INTRO

UNIT 1

Data: Known facts that can be recorded which has an implicit meaning
DATABASE: It is a colection of inter-related data which is used to retrieve, insert and delete data efficiently,
DBMS: It is a software used to manage the database. It provides interface to perform various operations like database creation, manipulation and retrieval operations
DATABASE SYSTEM: DBMS software + Database

Typical DBMS Functionality

Defining a database : Specifying the data types, structures, and constraints of the data to be stored in the database. T
Constructing the database: The process of storing the data on some storage medium that is controlled by the DBMS.
Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database, and generating reports from the data.

1. Retrieval: querying, generating reports
2. Modification: insertions, deletions and modifications to its content

Sharing a database allows multiple users and programs to access the database simultaneously.

Other features:

Protection or Security measures measures to prevent from unauthorized access
- **Maintaining** the database and associated programs over the lifetime of the database application

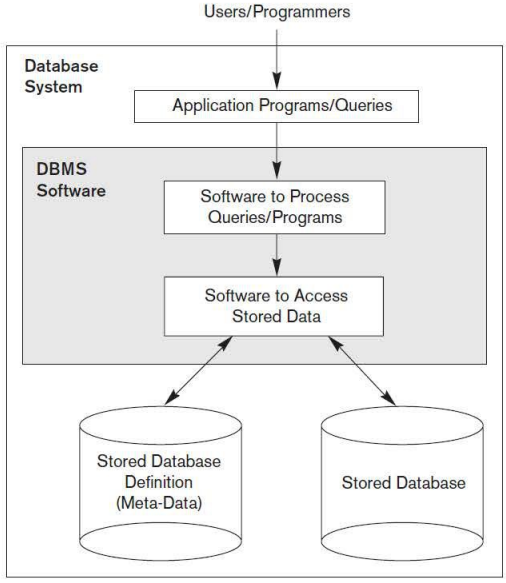


fig : DATABASE SYSTEM ENVIRONMENT

File-based Approach - Limitations / Problems

1. Data redundancy and inconsistency

Multiple file formats are present and we need to handle each type differently, duplication of information in different files (Data redundancy maybe present)

2. Difficulty in accessing data

Need to write a new program to carry out each new task
Multiple users cannot access the same file at a time

3. Data isolation - multiple files and formats

4. Structural Dependence

A change in any file's structure requires the modification of all programs using that file.

5. Data Dependence

A change in any file's data characteristics requires changes in all data access programs.

Characterstics of DBMS

In the database approach, a single repository of data is maintained that is defined once then accessed by various users

1. Self-describing nature of a database system

Database system contains not only the database itself but also a complete description of the database structure and constraints. This definition is stored in the DBMS catalog, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data. The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database

2. Insulation between programs and data:

In file processing, if any changes to the structure of a file may require changing all programs that access the file

In database system, the structure of data files is stored in the DBMS catalog separately from the access program.This property is called program-data independence

3. Support of multiple views of the data:

Each user may see a different view of the database, which describes only the data of interest to that user. (Data abstraction : external(view), conceptual, and physical level schema)

It may also contain some virtual data that is derived from the database files but its not explicitly stored

4. Sharing of data and multi-user transaction processing:

Allowing a set of concurrent users to retrieve from and to update the database
Concurrency control within the DBMS guarantees that each transaction is correctly executed or aborted

There are the following differences between DBMS and File systems:

Basis	DBMS Approach	File System Approach
Meaning	DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	The file system is a collection of data. In this system, the user has to write the procedures for managing the database.
Sharing of data	Due to the centralized approach, data sharing is easy.	Data is distributed in many files, and it may be of different formats, so it isn't easy to share data.
Data Abstraction	DBMS gives an abstract view of data that hides the details.	The file system provides the detail of the data representation and storage of data.
Security and Protection	DBMS provides a good protection mechanism.	It isn't easy to protect a file under the file system.
Recovery Mechanism	DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from system failure.	The file system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will be lost.
Manipulation Techniques	DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	The file system can't efficiently store and retrieve the data.
Concurrency Problems	DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while deleting some information or updating some information.
Where to use	Database approach used in large systems which interrelate many files.	File system approach used in large systems which interrelate many files.
Cost	The database system is expensive to design.	The file system approach is cheaper to design.
Data Redundancy and Inconsistency	Due to the centralization of the database, the problems of data redundancy and inconsistency are controlled.	In this, the files and application programs are created by different programmers so that there exists a lot of duplication of data which may lead to inconsistency.
Structure	The database structure is complex to design.	The file system approach has a simple structure.
Data Independence	In this system, Data Independence exists, and it can be of two types. ◦ Logical Data Independence ◦ Physical Data Independence	In the File system approach, there exists no Data Independence.
Data Models	In the database approach, 3 types of data models exist: ◦ Hierarchal data models ◦ Network data models ◦ Relational data models	In the file system approach, there is no concept of data models exists.
Flexibility	Changes are often a necessity to the content of the data stored in any system, and these changes are more easily with a database approach.	The flexibility of the system is less as compared to the DBMS approach.
Examples	Oracle, SQL Server, Sybase etc.	Cobol, C++ etc.

Need for DBMS

A Data Base Management System is a system software for easy, efficient and reliable data processing and management. It can be used for:

*Creation of a database.
*Retrieval of information from the database.
*Updating the database.
*Managing a database.

It provides us with the many functionalities and is more advantageous than the traditional file system in many ways listed below:

1) Processing Queries and Object Management
2) Controlling redundancy and inconsistency
3) Efficient memory management and indexing
4) Concurrency control and transaction management
5) Access Control and ease in accessing data
6) Integrity constraints:

DBA AND HIS FUNCTIONS

A Database Administrator (DBA) is an individual or person responsible for controlling, maintaining, coordinating, and operating a database management system. Managing, securing, and taking care of the database systems is a prime responsibility.

Role and Duties of Database Administrator (DBA) :

- **Decides hardware –**
  They decide on economical hardware, based on cost, performance, and efficiency of hardware, and best suits the organization. It is hardware that is an interface between end users and the database.
- **Manages data integrity and security –**
  Data integrity needs to be checked and managed accurately as it protects and restricts data from unauthorized use. DBA eyes on relationships within data to maintain data integrity.
- **Database Accessibility –**
  Database Administrator is solely responsible for giving permission to access data available in the database. It also makes sure who has the right to change the content.
- **Database design –**
  DBA is held responsible and accountable for logical, physical design, external model design, and integrity and security control.
- **Database implementation –**
  DBA implements DBMS and checks database loading at the time of its implementation.
- **Database implementation –**
  DBA implements DBMS and checks database loading at the time of its implementation.
- **Query processing performance –**
  DBA enhances query processing by improving speed, performance, and accuracy.
- **Tuning Database Performance –**
  If the user is not able to get data speedily and accurately then it may lose organization's business. So by tuning SQL commands DBA can enhance the performance of the database.

Various responsibilities of Database Administrator (DBA) :

- Responsible for designing overall database schema (tables & fields).
- To select and install database software and hardware.
- Responsible for deciding on access methods and data storage.
- DBA selects appropriate DBMS software like oracle, SQL server or MySQL.
- Used in designing recovery procedures.
- DBA decides the user access level and security checks for accessing, modifying or manipulating data.
- DBA is responsible for specifying various techniques for monitoring the database performance.
- DBA is responsible for operation managements.

BY SHIVKARAMAN

DBMS Languages, Interfaces

Once the design of a database is completed and a DBMS is chosen to implement the database, the first step is to specify conceptual and internal schemas for the database and any mappings between the two levels/schamas

1. Data Definition Language (DDL)

Used by DBA and DB designers

DDL -> DDL is Data Definition Language which is used to define data structures(col attributes)

In DBMS, where no strict separation of levels is maintained, Data Definition Language (DDL) is used to define the internal and conceptual schemas .The DBMS will have a DDL compiler to process the DDL statements

2. Storage Definition Language (SDL)

Used to specify internal schema

Recent RDBMSs: there is no specific language that performs the role of SDL, internal schema is specified by a combination of functions, parameters, and specifications related to storage.

In DBMS, where a clear separation is maintained between conceptual and internal levels:

DDL -> used to specify the conceptual schema only.

