Database Hardware	Mainframes	Mainframes Minis PCs	Faster PCs Workstations Database machines	Parallel processing Optical memories
User Interface	None Forms	Query languages - SQL, QUEL	Graphics Menus Query-by-forms	Natural language Speech input
Program Interface	Procedural	Embedded query language	4GL Logic programming	Integrated database and programming language
Presentation and display processing	Reports Processing data	Report generators Information and transaction processing	Business graphics Image output Knowledge processing	Generalized display managers Distributed knowledge processing

Relational Algebra

The basic set of operations for the relational model is known as the relational algebra. These operations enable a user to specify basic retrieval requests.

The result of a retrieval is a new relation, which may have been formed from one or more relations. The **algebra operations** thus produce new relations, which can be further manipulated using operations of the same algebra.

A sequence of relational algebra operations forms a **relational algebra expression**, whose result will also be a relation that represents the result of a database query (or retrieval request)

1) **Unary Relational Operations:** Needs only one relation.
   Types: *SELECT, PROJECT, RENAME*
2) **Relational Algebra Operations From Set Theory:** Needs two relations.
   Types: *UNION, INTERSECTION, DIFFRENCE, CARTICIAN PRODUCT, JOIN*

1) SELECT Operation Properties (σ)

Selection(σ) in relational algebra returns those tuples in a relation(R) that fulfil a condition.

Syntax:

σ<selection condition>(R)

Condition can be created using column, values and operators.

In general, we allow comparisons using =, ≠, <, ≤, >, ≥ in the selection predicate.

Furthermore, we can combine several predicates into a larger predicate by using the connectives and (∧), or (∨), and not (¬).

Example: σ(rollno=101)(student_tbl)

We can give more then one condition

σ(rollno=101 V percent>80)(student_tbl)

SQL representation: > SELECT * FROM student_tbl WHERE rollno=101

Output:

Rollno	name	stream	branch	Percent
101	abc	MSc	CS	80

2) PROJECT Operation Properties (∏)

The PROJECT(∏) operation is used to select a subset of the attributes of a relation(R) by specifying the names of the required attributes.

Syntax:

$$\prod_{\text{<subset of attributes>}}(R)$$

Example:

$$\prod (rollno, name, percentage)(student\_tbl)$$

SQL representation:> *SELECT rollno, name,percentage FROM student_tbl*

Output:

Rollno	name	Percent
101	abc	80
102	xyz	70
103	pqw	90
104	rmn	60

We can also specified attributes with condition:

$$\prod (rollno, name, percentage)(\sigma(rollno=101 \text{ OR } percent>80)(student\_tbl))$$

Sql> SELECT rollno, name,percentage FROM student_tbl WHERE rollno=101 OR percent>80

Rename:(ρ)

The results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names;

Syntax1: $\rho_{S(B1, B2, \ldots, Bn)}(R)$

is a renamed relation S based on R with column names B1, B2, …..Bn.

Ex:ρ*student_contact(rollno,name,address,mobile)*
*(**Π**(rollno,name,address,mobile)(student_tbl))*

Sql> *CREATE TABLE student_contact(rollno,name,address,mobile)*

AS SELECT rollno,name,address,mobile FROM student_tbl;

Syntax 2: $\rho_S(R)$

is a renamed relation S based on R (which does not specify column names).

Ex: RENAME student_tbl TO student_master;

ρstudent_master(student_tbl)

UNION Operation(∪)

The result of this operation, denoted by R ∪ S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

Example: Πcustomer-name (borrower) ∪ Πcustomer-name (depositor)

find the names of all bank customers who have either an account or a loan or both

Type Compatibility

–The operand relations R1(A1, A2, ..., An) and R2(B1, B2, ...,Bn) must have the same number of attributes.

- the domains of corresponding attributes must be compatible; that is, dom(Ai)=dom(Bi) for i=1, 2, ..., n.

–The resulting relation for R1∪R2,R1∩R2, or R1-R2has the same attribute names as the *first* operand relation R1 (by convention).

R		S	
A	B	A	B
a1	b1	a1	b1
a2	b2	a2	b2
a3	b3	a4	b4

R U S

A	B
a1	b1
a2	b2
a3	b3
a4	b4

INTERSECTION OPERATION(∩)

The result of this operation, denoted by R ∩S, is a relation that includes all tuples that are in both R and S. The two operands must be "type compatible"

Example: Πcustomer-name (borrower) ∩ Πcustomer-name (depositor)

find all customers who have both a loan and an account

R		S	
A	B	A	B
a1	b1	a1	b1
a2	b2	a2	b2
a3	b3	a4	b4

R ∩ S

A	B
a1	b1
a2	b2

Set Difference (or MINUS) Operation(-)

The result of this operation, denoted by R - S, is a relation that includes all tuples that are in R but not in S. The two operands must be "type compatible".

Example: Πcustomer-name (borrower) - Πcustomer-name (depositor)

find all customers who have a loan but not an account

R

A	B
a1	b1
a2	b2
a3	b3

S

A	B
a1	b1
a2	b2
a4	b4

R - S

A	B
a3	b3

S - R

A	B
a4	b4

Notice that both union and intersection are *commutative operations;*that is

R ∪S = S ∪R, and R ∩S = S ∩R

Both union and intersection are *associative operations;* that is

R ∪(S ∪T) = (R ∪S) ∪T, and (R ∩S) ∩T = R ∩(S ∩T)

The minus operation is *not commutative;* that is, in general

R -S ≠S –R

The Cartesian-product operation(X)

This operation is used to combine tuples from two relations

In general, the result of R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm) is a relation Q with degree n + m attributes Q(A1, A2, . . ., An,B1, B2, . . ., Bm).

The resulting relation Q has one tuple for each combination of tuples—one from R and one from S.

The two operands do NOT have to be "type compatible"

R

A	B
a1	b1
a2	b2
a3	b3

S

A	B
a1	b1
a2	b2
a4	b4

R x S

A	B	A	B
a1	b1	a1	b1
a1	b1	a2	b2
a1	b1	a4	b4
a2	b2	a1	b1
a2	b2	a2	b2
a2	b2	a4	b4
a3	b3	a1	b1
a3	b3	a2	b2
a3	b3	a4	b4

JOIN Operation⋈

The sequence of Cartesian product followed by select called **JOIN**.

It is used to identify and select related tuples from two relations.

The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

R JOIN<join condition>S

where R and S can be any relations that result from general *relational algebra expressions.*

R S

A	B	B	C
a1	b1	b1	c1
a2	b2	b2	c2
a3	b3	b4	c4
a4	b4	b1	c2

R JOIN(R.B=S.B) S

A	R.B	S.B	C
a1	b1	b1	c1
a1	b1	b1	c2
a2	b2	b2	c2
a4	b4	b4	c4

Functional Dependency

Before defining functional dependency it is clear that- each tuple in a relation should represent one entity or relationship instance.

- Attributes of different entities should not be mixed in the same relation Only foreign keys should be used to refer to other entities

About FD

Functional dependencies (FDs) are used to specify *formal measures* of the "goodness" of relational designs. FDs and keys are used to define **normal forms** for relations. FDs are **constraints** that are derived from the *meaning* and *interrelationships* of the data attributes

A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y

X →Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y.

For any two tuples t1 and t2 in any relation instance r(R):

If t1[X]=t2[X], *then* t1[Y]=t2[Y]

X → Y in R specifies a *constraint* on all relation instances r(R)

Example: Suppose a company has a number of projects and they are handled by employees of different department and relations are define like this.

Employee→Ename, SSN(PK), Bdate, address, Dnumber(FK)
Department→Dname, Dnumber(PK), DmgrSSN(FK)
Dept_location→ Dnumber(FK), Dlocation(PK)
Project→ Pname, Pnumber(PK), Plocation, Dnumber(FK)
Works_On→Pnumber(FK), SSN(FK) (PK), hours.

In above relations-

social security number determines employee name

SSN -> ENAME

project number determines project name and location

PNUMBER -> {PNAME, PLOCATION}

employee ssn and project number determines the hours per week that the employee works on the project

{SSN, PNUMBER} -> HOURS

It is clear that If K is a key of R, then K functionally determines all attributes in R.

What is Normalization?