SDL -> used to specify the internal schema only

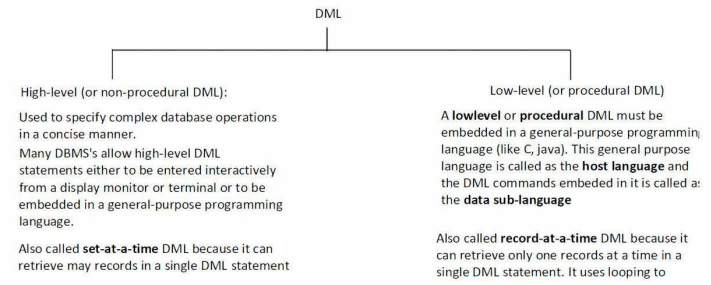
3. View Defination Language (VDL)

For a true three-schema architecture, the view definition language (VDL) is used to specify user views and their mappings to the conceptual schema.

But in most DBMS's the DDL is used to define both conceptual and external schemas.

4. Data manipulation language (DML)

Allows retrieval, insertion, deletion, modification

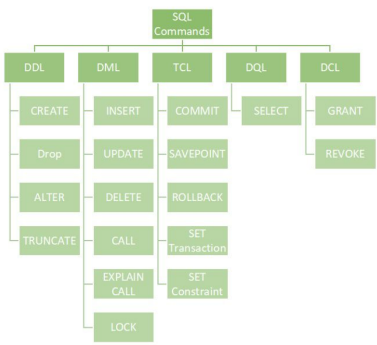


SQL -> Combination of DDL, DML, VDL

TCL (Transaction Control Language):

Transactions group a set of tasks into a single execution unit. Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: success or failure. You can explore more about transactions ***here***. Hence, the following TCL commands are used to control the execution of a transaction:

- **BEGIN:** Opens a Transaction.
- **COMMIT:** Commits a Transaction.
- **ROLLBACK:** Rollbacks a transaction in case of any error occurs.
- **SAVEPOINT:** Sets a save point within a transaction.
- **SET TRANSACTION:** Specifies characteristics for the transaction.



BY SHIVKARAMAN

Instances, Schema and Sub Schema In DBMS

Instances in DBMS / Current state

it is the snapshot of the database taken at a particular moment. It can also be described in more significant way as the collection of the information stored in the database at that particular moment. Instance can also be called as the database state or current set of occurrence due the fact that it is information that is present at the current state.

Every time we update the state say we insert, delete or modify the value of the data item in the record, it changes from one state to other. At the given time, each schema has its own set of instances.

Schema in DBMS

It is the overall description or the overall design of the database specified during the database design. Important thing to be remembered here is it should not be changed frequently. Basically, it displays the record types(entity),names of data items(attribute) but not the relation among the files.
To understand it well, Schema can be assumed as a framework where in the values of data items are to be fitted, these values can be changed but not frame/format of the schema.

Sub schema in DBMS / View

It can be defined as the subset or sub-level of schema that has the same properties as the schema. In simple words it is just a effective plan or the schema for the view. Well, it is interesting to note that it provides the users a window through which the user can view only that part of database which is of matter of interest to him. It Identifies subset of areas, sets, records, data names defined in database that is of interest to him. Thus a portion of database can be seen by application programs and different application programs has different view of data.

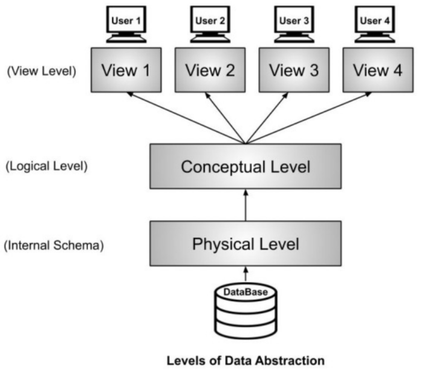
Quickly we can summarize the above things, information/data in database at particular moment is known as instance,physical arrangement of data as it appears in database can be defined as schema, and the logical view of data as it appears to the application can be called as sub schema.

Data Abstraction

Data Abstraction refers to the process of hiding irrelevant details from the user. So, what is the meaning of irrelevant details? Let's understand this with one example. Example: If we want to access any mail from our Gmail then we don't know where that data is physically stored i.e is the data present in India or USA or what data model has been used to store that data? We are not concerned about these things. We are only concerned with our email. So, information like these i.e. location of data and data models are irrelevant to us and in data abstraction, we do this only. Apart from the location of data and data models, there are other factors that we don't care of. We hide the unnecessary data from the user and this process of hiding unwanted data is called Data Abstraction.

There are mainly three levels of data abstraction and we divide it into three levels in order to achieve Data Independence . Data Independence means users and data should not directly interact with each other. The user should be at a different level and the data should be present at some other level. By doing so, Data Independence can be achieved. So, let's see in details what are these three levels of data abstraction:

View Level
Conceptual Level
Physical Level



Levels of Data Abstraction

View Level or External Schema
This level tells the application about how the data should be shown to the user. Example: If we have a login-id and password in a university system, then as a student, we can view our marks, attendance, fee structure, etc. But the faculty of the university will have a different view. He will have options like salary, edit marks of a student, enter attendance of the students, etc. So, both the student and the faculty have a different view. By doing so, the security of the system also increases. In this example, the student can't edit his marks but the faculty who is authorized to edit the marks can edit the student's marks. Similarly, the dean of the college or university will have some more authorization and accordingly, he will has his view. So, different users will have a different view according to the authorization they have.

Conceptual Level or Logical Level
This level tells how the data is actually stored and structured. We have different data models by which we can store the data(You can read more about the different types of data model from here). Example : Let us take an example where we use the relational model for storing the data. We have to store the data of a student, the columns in the student table will be student_name, age, mail_id, roll_no etc. We have to define all these at this level while we are creating the database. Though the data is stored in the database but the structure of the tables like the student table, teacher table, books table, etc are defined here in the conceptual level or logical level. Also, how the tables are related to each other are defined here. Overall, we can say that we are creating a blueprint of the data at the conceptual level.

Physical Level or Internal Schema
As the name suggests, the Physical level tells us that where the data is actually stored i.e. it tells the actual location of the data that is being stored by the user. The Database Administrators(DBA) decide that which data should be kept at which particular disk drive, how the data has to be fragmented, where it has to be stored etc. They decide if the data has to be centralized or distributed. Though we see the data in the form of tables at view level the data here is actually stored in the form of files only. It totally depends on the DBA, how he/she manages the database at the physical level.

RDBMS

BY SHIVKARAMAN

How DBMS store the data?

They use B+ trees. B+ tree is a special data structure allowing to efficiently store (i.e. access and update) a large sorted dictionary on a block storage device (i.e. HDD or SSD).

Sorted dictionary is, essentially, a phone book: it allows to locate a random entry by doing a tiny number of steps - i.e. without reading the whole book.

Block storage device can be described very well by a book too:

It's accessed page-by-page (block-by-block)
There is nearly fixed amount of information on each page (block)
You can quickly find a page (block) by its number without need to read anything else. This isn't quite correct for the book - usually you need to see the numbers of few nearby pages to do this. But let's assume we can do this in 1 step.
Reading next or previous page is very fast. This isn't a required property of block storage, but it's good to remember this, since actual block storages we use (HDDs and SSDs) really have it too, and DBMS rely on this fact.

B+ tree is an algorithm (i.e. a set of rules) allowing not just quickly find the entries in phone book, but also:

Add / edit and delete these entries (assuming you can use an eraser)
Maintain desirable page fill ratio - i.e. ensure that % of used space on all pages (except may be 1) in our phone book is higher than desirable limit (usually 75-90%).

As you know, database is a set of tables, and table is a set of indexes. And each index is, essentially, a B+ tree storing its data.

Few examples of how this structure can be mapped to something we know very well:

1. Files and folders:

Database = some root folder
Tables = its subfolders
Indexes = files in each subfolder

2. Bookcase:

Database = bookcase
Table = a book shelf in the bookcase
Index = a book on a shelf

So how table is actually stored as indexes? Let's assume each table has primary key (PK) - a set of columns uniquely identifying any row there. If some table doesn't, we can always add so-called surrogate primary key - a column storing e.g. a very precise time stamp for any particular row showing the moment it was added at.

Since any table has its own primary key (PK), we can store its data as a dictionary sorted by its primary key in B+ tree structure. E.g. if the original table is this one:

Name	Age	State
John	25	NY
Jane	28	CA
Alex	33	CA
Mike	33	NY

Its *primary index* has this structure:

```
'Alex' -> (33, 'CA')
'Jane' -> (28, 'CA')
'John' -> (25, 'NY')
'Mike' -> (33, 'NY')
```

As you see, I sorted it in alphabetical order: we store indexes as B+ trees, thus any index is a sorted dictionary as well.

So we know each table has at least primary index, which actually stores the table itself. But there can be other indexes called secondary indexes, that map selected set of columns (secondary index key) to a primary key. Let's see how index on State column (let's call it "IX_State") looks like:

```
'CA' -> 'Alex'
'CA' -> 'Jane'
'NY' -> 'John'
'NY' -> 'Mike'
```

It's sorted by State column first, and then by primary key (Name column): normally any index is fully sorted by its key and all columns from the primary key, which aren't part of the secondary key.

Also note that there is no Age column in this index - later I'll show some consequences of this.

How DBMS query the data efficiently?

Let's see how database engine executes this query:

```
Select Name, State
from People
where State = 'CA'
order by Age
```

Query translator transforms it to nearly this execution plan:

```
a = a sequence of all entries from People's table primary index
b = a sequence of entries from a, but where State = 'CA'
c = a sequence of entries from b, but sorted by Age

d = a sequence of entries from c, but only Name and State columns
result = d
```

a, b, c and d are sequences here - i.e. programming abstractions allowing to enumerate all the entries produced by each stage of query execution (an operation of the query execution plan).

> Most of operations don't actually store these entries - in fact, they act like Unix pipes sequentially consuming some input and producing the output. Such operations are called streaming operations. Note that index range scan (AKA "index seek") and full index scan (AKA "full scan") are streaming operations, i.e. when a single entry or a sequential range of entries is extracted from index, DBMS doesn't need to even store the result.
>
> But there are some non-streaming operations - they need to read and process the whole result before producing any output. E.g. sorting is non-streaming operation.

To fully clarify these concepts, let's see how above plan could look like, if our database engine would be written in Python:

```
a = db.get_index('People', 'PK').all()
b = (entry for entry in a if entry['State'] == 'CA')
c = sort(b, 'Age')
d = select(c, 'Name', 'State')
return d
```

> a, b and d are streaming operations here: Index.all() extracts index range (full index), and two others are just Python sequences
> c is non-streaming operation, since it requires the whole sequence to be loaded and sorted before it produces the first entry of result.

So it's easy to estimate how much RAM a particular query needs to execute: it's roughly the amount of RAM needed to store all intermediate results of non-streaming operations in query plan. You can assume virtual RAM is used when intermediates don't fit in physical RAM. Actually it's more complex - there are on-disk temporary tables, etc., but it's nearly the same in terms of expected performance.

You may find this is a suboptimal plan: it implies DBMS must read all entries from People's primary index, and also, store full c operation result, since sort is non-streaming operation.

But the good thing is: DBMS passes this initial query plan to a query optimizer, which should transform it to a nearly optimal plan. In our case this plan should be this:

```
a = entries from People's table IX_State secondary index with 'CA' key
b = a joined with People's table primary index by Name column
c = b sorted by Age
d = select only Name and State columns from c
result = d
```

Or as Python code:

```
pk = db.get_index('People', 'PK')
ix_state = db.get_index('People', 'IX_State')

a = ix_state.range(point('CA').inclusive(), point('CA').inclusive())
b = (pk.get(entry['Name']) for entry in a)
c = sort(b, 'Age')
d = select(c, 'Name', 'State')
return d
```

Why this plan is better? Because it processes only a limited number of rows - precisely, all rows with State == 'CA'. Unoptimized query plan processes all rows on steps a and b, so we can predict that optimized plan executes ~ 50x faster (assuming there are 50 states, and people are evenly distributed among them).

This was a very rough description of how DBMS work. Actually:

There are other types of indexes - e.g. hash and bitmap indexes. But they are used pretty rarely - e.g. InnoDB engine in MySQL supports just B+ trees, and as you know, many of top internet sites (Facebook, Wikipedia, and Quora as well) don't need others to run smoothly. Probably the only other type of index you need to know about is R-tree, which helps to query spatial data (so apps like Google Maps use similar indexes).

What is the meaning of the word Relational in RDBMS?

The term "relational" in RDBMS stands for "relational database management system." It refers to a type of database management system (DBMS) that is based on the relational model of data.

The relational model of data organizes data into tables, with each table consisting of a set of rows (also called "records" or "tuples") and columns (also called "attributes" or "fields"). The tables can then be related to each other through common data elements, known as "keys."

In a relational database, the relationships between tables are defined through the use of foreign keys and primary keys. The primary key of a table is a unique identifier for each row, while the foreign key is a reference to the primary key of another table.

The use of a relational model allows for efficient data retrieval and manipulation through the use of SQL (Structured Query Language) commands. This makes it easier for developers to manage data and build applications that rely on accurate, consistent data storage and retrieval.

DATA MODELS

Data Model : Collection of concepts that can be used to describe the structure of a database (Structure of the database typically include data types, relationships and constraints even basic operations).
It provides necessary means to achieve this abstraction

Categories of data-models

1. Conceptual (high-level, semantic) data models:

-Provide concepts that are close to the way many users perceive data.
-Also called entity-based or object-based data models.
-Describes the database at a very high level
Eg : ER Diagram

2. Physical (low-level, internal) data models

Provide concepts that describe details of how data is stored in the computer.
Eg. Record format, record ordering, access path, index etc.,

3. Implementation (representational) data models

Provide concepts that fall between the above two, used by many commercial DBMS implementations
It is used to represent only the logical part of the database and hide the physical structure of database. It primarily focuses on the design part of the database

e.g. relational data model
Concepts are easily understood by end users.

4. Self-Describing Data Models

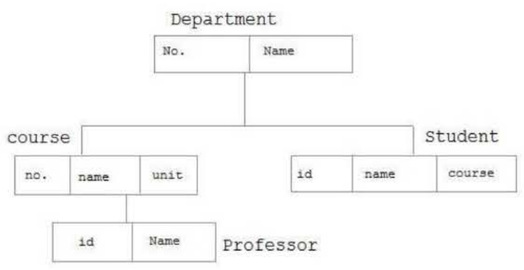
Data storage combines the description of the data with the data values themselves.
In traditional DBMS both are separated.
Ex: XML, NoSQL Systems (Big Data Applications using Mongo DB)

REPRESENTATIONAL DATA MODEL

Represents data by using record structures

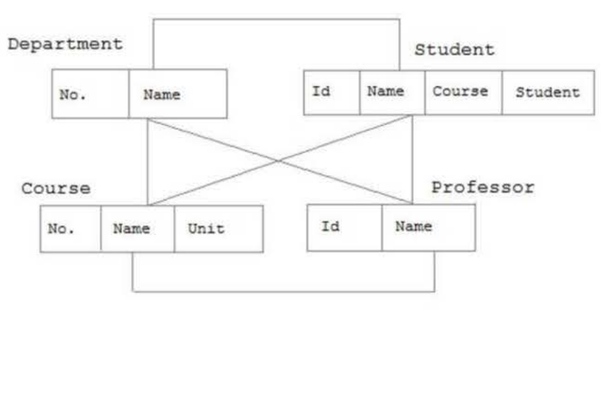
HIERARCHIAL MODEL

Data is organized in a tree like structure, each entity has only one parent but can have several children
At the top hierarchy there is only one entity which is root



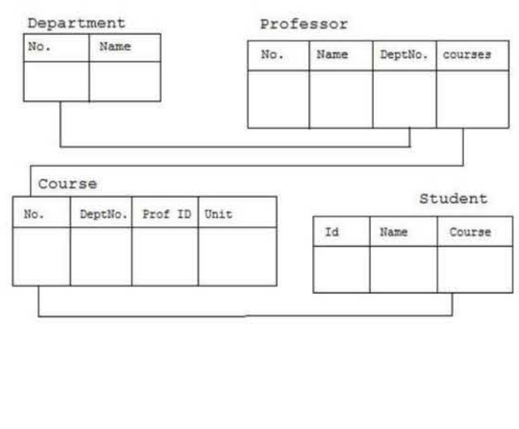
NETWORK MODEL

Entities are organized like a graph, in which some entities can be accessed through several path



RELATIONAL MODEL

Data is organized in two dimensional tables called relation.
The tables or relation are related to each other



Conceptual data models

Provides the concepts of entities, relationships and attributes

Entity is an object which has logical or physical existence
Attributes represents some property of interest that further describes an entity

Database Schema or Intension : The description of a database fields or attributes. It is specified during database design and is not expected to change frequently
Most data models have certain conventions for displaying schema

Schema diagram : Diagramatic representation of database schama

Schema construct: A component of the schema or an object within the schema
e.g., STUDENT, COURSE.

Database State or Extension: The actual data stored in a database at a particular moment in time. This includes the collection of all the data in the database (Also called database instance (or occurrence snapshot).

Empty state: When new database is defined we specify its schema only at this point corresponding state is empty state.
Initial Database State: Refers to the database state when it is initially loaded.
Every time an update operation is applied to database, we get another database state.
Current state : At any point in time the database has current state
Valid State: A state that satisfies the structure and constraints of the database.