- Normalization is a process of converting a relation into a standard form.
- The possible approach is to design schemas that are in normal form.
- The normalization process is built around the concept of Normal form.
- It is the process of minimizing redundancy form the relation.

Why we need Normalization?

- The goal of RDBMS is to generate a set of relation schemas.
- To reduce unnecessary redundancy in stored information.
- The process of analyzing the given schemas based on their FDs and Primary keys to achieve desirable properties of
  a. Minimizing redundancy
  b. Minimizing Insertion, deletion and update anomalies

Normal Forms:

The 1st three NF were defined by Codd.

- 1NF
- 2NF
- 3NF

Later form proposed by Boyce and Codd

- BCNF
- 4NF
- 5NF

Unnormalized Relations:

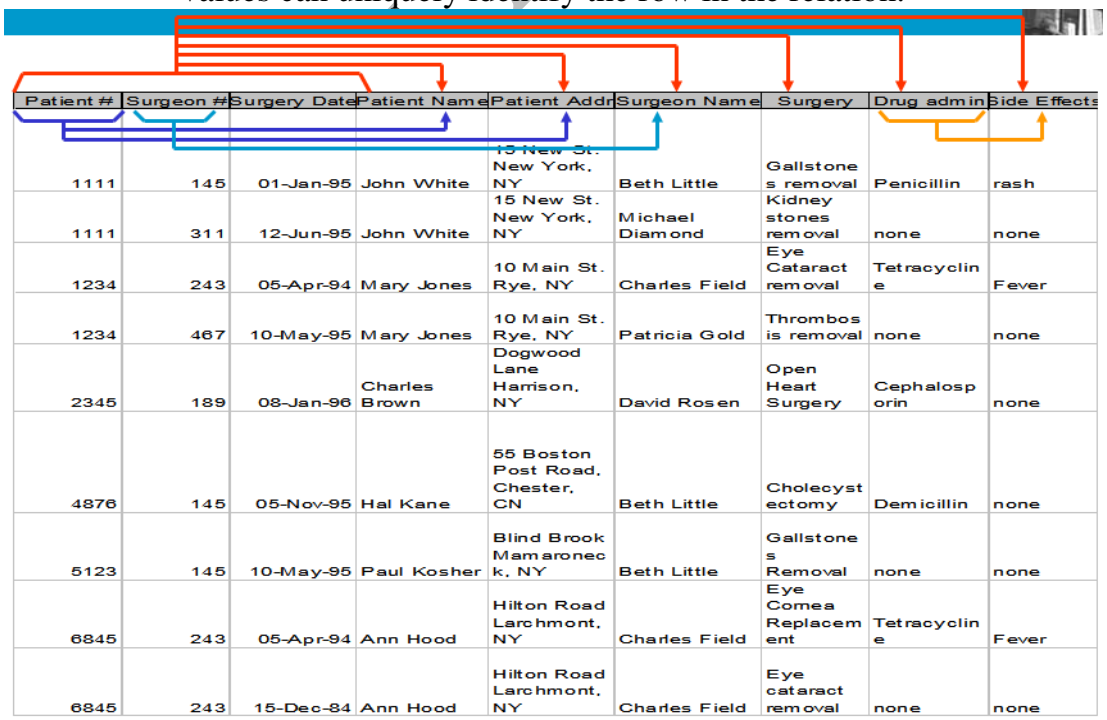
- First step in normalization is to convert the data into a two-dimensional table. In unnormalized relations data can repeat within a column.

Patient #	Surgeon #	Surg. date	Patient Name	Patient Addr	Surgeon	Surgery	Postop drug	ug side effec
1111	145 311	Jan 1, 1995; June 12, 1995	John White	15 New St. New York, NY	Beth Little Michael Diamond	Gallstones removal; Kidney stones removal	Penicillin, none-	rash none
1234	243 467	Apr 5, 1994 May 10, 1995	Mary Jones	10 Main St. Rye, NY	Charles Field Patricia Gold	Eye Cataract removal Thrombosis removal	Tetracycline none	Fever none
2345	189	Jan 8, 1996	Charles Brown	Dogwood Lane Harrison, NY	David Rosen	Open Heart Surgery	Cephalosporin	none
4876	145	Nov 5, 1995	Hal Kane	55 Boston Post Road, Chester, CN	Beth Little	Cholecystectomy	Demicillin	none
5123	145	May 10, 1995	Paul Kosher	Blind Brook Mamaroneck, NY	Beth Little	Gallstones Removal	none	none
6845	243	Apr 5, 1994 Dec 15, 1984	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye Cornea Replacement Eye cataract removal	Tetracycline	Fever

First Normal Form:

To move to First Normal Form a relation must contain only atomic values at each row and column.

- No repeating groups
- A column or set of columns is called a Candidate Key when its values can uniquely identify the row in the relation.



Patient #	Surgeon #	Surgery Date	Patient Name	Patient Addr	Surgeon Name	Surgery	Drug admin	Side Effects
1111	145	01-Jan-95	John White	15 New St. New York, NY	Beth Little	Gallstones removal	Penicillin	rash
1111	311	12-Jun-95	John White	15 New St. New York, NY	Michael Diamond	Kidney stones removal	none	none
1234	243	05-Apr-94	Mary Jones	10 Main St. Rye, NY	Charles Field	Eye Cataract removal	Tetracycline	Fever
1234	467	10-May-95	Mary Jones	10 Main St. Rye, NY	Patricia Gold	Thrombosis removal	none	none
2345	189	08-Jan-96	Charles Brown	Dogwood Lane Harrison, NY	David Rosen	Open Heart Surgery	Cephalosporin	none
4876	145	05-Nov-95	Hal Kane	55 Boston Post Road, Chester, CN	Beth Little	Cholecystectomy	Demicillin	none
5123	145	10-May-95	Paul Kosher	Blind Brook Mamaroneck, NY	Beth Little	Gallstones Removal	none	none
6845	243	05-Apr-94	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye Cornea Replacement	Tetracycline	Fever
6845	243	15-Dec-84	Ann Hood	Hilton Road Larchmont, NY	Charles Field	Eye cataract removal	none	none

1NF Storage Anomalies:

- Insertion: A new patient has not yet undergone surgery -- hence no surgeon # -- Since surgeon # is part of the key we can't insert.
- Insertion: If a surgeon is newly hired and hasn't operated yet -- there will be no way to include that person in the database.
- Update: If a patient comes in for a new procedure, and has moved, we need to change multiple address entries.
- Deletion (type 1): Deleting a patient record may also delete all info about a surgeon.
- Deletion (type 2): When there are functional dependencies (like side effects and drug) changing one item eliminates other information.

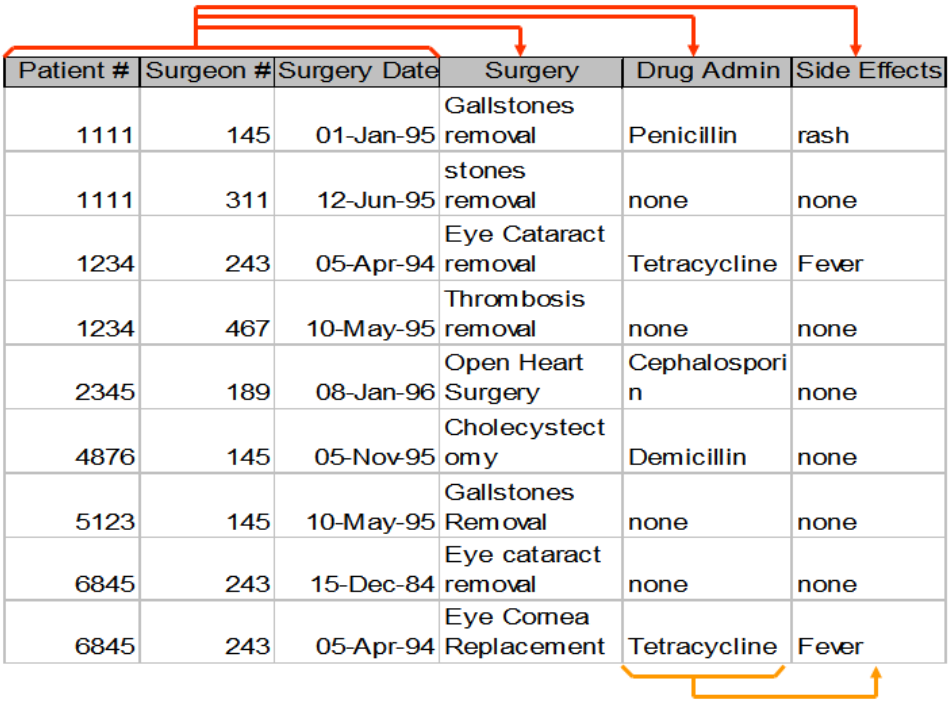
Second Normal Form:

A relation is said to be in Second Normal Form when every nonkey attribute is **fully functionally dependent** on the primary key.

– That is, every non key attribute needs the full primary key for unique identification.

Surgeon #	Surgeon Name
145	Beth Little
189	David Rosen
243	Charles Field
311	Michael Diamond
467	Patricia Gold

Patient #	Patient Name	Patient Address
1111	John White	15 New St. New York, NY
1234	Mary Jones	10 Main St. Rye, NY
2345	Charles Brown	Dogwood Lane Harrison, NY
4876	Hal Kane	55 Boston Post Road, Chester,
5123	Paul Kosher	Blind Brook Mamaroneck, NY
6845	Ann Hood	Hilton Road Larchmont, NY



Patient #	Surgeon #	Surgery Date	Surgery	Drug Admin	Side Effects
1111	145	01-Jan-95	Gallstones removal	Penicillin	rash
1111	311	12-Jun-95	stones removal	none	none
1234	243	05-Apr-94	Eye Cataract removal	Tetracycline	Fever
1234	467	10-May-95	Thrombosis removal	none	none
2345	189	08-Jan-96	Open Heart Surgery	Cephalosporin	none
4876	145	05-Nov-95	Cholecystectomy	Demicillin	none
5123	145	10-May-95	Gallstones Removal	none	none
6845	243	15-Dec-84	Eye cataract removal	none	none
6845	243	05-Apr-94	Eye Cornea Replacement	Tetracycline	Fever

1NF Storage Anomalies Removed in 2NF:

- Insertion: Can now enter new patients without surgery.
- Insertion: Can now enter Surgeons who haven't operated.
- Deletion (type 1): If Charles Brown dies the corresponding tuples from Patient and Surgery tables can be deleted without losing information on David Rosen.
- Update: If John White comes in for third time, and has moved, we only need to change the Patient table

2NF Storage Anomalies:

- Insertion: Cannot enter the fact that a particular drug has a particular side effect unless it is given to a patient.
- Deletion: If John White receives some other drug because of the penicillin rash, and a new drug and side effect are entered, we lose the information that penicillin can cause a rash

Update: If drug side effects change (a new formula) we have to update multiple occurrences of side effects.

Third Normal Form:

- A relation is said to be in Third Normal Form if there is no transitive functional dependency between nonkey attributes
  - When one nonkey attribute can be determined with one or more nonkey attributes there is said to be a transitive functional dependency.
- The side effect column in the Surgery table is determined by the drug administered
  - Side effect is transitively functionally dependent on drug so Surgery is not 3NF.

Patient #	Surgeon #	Surgery Date	Surgery	Drug Admin
1111	145	01-Jan-95	Gallstones removal	Penicillin
1111	311	12-Jun-95	Kidney stones removal	none
1234	243	05-Apr-94	Eye Cataract removal	Tetracycline
1234	467	10-May-95	Thrombosis removal	none
2345	189	08-Jan-96	Open Heart Surgery	Cephalosporin
4876	145	05-Nov-95	Cholecystectomy	Demicillin
5123	145	10-May-95	Gallstones Removal	none
6845	243	15-Dec-84	Eye cataract removal	none
6845	243	05-Apr-94	Eye Cornea Replacement	Tetracycline

Drug Admin	Side Effects
Cephalosporin	none
Demicillin	none
none	none
Penicillin	rash
Tetracycline	Fever

2NF Storage Anomalies Removed in 3NF:

- Insertion: We can now enter the fact that a particular drug has a particular side effect in the Drug relation.
- Deletion: If John White receives some other drug as a result of the rash from penicillin, but the information on penicillin and rash is maintained.
- Update: The side effects for each drug appear only once.

Boyce-Codd Normal Form:

- Most 3NF relations are also BCNF relations.
- A 3NF relation is NOT in BCNF if:
  - Candidate keys in the relation are composite keys (they are not single attributes)
  - There is more than one candidate key in the relation, and
  - The keys are not disjoint, that is, some attributes in the keys are common

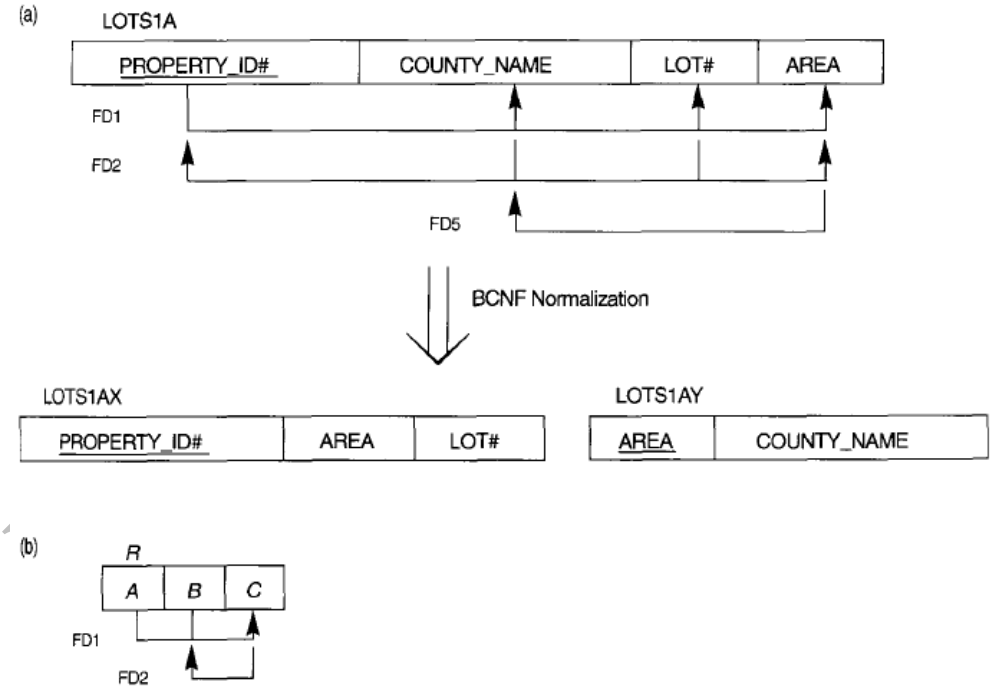


FIGURE 10.12 Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF.

Fourth Normal Form:

- A relation is in 4 NF iff, it does not exists an multi-valued dependency (MVD) in the relation.
- MVD exists only if the relation R has at least 3 attributes.
- R(A,B,C) then MVD between (R.A→→R.B) holds iff the MVD between (R.A→→R.C) holds and (R.B, R.C) are independent to each other.
- Common notation (R.A→→R.B|R.C)

CTX

Course	Teacher	Text
Pyhsics	Prof. Green Prof. Brown Prof. Black	Basic Mechanics Principle of Optics
Maths	Prof. White	Modern Algebra Projective Geometry

Table 4.14 Sample Tabulation of CTX (unnormalised)

It is unnormalized relation indicate course can be taught by any of the indicated teachers and uses all the indicated texts.

Convert equivalent normalized form

CTX

Course	Teacher	Text
Physics	Prof. Green	Basic Mechanics
Physics	Prof. Green	Principle of Optics
Physics	Prof. Brown	Basic Mechanics
Physics	Prof. Brown	Principle of Optics
Physics	Prof. Black	Basic Mechanics
Physics	Prof. Black	Principle of Optics
Maths	Prof. White	Modern Algebra
Maths	Prof. White	Projective Geometry

Table 4.15 Sample Tabulation of CTX (Normalised)

Here teacher and text are independent of each other. Decompose it.