DBMS stores schema constructs and constraints together as meta-data in DBMS catalog.
Schema evolution: adding new field to the database schema

Three Schema Archetecture

Goal: to separate the user applications from the physical database. The user application is at the top most level and the physical database is at the bottom most level

- Objectives of Three-Schema Architecture
  - All users should be able to access same data.
  - separate each user's view of the database from the way it is physically represented.
  - Users should not need to know physical database storage details.

Scehmas can be defnied at 3 levels

1. Internal Level:

   Internal level has an internal schema.
   It **describes the physical storage structure** of the database

   The internal schema uses a **physical data model** and describes the complete details of data storage and access paths for the database.

2. Conceptual level

   Conceptual level has a conceptual schema
   It **describes the structure of the whole database**

   A **representational data model** is used to describe the conceptual schema

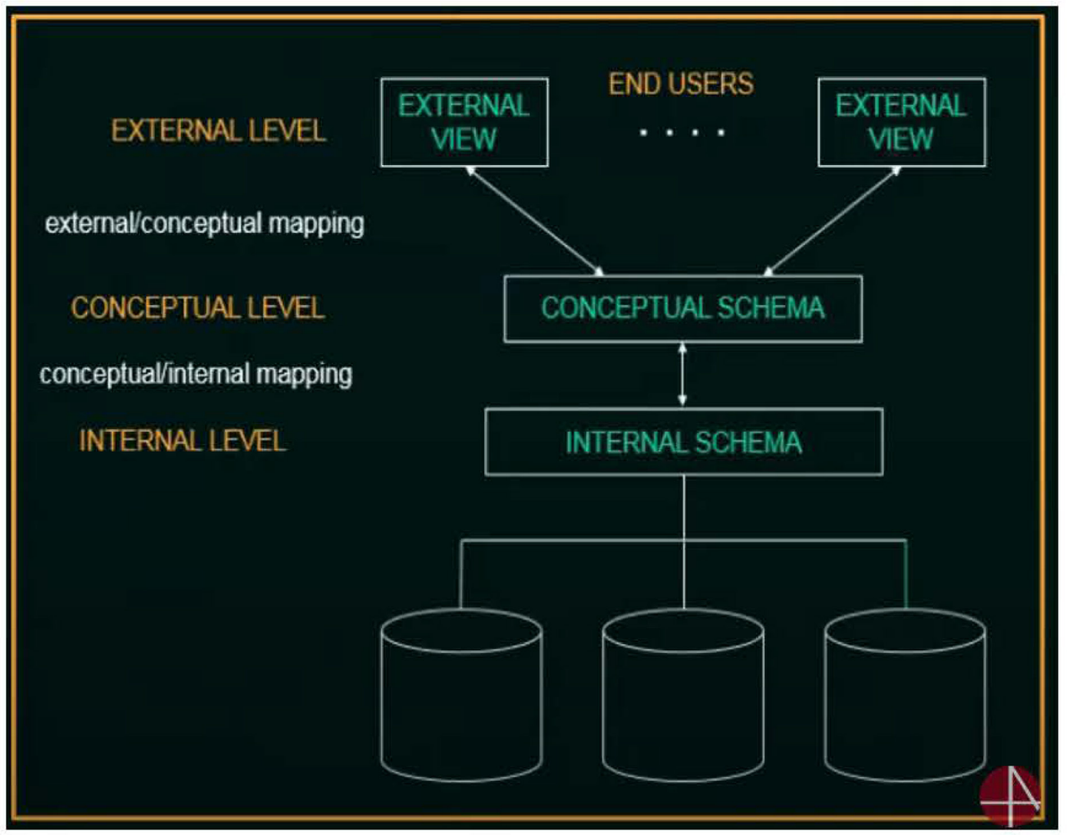
   Hides the details of the physical storage structure and concentrates on describing entities, data types, relationships, constraints, etc.

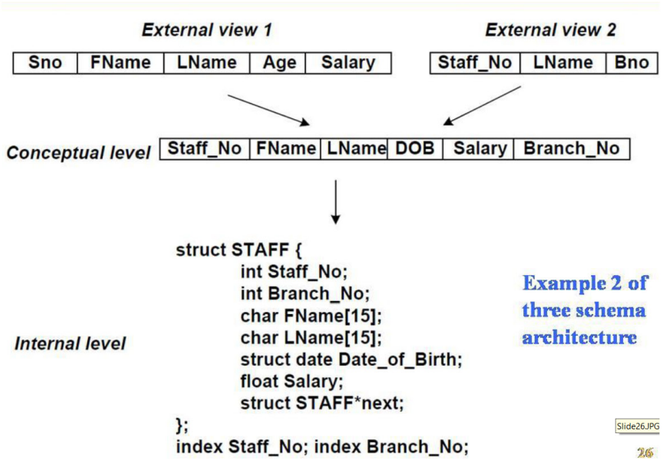
3. External Level:

   It includes a number of **external schemas** or **user views**. Each external schema is typically implemented using a **representational data model**

   It 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

   This level has a view for each user group





DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database.The processes of transforming requests and results between levels are called **mappings**

Data Independence

Definition: Capacity to change the schema at one level of a database system without having tg change the schema at the next higher level.

1. Logical Data Independence:

Works between conceptual schema and external schema
It is the ability to modify the conceptual schema without changing the external schemas or application programs.

After the conceptual schema undergoes a logical reorganization, application programs that reference the external schema constructs must work as before.

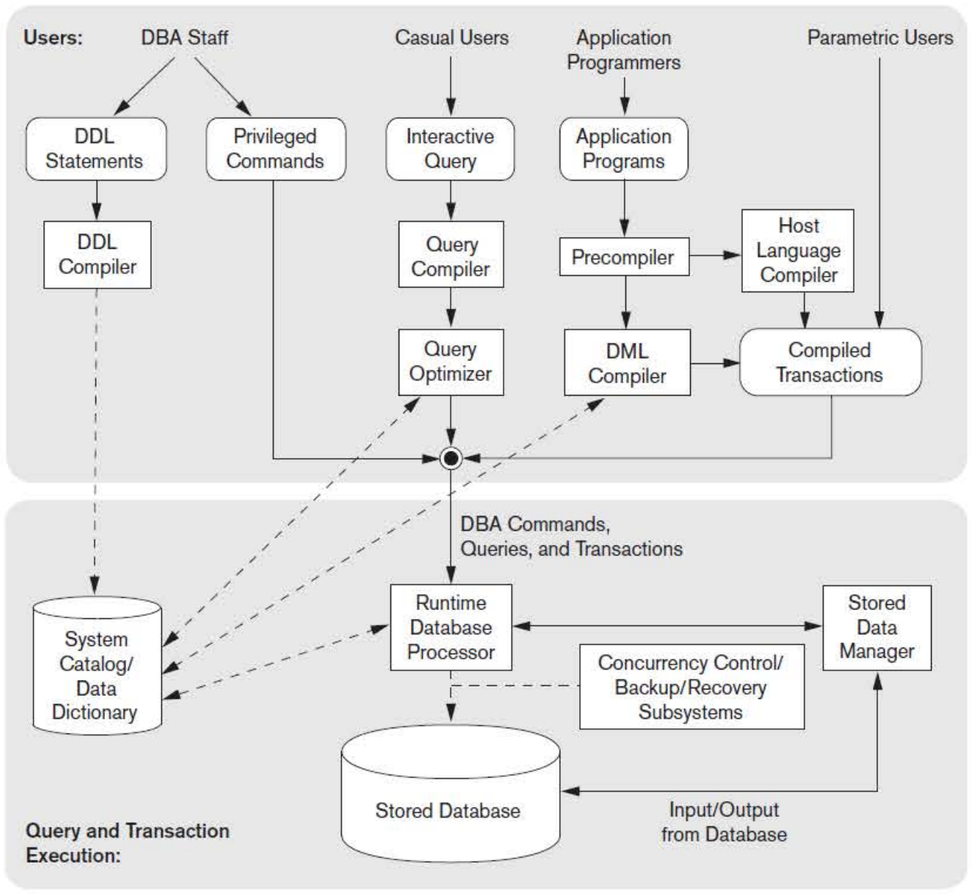
2. Physical Data Independence:

Works between conceptual schema and internal schema.
Ability to modify the internal schema without changing the conceptual schema.
(changes in internal schema could be changing the file loaction or changing the access path etc)
These changes may be required to improve performance

Generally, physical data independence exists in most databases and file environmentsApplications remain unaware of these details. On the other hand, logical data independence is harder to achieve because it allows structural and constraint changes without affecting application programs.