CT

Course	Teacher
Physics	Prof. Green
Physics	Prof. Brown
Physics	Prof. Black
Maths	Prof. White

CX

Course	Text
Physics	Basic Mechanics
Physics	Principles of Optics
Maths	Modern Algebra
Maths	Projective Geometry

Table 4.16 Sample Tabulation of CT and CX

Fifth Normal Form:

- A relation R is in 5 NF also called Projection-Join (PJNF), iff every join dependency in relation R is implied by candidate keys of R.
- Ex: following relation SPJ is in upto 4NF.
- It has three projection SP, PJ and JS.
- Effect of joining SP and PJ over P#.
- joining result and JS over (J#, S#) must produce original copy of SPJ.

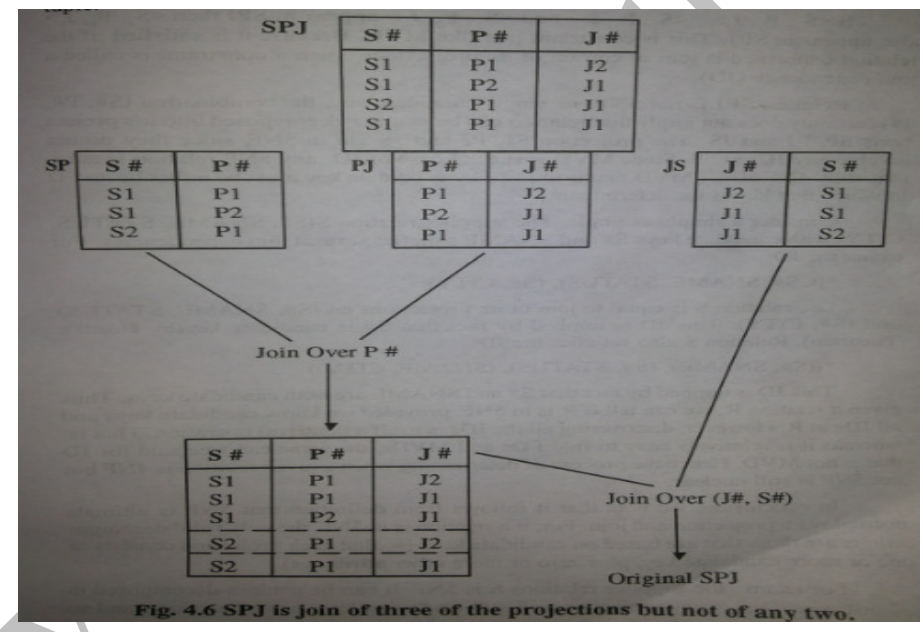


Fig. 4.6 SPJ is join of three of the projections but not of any two.

- Relation SPJ is not 5NF; Its single candidate key(combination of S#,P#,J#) does not imply that relation can be nonloss-decomposed into it's projections SP,PJ and JS.
- The projection SP, PJ and JS are in 5 NF because they do not involve in any join dependency.

TRANSACTION, CONCURRENCY & RECOVERY

The concept of *transaction*:

Collections of operations that form a single logical unit of work are called transactions.

Transaction processing systems: Systems with large databases and hundreds of concurrent users that are executing database transactions**.**

What is a Transaction ?

- A logical unit of work on a database
  - An entire program
  - A single command
- The entire series of steps necessary to accomplish a logical unit of work
- Successful transactions change the database from one CONSISTENT STATE to another

  (One where all data integrity constraints are satisfied)

Example of a Transaction

For example, transfer of funds from a checking account to a saving account is a single operation from the customer's standpoint; within the database system, however, it consists of several operations. Like when we want to update a record-

- Locate the Record on Disk
- Bring record into Buffer
- Update Data in the Buffer
- Writing Data Back to Disk

"Single user Versus Multiuser Systems"

- **Single-User** : at most one user at a time can use the system
- **Multiuser** : many users can use the system concurrently.

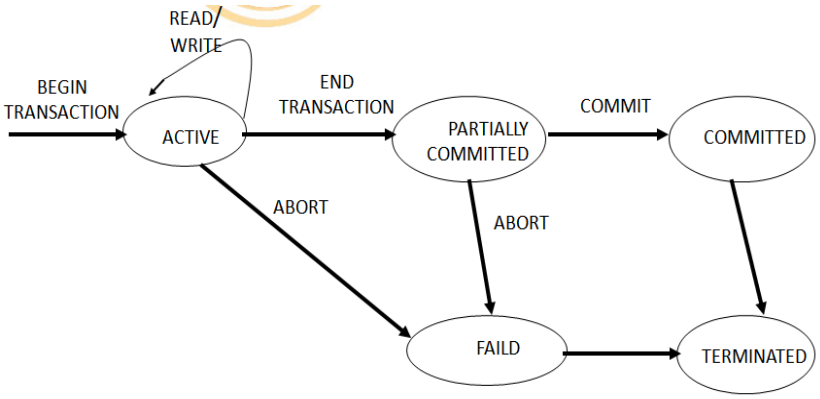
- **Multiprogramming** : allows the computer to execute multiple programs at the same time.
- **Interleaving** : keeps the CPU busy when a process requires an input or output operation, the CPU switched to execute another process rather than remaining idle during I/O time .
- Most of the theory concerning concurrency control in databases is developed in terms of interleaved concurrency.

"Transactions, Read and Write Operations and DBMS Buffers "

- **A Transaction** : is a logical unit of database processing that includes one or more database access operation.
- All database access operations between **Begin Transaction** and **End Transaction** statements are considered one logical transaction.
- If the database operations in a transaction do not update the database but only retrieve data , the transaction is called a **read-only transaction.**
- **Basic database access operations** :
  - **read_item(X)** : reads a database item **X** into program variable.
  - **Write_item(X)** : Writes the value of program variable **X** into the database item **X**.

"Transaction states and additional operations

- A transaction is an atomic unit of work that is either completed in its entirety or not done at all.
- For recovery purposes the system needs to keep track of when the transaction starts, terminates, and commits or aborts.
- The recovery manager keeps track of the following operations :
  - BEGIN_TRANSACTION
  - READ OR WRITE
  - END_TRANSACTION
  - COMMIT_TRANSACTION
  - ROLLBACK



TRANSACTION STATES

- **Active - T**he initial state; the transaction stays in this state until while it is still executing.
- **Partially committed -** After the final statement has been executed. At this point failure is still possible since changes may have been only done in main memory, a hardware failure could still occur.
- **Committed**- After successful completion. Once committed, the transaction can no longer be undone by aborting it.
- **Failed - A**fter the discovery that normal execution can no longer proceed.
- **Aborted - A**fter the transaction has been rolled back, the database has been restored to its state prior to the start of the transaction.

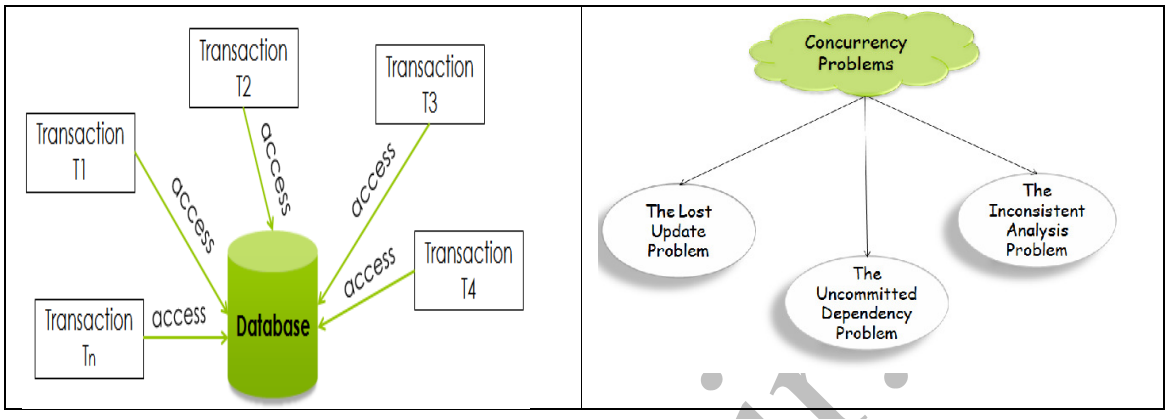
The ACID Properties

- Atomicity : All actions in the exact happen, or none happen.
- Consistency : If each exact is consistent, and the DB starts consistent, it ends up consistent.
- **Isolation** : Execution of one exact is isolated from that of other exacts.
- **Durability** : After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Concurrency Control Mechanisms

What is Concurrency?