DBMS COMPONENT MODULES

The figure is divided into two parts.
1. The top part of the figure refers to the various users of the database environment and their interfaces.
2. The lower part shows the internal modules of the DBMS responsible for storage of data and processing oftransactions.



The **DBA staff** works on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.
The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints.

Casual users and persons with occasional need for information from the database interact using the **interactive query** interface. The query compiler analyses the queries. checks for syntax errors and compiles them into internal form. The internal query it thenm sent to the query optimizer for optimization. The **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of efficient search algorithms during execution. It consults the system catalog for statistical and other physical information of stored data and generates executable code and makes calls on the runtime processor.

Application programmers write programs in host languages such as Java, C, or C++that are submitted to a precompiler. The **precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access.
The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor.

In the lower part

The **runtime database processor** executes the privileged commands, the executable queries, and the canned transactions
with runtime parameters.
It works with the **system catalog** and may update.
It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.

concurrency control and **backup and recovery systems ->** They are integrated into the working of the runtime database processor for purposes of transaction management

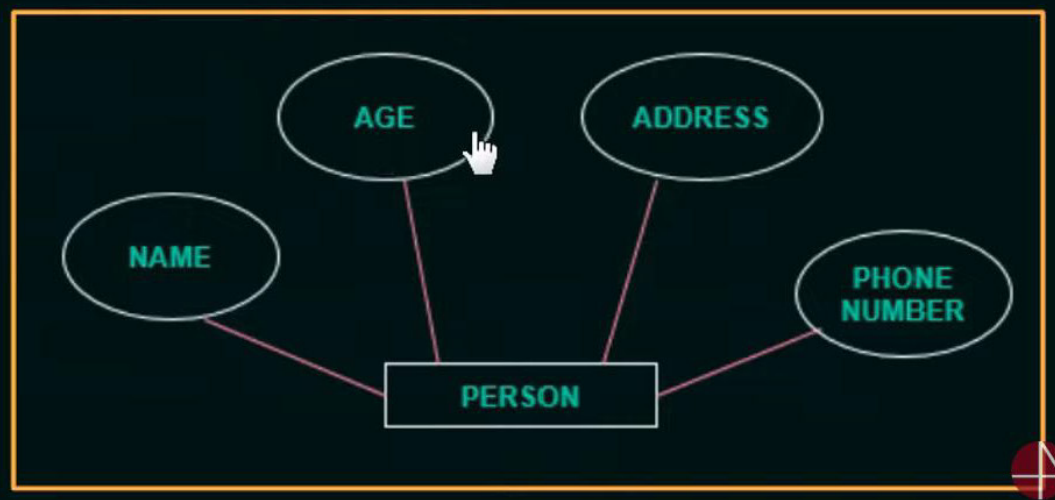
❖ Terminologies

★ Entity:

- A "thing" in the real world with an independent existence.
- May be an object with physical existence (ex: house, person) or with a conceptual existence(ex: course, job).

★ Attributes:

- Properties that describe the entities.



Types of Attributes

Composite Attributes

★ Can be divided into further parts.

★ Ex: Name → First Name, Middle Name, Last Name

Simple Attributes

★ Cannot be divided further.

★ Ex: Weight → cannot be further divided.

Single-Valued Attributes

★ Have a single value for a particular entity.

★ Ex: Age → single-valued attribute of a person.

Multivalued Attributes

★ Can have set of values for a particular entity.

★ Ex: College degree, languages known → multivalued attributes of a person.

nesoacademy.org

Derived Attributes

★ Can be derived from other attributes.

★ Ex: Age → can be derived from date of birth.

Stored Attributes

★ From which the value of other attributes are derived.

★ Ex: BirthDate of a person

nesoacademy.org

★ Complex Attributes:

- Has multivalued & composite components in it.
- Multivalued attributes → represented within '{ }'.
- Composite attributes → represented within '( )'.
- Ex: {CollegeDegrees(College, Year, Degree, Field)}

★ Null Values:

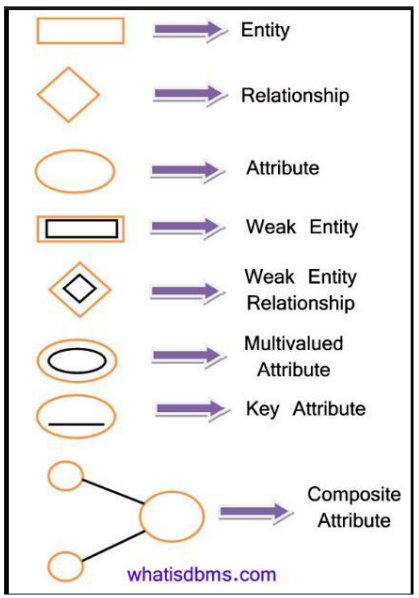
- Null is something which is not applicable or unknown.

STUDENT_ID	NAME	AGE	PHONE_NO
1	HARRY	20	123546897
2	JEREMY	22	NULL
3	JOSHUA	NULL	2525468912

nesoacademy.org

Key Attribute:

- That attribute that is capable of identifying each entity uniquely.
- Ex: Roll number of a student



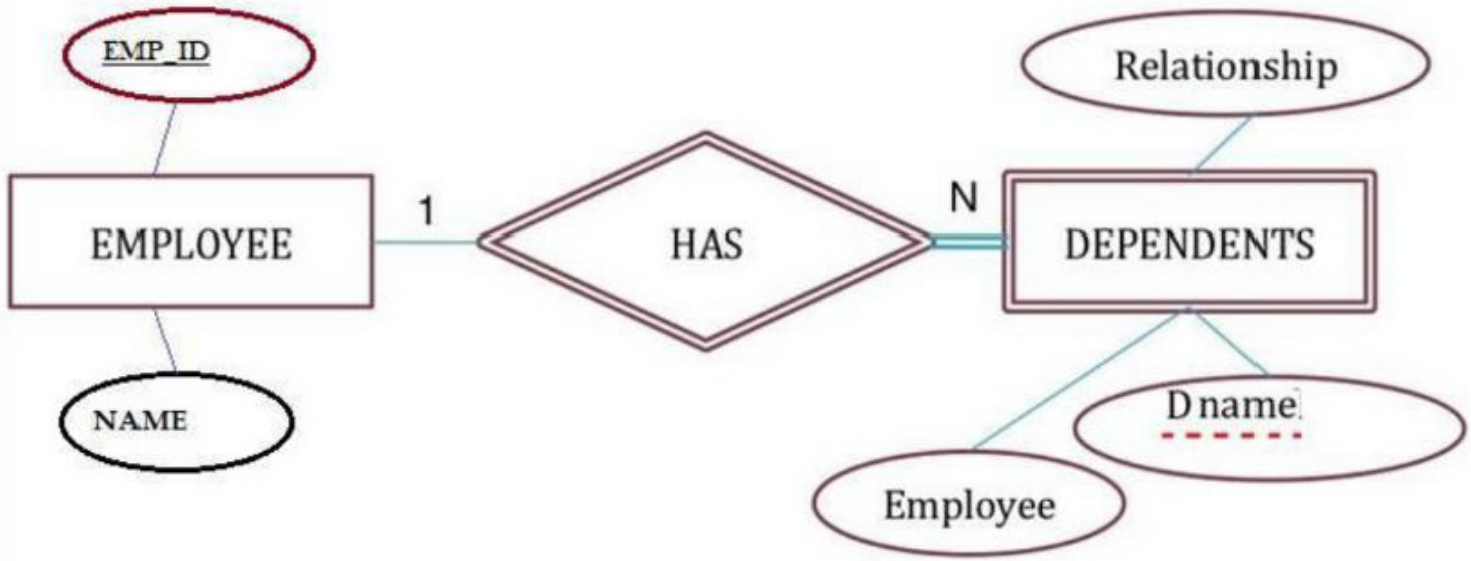
Weak Entity Types

- Entity types that do not have key attributes of their own.
- Identified by relating to another entity type called the identifying or the owner entity type.
- Relationship between weak entity type to its owner → identifying relationship.

- For example:employee's dependents might be weak entities
  - They cannot exist (in the database) if the relevant employee does not exist
  - If an employee is deleted, his dependents are deleted too

3.5 Weak Entity Types

Weak entities have always a total participating constraint because they cannot be identified without an owner entity.



❖ Relationships

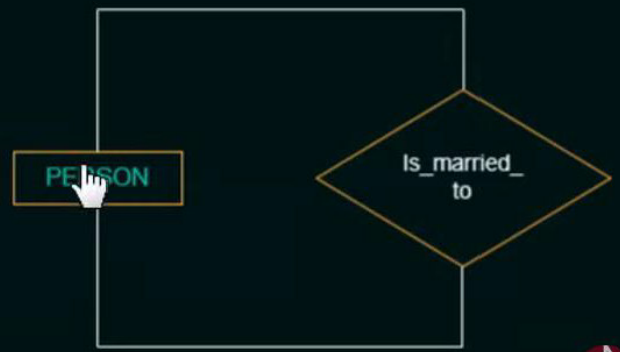
- Relationship → association among 2 or more entities.
- Ex: teacher teaches student

  ↓

  relationship

★ Degree of Relationship: Denotes the number of entity types that participate in a relationship.

1. Unary relationship:
   - ➢ Exists when there is an association with only one entity.

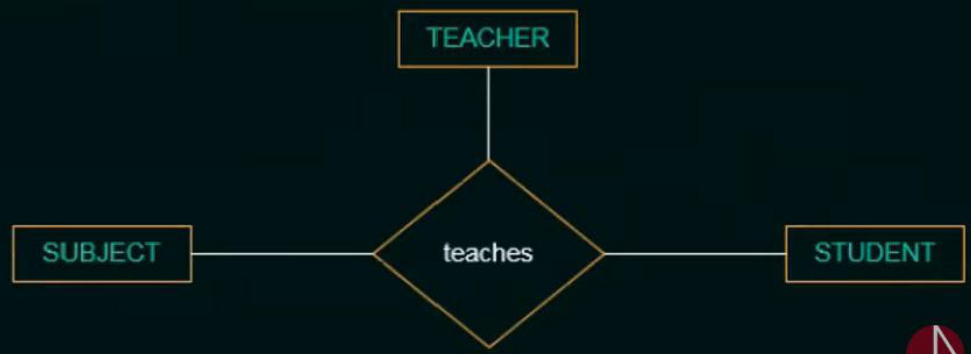


❖ Relationships

★ Degree of Relationship:

2. Binary relationship:
   - ➢ Exists when there is an association among two entities.

3. Ternary relationship:
   - ➢ Exists when there is an association among three entities.

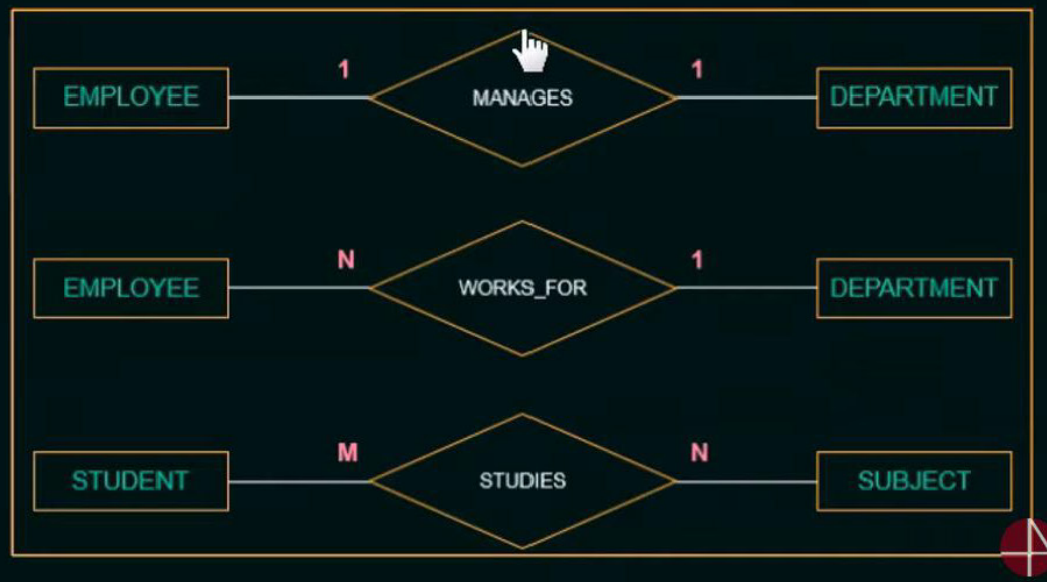


❖ Relationships

★ Relationship Constraints:

1. Cardinality Ratio

- ➢ Maximum number of relationship instances that an entity can participate in.
- ➢ Possible cardinality ratios for binary relationship → 1:1, 1:N, N:1, M:N.



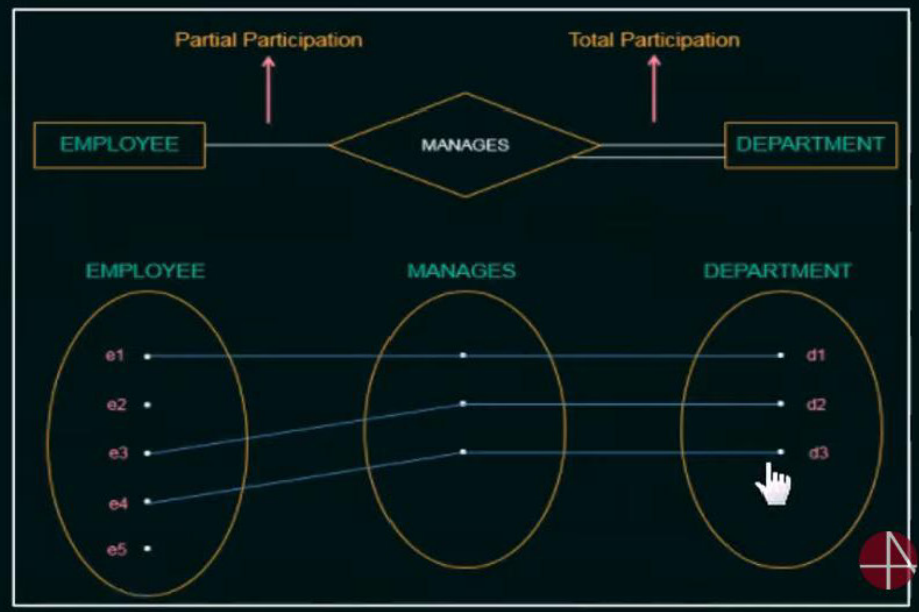
❖ Relationships

★ Relationship Constraints:

2. Participation Constraints

➢ Specifies whether existence of an entity depends on its being related to another entity.

➢ 2 types: Total participation & Partial participation.



Keys

1 Candidate Key, PRIMARY KEY, Alternate Key

An attribute whose value is unique and is capable of identifying an entity is called as key attribut
Eg: Roll no, aadhaar no, email id etc.

Collection of all these key attributes → CANDIDATE KEY

Most appropriate key selected from the candidate key → PRIMARY KEY

Rest of keys in candidate key → Alternate key's

2 Foreign Key

It is an attribute or set of attributes that references to Primary key of same table or another table (in relation).

MAINTAINS REFERENTIAL INTEGRITY

Eg:

Primary Key

Roll no	Name	Address
1	A	Delhi
2	B	Mysore
3	C	Bang

Referenced table

Base table

STUDENT

Course id	Course name	Roll no
C1	DBMS	1
C2	CN	2

Foreign Key

Referencing table.

COURSE

~~Referenced table → Table which is referred by~~ ie ~~which gets~~

Referenced Table

- Insertion → No violation
- Deletion → May cause problem (If student with same PK is present / assigned a course in Referencing table, integrity is lost as student deleted from referenced table, but student is still present in referencing table.

  Solution → Delete row from both tables.
  OR
  Put NULL in Course table.
- Update → If we change PK of a student in Referenced table, it is not changed in referencing table. Hence integrity is lost

  Solution → Update in both table

Referencing table

- Insertion → May cause violation when student with roll no is not at all present in Student / Referenced table.
- Deletion → No Violation
- Updation → May cause ~~updation~~ violation.

~~3 Super key~~

3 Super key

It is a combination of all possible attributes with a candidate key.

OR

Combination of any key in which atleast one Candidate key is present.

Eg: Student (roll no, name, address, Ph no).