The term concurrency refers to the fact that DBMSs typically allow many transactions to access the same database at the same time.



- Coordination of simultaneous transaction execution in a multiprocessing database system.
- Ensure transaction serializability in a multi-user database.
- Lack of Concurrency Control can create data integrity and consistency problems:
  - Lost Updates
  - Temporary Update (Dirty Read)
  - Incorrect Summary
  - Unrepeatable Read

Why Concurrency Control is needed ?

Several problems can occur when concurrent transactions execute in an uncontrolled manner.

1. **The Lost Update Problem** : A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running

concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.

2. **The temporary Update (or Dirty read) problem:** Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have been read by any transaction ("dirty read"). The reading transactions end with incorrect results.

3. **The Incorrect Summary Problem : ** While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

4. **Unrepeatable Read problem** : A transaction reads items twice with two different values because it was changed by another transaction between the two reads.

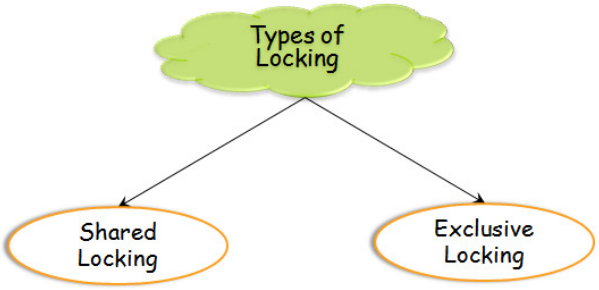
Transaction Recovery

- Restore a database from a given state to a previous consistent state.
- Atomic Transaction Property (All or None)
- Backup Levels:
  - Full Backup
  - Differential Backup
  - Transaction Log Backup
- Database / System Failures:
  - Software (O.S., DBMS, Application Programs, Viruses)
  - Hardware (Memory Chips, Disk Crashes, Bad Sectors)
  - Programming Exemption (Application Program rollbacks)
  - Transaction (Aborting transactions due to deadlock detection)
  - External (Fire, Flood, etc)
- Recover Database by using data in the Transaction Log

- Write-Ahead-Log – Transaction logs need to be written before any database data is updated
- Redundant Transaction Logs – Several copies of log on different devices
- Database Buffers – Buffers are used to increase processing time on updates instead of accessing data on disk
- Database Checkpoints – Process of writing all updated buffers to disk → While this is taking place, all other requests are not executes
  - Scheduled several times per hour
  - Checkpoints are registered in the transaction log

Locking: (technique for Concurrency Control)

- Some of the main techniques used to control concurrent execution of transactions are based on the concept of locking data items.
- "The concept of locking data items is one of the main techniques used for controlling the concurrent execution of transactions."
- Locking is a mechanism used to ensure data integrity while providing concurrent access to data.



- Shared/Exclusive or Read/Write or S/X locking is a multiple-mode lock.
- The lock has three values:
  - read_locked (shared lock): The item is locked for read purpose and can be shared for reading by another transaction.
  - write_locked (exclusive lock): The item is locked for write purpose and cannot be accessed by another transaction.
  - unlocked: The item is unlocked and can be accessed by any transaction.

Locking Protocols

Consider some tuple t; suppose transaction A holds lock on t and some distinct transaction B issues a request for a lock on t, then-

1. A transaction that wishes to retrieve a tuple must first acquire an S lock on that tuple.
2. A transaction that wishes to update a tuple must first acquire an X lock on that **tuple.**
3. If transaction A holds an exclusive (X) lock on tuple t, then a request from some distinct transaction B for a lock of either type on t cannot be immediately granted because it conflict with a lock already held by transaction A, B goes into a **wait state.** B will wait until the A's lock is released.
4. **X** locks are released at the end of transaction (COMMIT or ROLLBACK). **S** locks are normally released at that time also.
5. If transaction A holds a Shared (S) lock on tuple t, then:
   - A request from some distinct transaction B for an X lock on t cannot be immediately granted.
   - A request from some distinct transaction B for an S lock on t can and will be immediately granted (i.e., B will now also hold an S lock t).

Above rules can be summarized by means of a **lock type compatibility matrix** as follows:

	X	S
X	N	N
S	N	Y

Granting a Lock

- The **Concurrency Control Manager** is responsible for granting a lock.
- When a transaction Ti request a lock on a data item Q in a particular mode M, the concurrency control manager grants the lock provided that:

1. There is no other transaction holding a lock on Q in a mode that conflicts M.

2. There is no other transaction that is waiting for a lock on Q and that made a lock request before Ti.

Implementation of Locking

- The **Lock Manager** implements a process that receives message from transactions and send message in reply.
- The Lock Manager uses a linked list data structure of records, one for each request, in the order in which the requests arrived. It uses Lock table for this purpose.
- **The Lock Manager processes requests in following way**:
  - When a **lock request message** arrives, it adds a record to the end of linked list for the data item. It always grants a lock request on a data item that is not currently locked.
  - When the lock manager receives an **unlock message** from a transaction, it deletes the records for that data item in the linked list corresponding to that transaction.
  - If a transaction **aborts**, the lock manager deletes any waiting request made by the transaction.

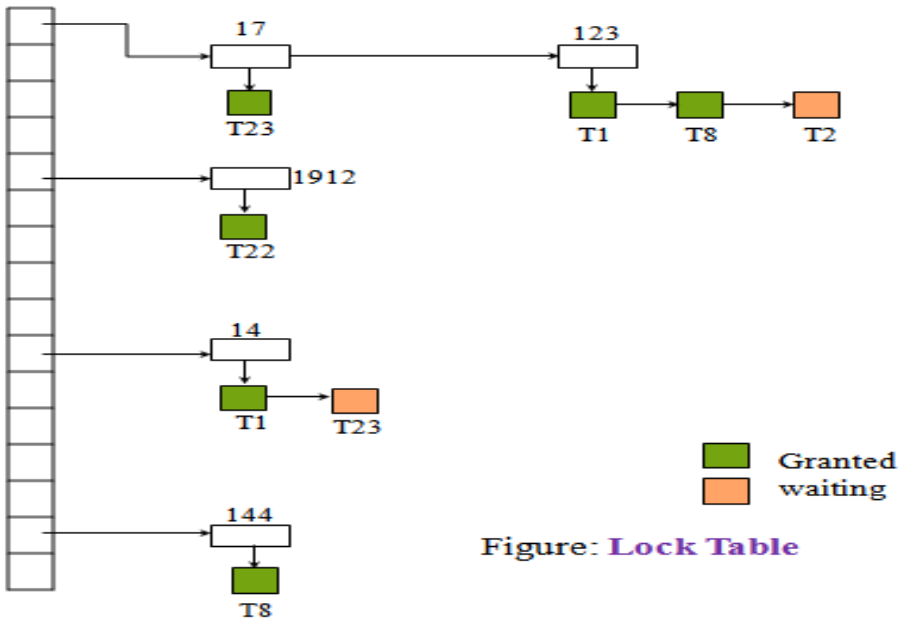


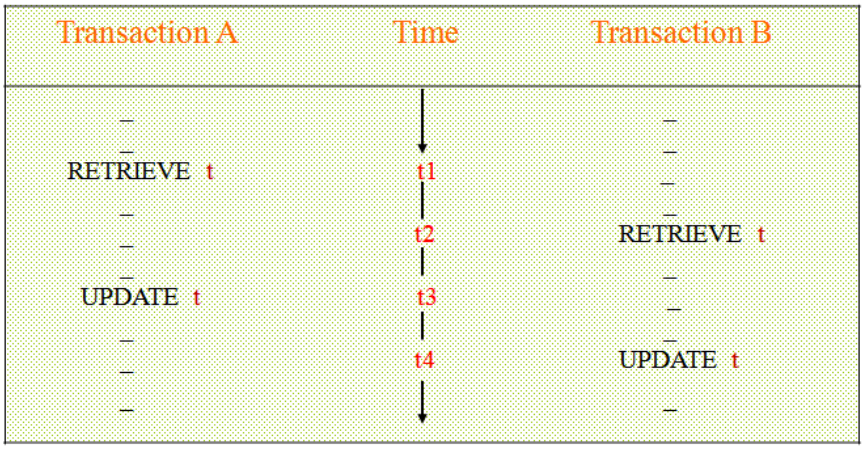
Figure: Lock Table

Releasing a Lock:

A lock can be released using one of the following statements:
1. ROLLBACK statement
2. ROLLBACK to COMMIT statement
3. COMMIT statement

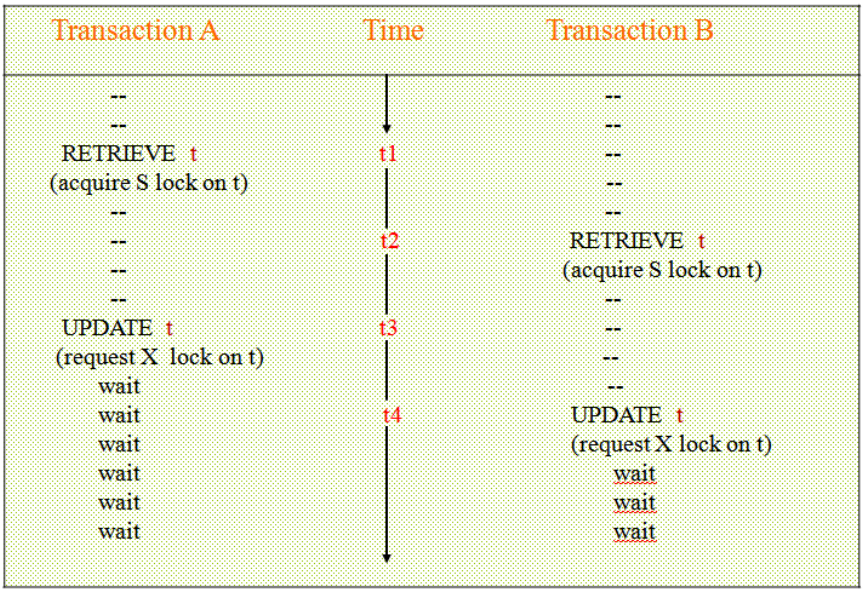
Removing three Concurrency Problems through Locking:

1. The Lost Update Problem



Transaction A losses an update at time t4

Solution of problem

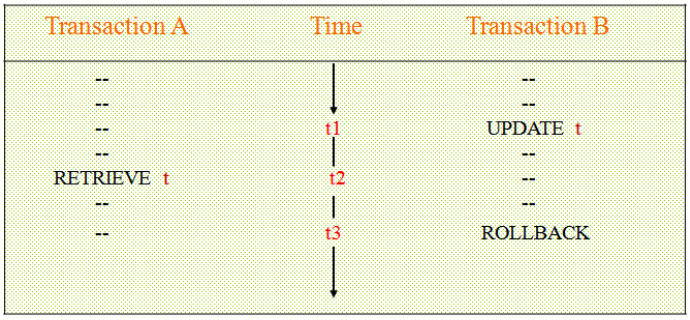


No update is lost, but deadlock occur at time t4

2. The Uncommitted Dependency Problem:

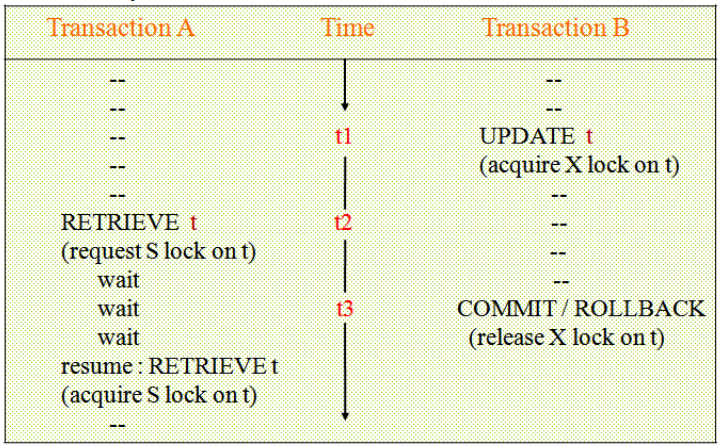
The problem arises if one transaction is allowed to retrieve or update a tuple that has been updated by another transaction but not yet committed by that other transaction.

Problem1-



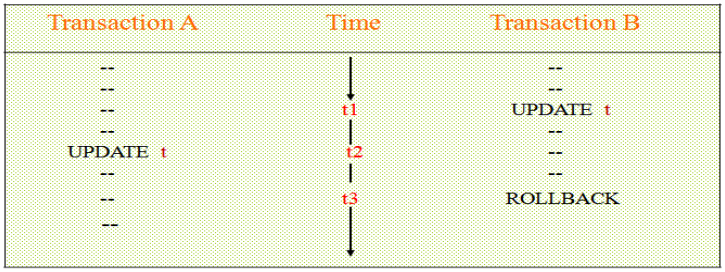
Transaction A becomes dependent on an uncommitted change at time t2

Solution of problem



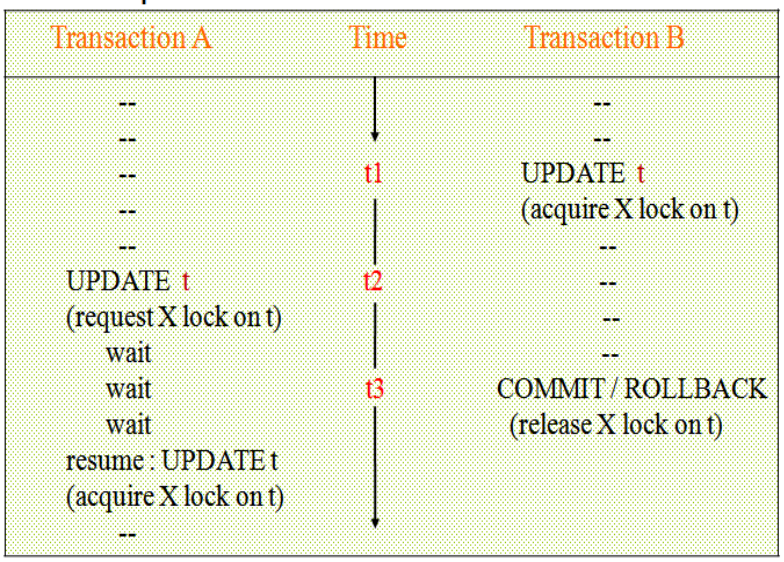
Transaction A is prevented from seeing an uncommitted change at time t2

Problem2-



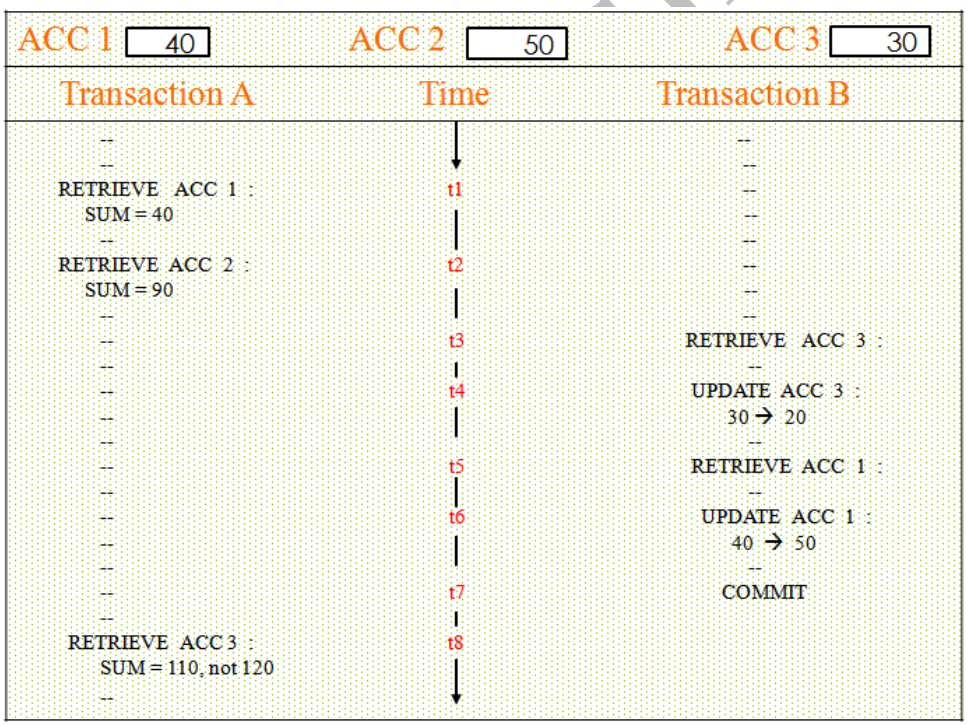
Transaction A updates an uncommitted change at time t2, and losses that update at time t3

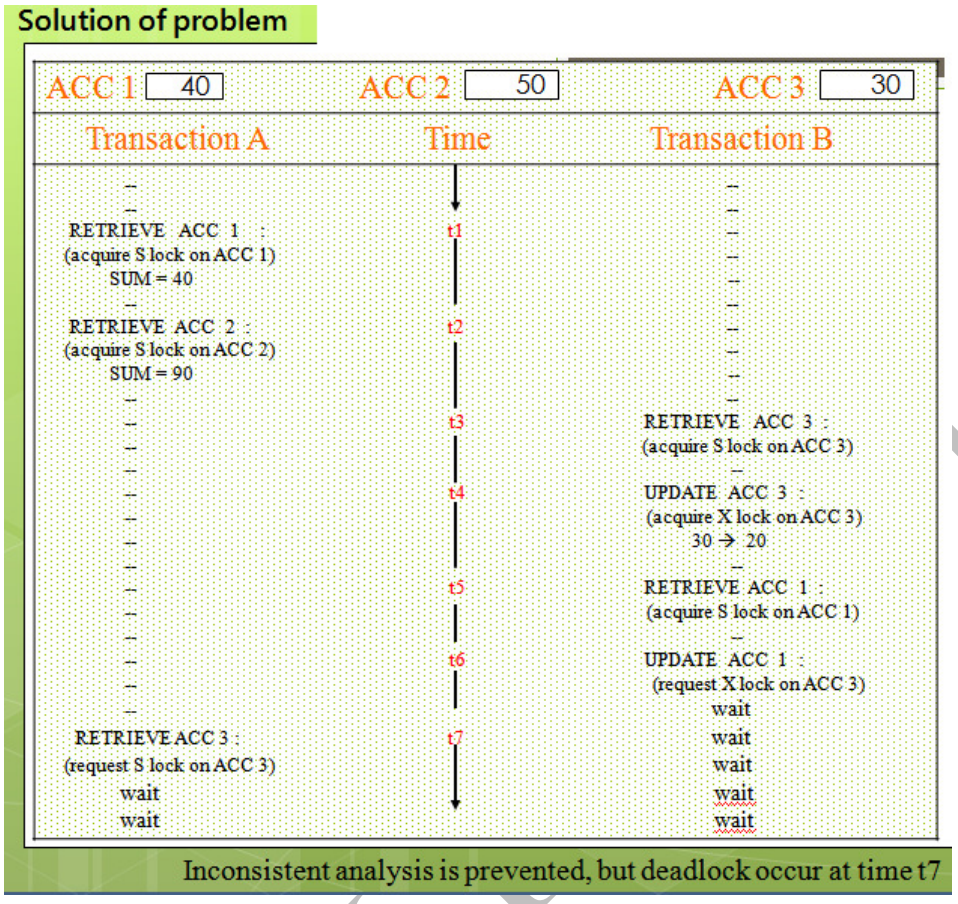
Solution of problem



Transaction A is prevented from updating an uncommitted change at time t2

3: The Inconsistent Analysis Problem:





Intent Locking:

► According to intent locking protocol no transaction is allowed to acquire a lock on a tuple before first acquiring a lock – probably *intent lock* on the relation that contains it.

► We introduce three additional kinds of locks, called intent locks.

- Intent shared (IS) : T intends to set S locks on individual tuples in R.
- Intent Exclusive (IX) : T might update individual tuples in R and set X lock on those tuples.
- Shared (S) : T can tolerate concurrent readers but not updates, in R.
- Shared Intent Exclusive (SIX) : Combine S and IX i.e., T can tolerate concurrent readers but not updates in R & T might updates individual tuples in R and will therefore set X locks on those tuples.
- Exclusive (X) : T cannot tolerate any concurrent access to R at all.

Intent Locking protocols:

There are two intent locking protocols:

1. Before a given transaction can acquire a S lock on a given tuple, it must first acquire an IS on the relation containing that tuple.

2. Before a given transaction can acquire a X lock on a given tuple, it must first acquire an IX on the relation containing that tuple.

	X	SIX	IX	S	IS
X	N	N	N	N	N
SIX	N	N	N	N	Y
IX	N	N	Y	N	Y
S	N	N	N	Y	Y
IS	N	Y	Y	Y	Y

" Compatibility Matrix extended to include intent locks "

Deadlock: (technique for concurrency control)

Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.

Definition:

" Deadlock is a situation in which two or more transactions are in a simultaneous wait state, each of them waiting for one of the others to release a lock before it can proceed. "

Example of Deadlock:

Transaction A	Time	Transaction B
--	↓	--
--		--
LOCK r1 EXCLUSIVE	t1	--
--	|	--
--	t2	LOCK r2 EXCLUSIVE
--	|	--
LOCK r2 EXCLUSIVE	t3	--
wait	|	--
wait	t4	LOCK r1 EXCLUSIVE
wait	↓	wait
wait		wait

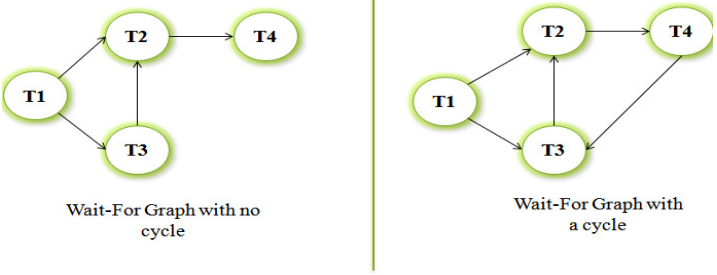
In figure, r1 and r2 are intended to represent any lockable resources & the "LOCK…EXCLUSIVE" statement are intended to represent any operations that request X locks.

Deadlock detection: **Wait-For Graph**

- ► If a Deadlock occurs, it is desirable that the system detect it and break it.
- ► Detecting a deadlock involves detecting a cycle in the **Wait-For Graph.**
- ► Wait-For Graph is a graph of **"who is waiting for whom".**

► Breaking a deadlock involves choosing one of the deadlocked transactions (i.e., one of the transaction in the cycle in the graph) as the **victim** and rolling it back, there by releasing a lock and so allowing some other transaction to proceed.

Wait-For Graph



Wait-For Graph with no cycle

Wait-For Graph with a cycle

Deadlock prevention:

- ► Instead of allowing Deadlocks to occur & dealing with them when they do, it would be possible to avoid them entirely by modifying the locking protocol in various ways.
- ► **Approach of Deadlock Prevention:**
  - → Wait-Die
  - → Wound-Wait

The approach works as follows:

- ► Every transaction is *timestamped* with its start time (which must be unique).

- ► when transaction A requests a lock on a tuple that is already locked by transaction B, then;
  - **Wait-Die:** Transaction A waits if it is older than B; otherwise, it "dies" – that is, A is rolled back and restarted.
  - **Wound-Wait:** Transaction A waits if it is younger than B; otherwise, it "wounds" B – that is, B is rolled back and restarted.
- ► If a transaction has to be restarted, it retain its original timestamp.
- ► The main disadvantage of this approach is that it does too many rollbacks.

Recovery from Deadlock:

To recover a deadlock three steps need to be taken**:**

1. **Selection of a victim**
2. **Rollback**
3. **Starvation**

SQL- Structure Query Language

- In current DBMSs, DDL, VDL, DML are *considered a comprehensive integrated languages.*
- SQL is typical example of a comprehensive database language.
- SQL is the relational database language which represents a combination of DDL, VDL, and DML, as well as statements for constraint specification, schema evolution, and other features.
- The SDL was a component in early versions of SQL but has been removed from the language to keep it at the conceptual and external levels only.

Feature of SQL

- It is a non procedural language.
- It is a 4GL programming language.
- focus only What to do? not How to do?
- It is a case insensitive language.