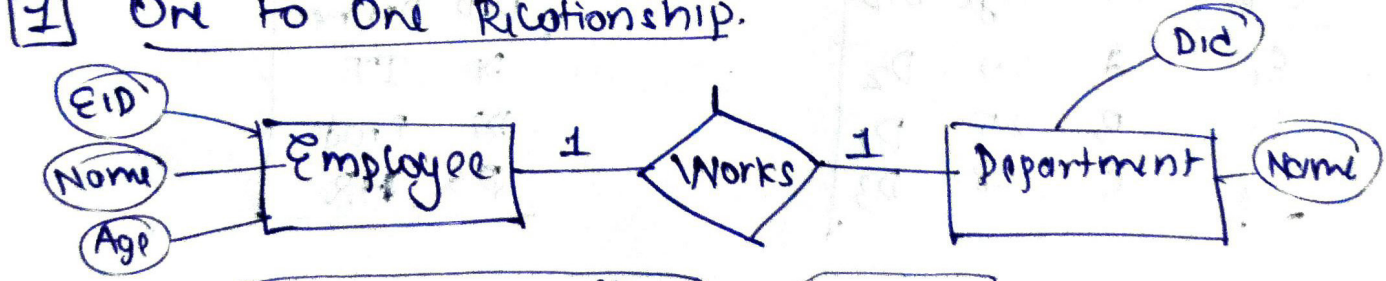
Candidate key → Roll no

Super keys → roll no, address
roll no, name
roll no, address, name

- ## Degree / Types / Cardinality of Relationships

Tells how entities are related to each other.

1 One to One Relationship.



PK ← Employee

EID	Name	Age
E_1	A	20
E_2	B	25
E_3	C	28

fk fk

EID	DID
E1	D2
E2	D1
E3	D3

PK ← Department

Did	Dname
D1	IT
D2	Production
D3	HR

↓

Relationship table

1) What will be the attributes of Relationship table?
- It must contain Primary keys of both Entities and other discriptive attributes.

2) How will you fill the relationship table?
- Since relation is one-one one value of Employee is atmost related to one value of department

3) What is the primary key of Relationship table?
- Either EID or DID as both has unique values.

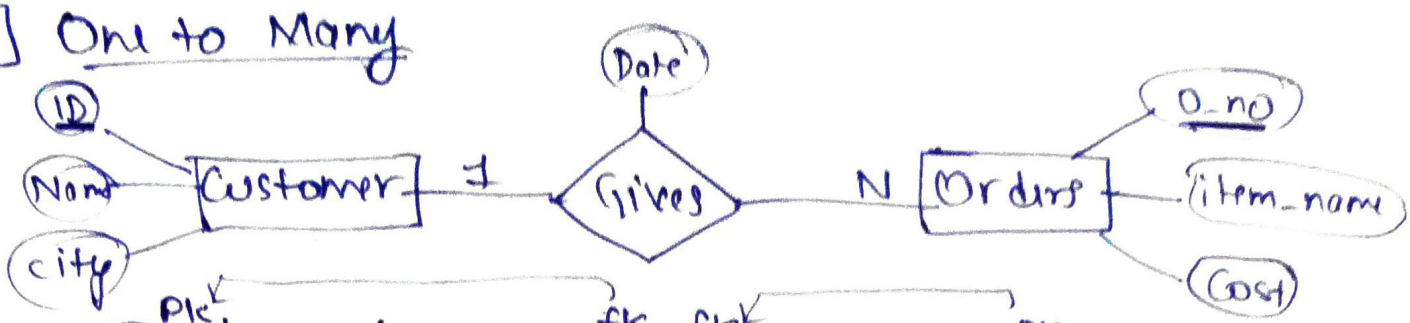
4) Can you reduce the no of tables?
- Yes
- Sina we made EID as primary key, we can merge the column of DID of relationship table with Employee table.

EID	Name	Age	DID
E_1	A	20	D_2
E_2	B	25	D_1
E_3	C	28	D_3

DID	DName
D1	IT
D2	Prod
D3	HR

PK → Either of entities
Merge → with table of PK

2 One to Many



PK ↓

ID	Name	city
C_1	A	Del
C_2	B	Mumbai
C_3	A	Agra

fk fk ↓

ID	O_no	Date
C1	O1	
C1	O2	
C_2	O_3	
C_2	O4	

PK

O_no	Item_name	Cost
O_1	Shoes	1000
O_2	Shirt	750
O_3	Pant	1200

Attribute of Relation → Discriptive Attribute

Here → 1 customer can give any no of orders
→ 1 product can be ordered only once.

PK of Relation Table. ⇐ Unique in Relation Table.

- Can you reduce the no of tables
  - Yes
  - Merge with Orders

PK → PK of Many Side

Merge → With Many side.

3 Many to one

* Completely same as one-to-many

* PK → PK of Many side
  Merge → with Many Side.

❖ Relationships

★ Role Names

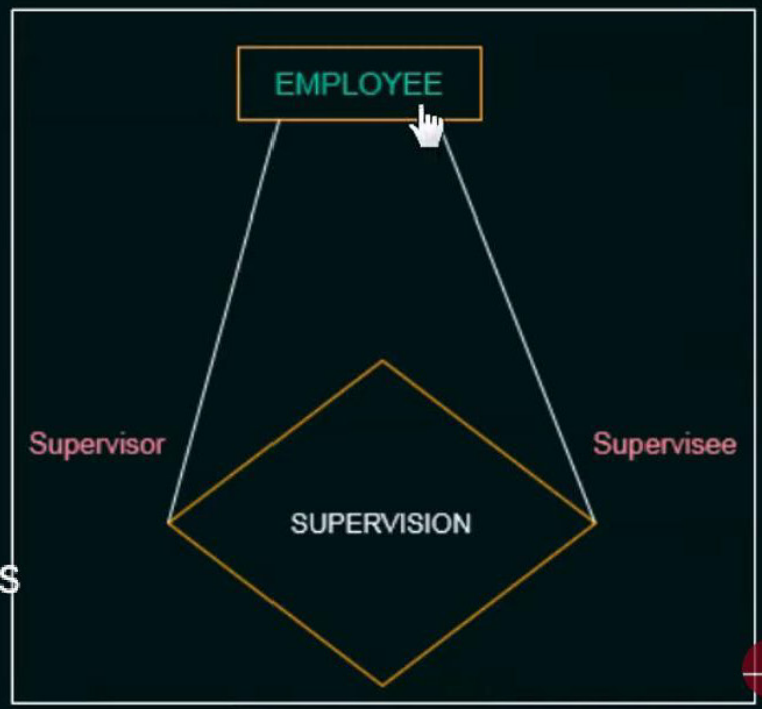
- Signifies the role that a participating entity plays in each relationship instance.

★ Recursive Relationships

- Same entity type participates more than once in a relationship type in different roles.

EMPLOYEE participates twice in two distinct roles

1. Supervisor(Boss role)
2. Supervisee (Subordinate role)



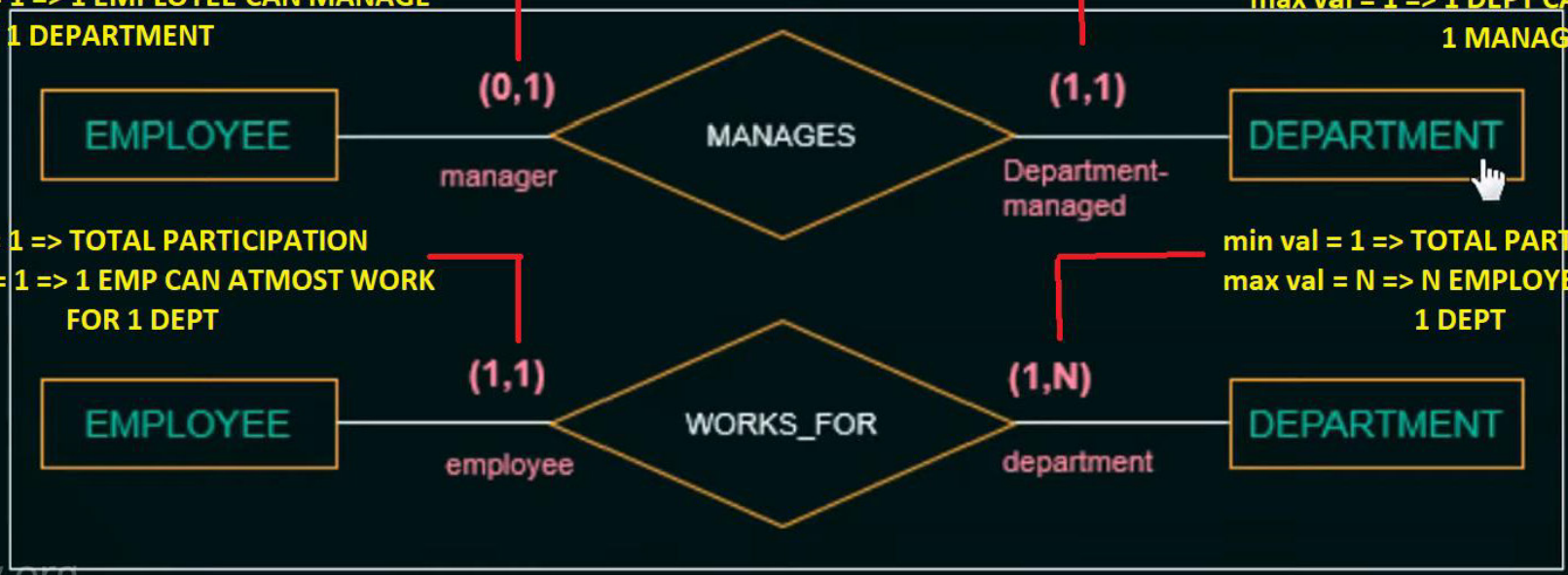
❖ Alternative Notations for ER Diagrams

- Associates a pair of integer numbers (min, max) with each participation of an entity type in a relationship type, where