Classification of SQL Commands

- DDL Commands
- DML Commands
- DCL Commands
- TCL Commands
- Query Language

Creating Table Space

- Used to create a file contain data dictionary of our complete data base.
- Step1:> Connect System/Manager;
- Step2: >CREATE TABLESPACE dept_space DATAFILE ‘dept_space.dat' SIZE 5M;

Create New user

Step1:> Connect system/manager; (if not connected)

Step2:> CREATE USER user_lokesh IDENTIFIED BY password DEFAULT TABLESPACE dept_space;

Step3:> GRANT CONNECT,RESOURCE TO user_lokesh ;

Step4:> Connect user_lokesh/password;

Remove Users

Step1: Connect system/manager;

Step2: DROP USER user_lokesh CASCADE;

Create Table:

```
CREATE TABLE table_name
(
        attr_name1  DataType,
        attr_name2  DataType,
     ……..
    attr_namen  DataType
);
```

Modify Table Attributes

Add column-

ALTER TABLE table_name ADD (new_attr DataType);

Resize/Change DataType-

ALTER TABLE table_name MODIFY

(old_attr newDataType);

Remove Attribute-

ALTER TABLE table_name DROP COLUMN old_attr_name);

Remove table:

DROP TABLE table_name;

Show table descriptions:

DESC table_name;

To list all tables in current usename:

SELECT * FROM tab;

Oracle Data-Type

- CHAR(n) : for fixed size of string.
- VARCHAR(n): for variable size of string up to n (limited size 256).
- VARCHAR2(n): for variable size of string up to n (unlimited size 4000).
- NUMBER(n,m):
- n is total participated digit, m is max digit after point(.)
- DATE: for date and time

Example:

```
CREATE TABLE customer
(
        cust_id NUMBER(4),
        cust_name VARCHAR2(30),
        dob DATE
 );
```

Inserting Data into Table

INSERT INTO <table_name> [<column_lists>] VALUES (<value1>, <value2>, …….);

Using column name specification:

INSERT INTO customer (custid, ename, sal) VALUES('E001','Vipin',5000);

The columns not listed in the insert into command will have their default values or null values.

Without column name specification:

INSERT INTO emp1 VALUES('E001','Vipin',5000);

Here the order of values matches the order of columns in the create table command of the table and giving all values are compulsory.

Showing Records

- All Column + All Record

SELECT * FROM tablename.

- Selected Column and all Records.

SELECT col1,col2,---FROM tablename.

Update Record data

- UPDATE table_name SET column_name1=value, col_name2=value;

Affect on all reords.

Remove All Records

- DELETE FROM table_name;

Only records remove not table.

Applying Constraints

- According to business rule, we apply some rules on column at the time of column definition so that the entire record is rejected and not stored in the table if violet applying constraints.
- NULL (default)
- NOT NULL
- UNIQUE
- PRIMARY KEY

- FOREIGN KEY
- CHECK

All constraint are defined at-

Column Level (At the time of column declaration)

Table Level (After declaration all column)

NOT NULL

- Columns become mandatory.
- Syntax:

  col_name DataType NOT NULL

- Can be used with many attributes.
- Always apply at column level.
- Ex:

  CREATE TABLE customer

  (

  name VARCHAR2(30) NOT NULL

)

UNIQUE Constraint

- A value in specified column never repeated w.r.t.. that domain. Can be used with many attributes.
- Syntax: At column level

  col_name DataType UNIQUE

- Example:

  CREATE TABLE customer

  (phone VARCHAR2(10) UNIQUE,

  voterid NUMBER(12) UNIQUE

);

- Syntax: At Table Level

  UNIQUE(col1,col2,……)

- Example:

```
CREATE TABLE customer
(       phone VARCHAR2(10) ,
        voterid NUMBER(12) ,
        UNIQUE(phone, voterid)
);
```

PRIMARY KEY Constraint

- Behave as a UNIQUE+ NOT NULL Constrains. Using More than one PRIMARY KEY creates composite key.
- Syntax: At column level

  col_name DataType PRIMARY KEY

- Example:

```
CREATE TABLE customer
(       custid NUMBER(5) PRIMARY KEY
);
```

- Syntax: At Table Level

  PRIMARY KEY(col1,col2,……)

- Example:

```
CREATE TABLE customer
(       custid NUMBER(5) ,
        PRIMARY KEY (custid)
);
```

FOREIGN KEY Constraint

- Represent relationship between two tables, Master Table(have PRIMARY KEY) and detail table(have FOREIGN KEY).
- In detail table, a column defined with FOREIGN KEY must be PRIMARY KEY in Master table.
- Syntax: At column level

  col_name DataType REFERENCES mastertable_name

- Example:

```
CREATE TABLE borrower
(       custid NUMBER(5) REFERENCES customer,
        loanid NUMBER(5) REFERENCES loan,
        PRIMARY KEY(custid, loanid)
);
```

- Syntax: At Table Level

  FOREIGN KEY (col_name) REFERENCES mastertable_name

- Example:

```
CREATE TABLE borrower
(       custid NUMBER(5) ,
        loanid NUMBER(5),
        PRIMARY KEY(custid, loanid)
        FOREIGN KEY (custid) REFERENCES customer,
        FOREIGN KEY (custid) REFERENCES customer,
);
```

The CHECK constraints

- Business rule validations can apply. It use logical expression.

- Syntax: At Column Level

  col_name DataType CHECK (logical expression)

- Example:

```
CREATE TABLE customer
(   salary NUMBER(10) CHECK (salary>=3000)
);
```

- Syntax: At Table Level

  CHECK (logical expression)

- Example:

```
CREATE TABLE customer
(   salary NUMBER(10),
    CHECK (salary>=3000)
);
```

Operators

- Apply on attributes names and value.

Arithmetic: +,-,*,/

Gives numeric value.

Ex: SELECT name, “Incremented Income” income+1000 FROM customer.

Relational: <,>,<=,>=,=,<>

Gives Boolean value.

Logical: AND, OR, NOT

Range Searching: BETWEEN

Pattern Matching: LIKE (%, _)

WHERE clause

- Works on all records of given table name.
- It can be used with SELECT, UPDATE, DELETE
- It needs logical condition that consist using relational, logical, range and pattern operator.
- Condition apply on each record, if satisfy then take action on that record. This process continued at last.

Syntax: SELECT /UPDATE/DELETE part WHERE condition;

Relational Operator: (<,>,<=,>=,=,<>)

- Operate on attribute and value, gives true /false value.
- True means action part apply on that record.
- Ex: WHERE cuid=101;

  WHERE income>=10000;

  WHERE name=`name`;

Logical operator: (AND, OR, NOT)

- To combine more then one relational expression.
- AND: Both condition must be true.

Ex: 9999 to 50000

WHERE income>9999 AND income<50000

- OR: Either one condition must be true.

  Ex: less than 9999 or larger than 50000

  WHERE income<9999 OR income>50000

- NOT: Reverse logical condition value.

  Ex: not less than 9999

  WHERE NOT(income<9999)

Range Searching

- WHERE attr_name BETWEEN f AND l.

  f=first value l=last value

  Ex: in between 10000 to 50000 (both value included)

  WHERE income BETWEEN 10000 AND 50000

- It can be reverse using NOT operator.

  Ex: Not in between 10000 to 50000 (both value included)

  WHERE income NOT BETWEEN 10000 AND 50000

Pattern matching

- Apply only with string attribute.
- Possible using LIKE operator and wild card characters(%, _).
- Syntax:

  WHERE attr_name LIKE 'string with wc'

- wc means wild card character % and _.

  % :matches zero or more mismatched character.

  _ :matches only one mismatched character.

- Ex: name begines with ja…..

  WHERE name LIKE 'ja%'

- Ex: name begins with *j* and three are any one.

  WHERE name LIKE 'j_ _ _'

- Both can be combined.
- Ex: First letter any one, second must be j and after it any length of unmatched character.

  WHERE name LIKE '_j%'

IN & NOT IN

- = matches single value. IN matches more value.
- Syntax: WHERE attr_name IN (value1, value2,-----)
- Ex: Match only two given name.

  WHERE name IN('john','merry')
- Can be reverse. Work on record exclude given value.
- Syntax:

  WHERE attr_name NOT IN (value1, value2,-----)
- Ex: Match only two given name.
- WHERE name NOT IN('john','merry')