$0 \leq min \leq max$ and $max \geq 1$.

min value = 0 => PARTIAL PARTICIPATION
min value > 0 => TOTAL PARTICIPATION

max value specifies max no of relationship instances an entity participates in



3.3.2 Entity Types, Entity Sets, Keys and Value Sets

- **Entity Types and Entity Sets**
  - **Collection of entities** all having the same properties.
  - ***Student entity set*** *–collection of all student entities.*

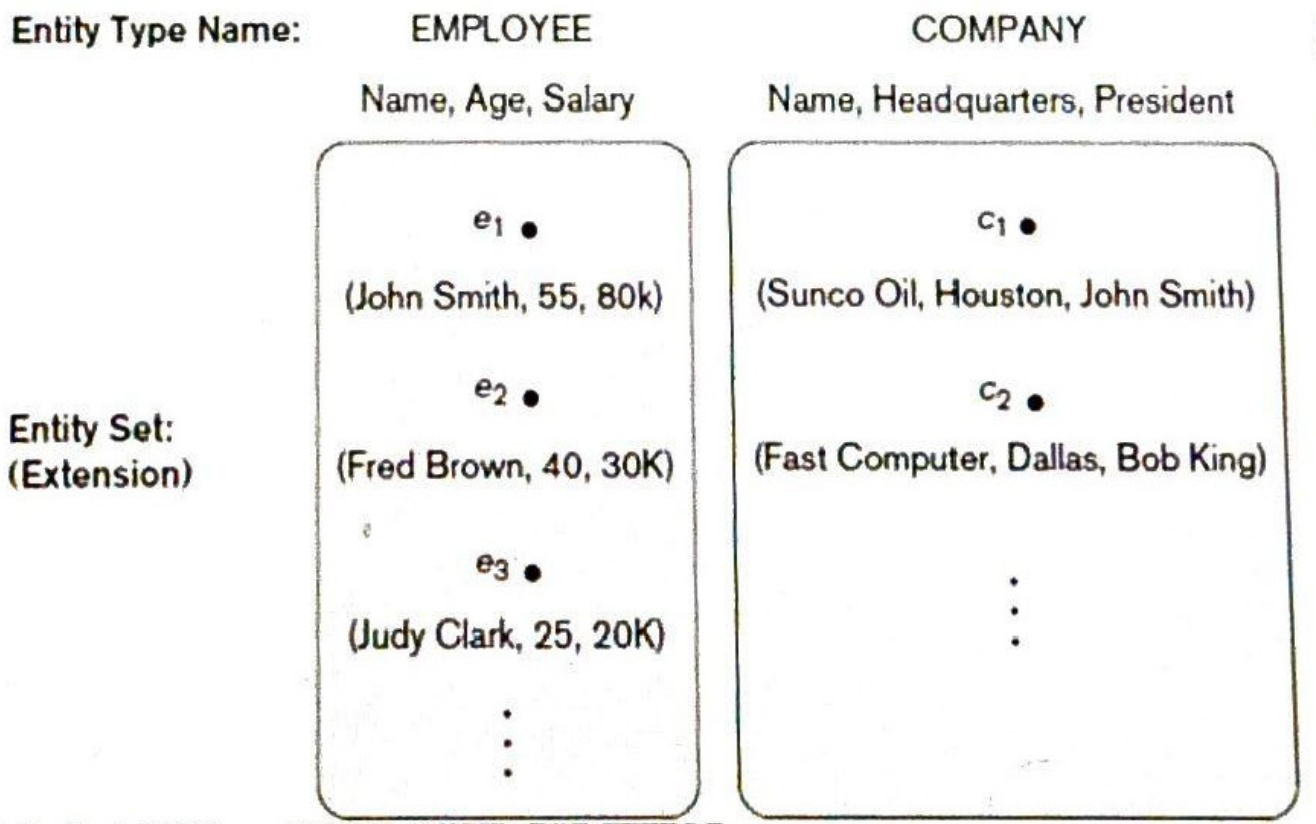


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

- **Value Sets (Domains) of Attributes**

  - Each attribute is associated with a **value set (or domain of values), which specifies the set of values.**

  - **Ex:** age of employees between 18 and 60

  - Domains are typically specified using the basic data types available such as integer, string, etc.,

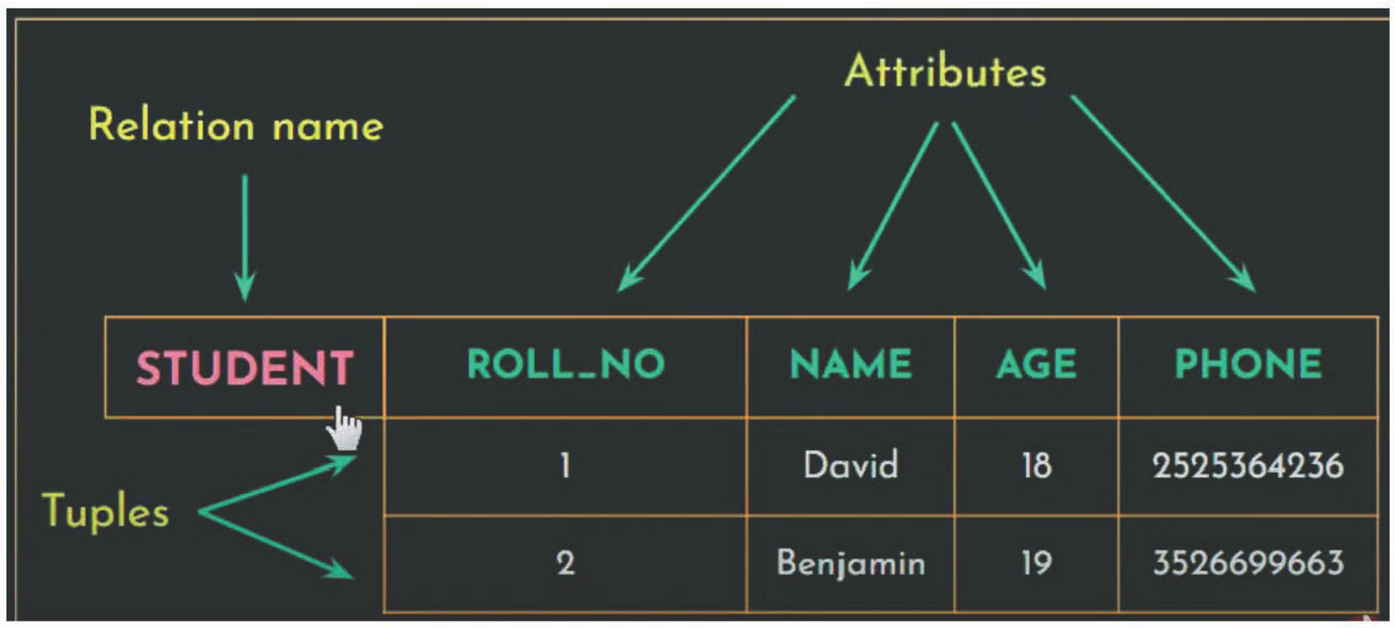
The Relational Data Model and Relational Database Constraints

Relational Model Concepts

The relational model represents the database as a collection of ***relations/tables***

Each relation resembles a table of values or, to some extent, a *flat* file of records. It is called a **flat file** because each record has a simple linear or *flat* structure

In the formal relational model terminology,
a row is called a *tuple*
a column header is called an *attribute,*
and the table is called a *relation.*
The data type describing the types of values that can appear in each column is represented by a *domain* of possible values.



Domains, Attributes, Tuples, and Relations

A **domain** *D* is a set of atomic values.
By **atomic** we mean that each value in the domain is indivisible further

A common method of specifying a domain is to specify a data type and set of values allowed for an attrubute.and format

Ex: (a) Name : string of characters that represent name of persons.

(b) Employee_ages: Possible ages of employees of a company (values between 20 & 70 years old).

Domain of Employee_ages : 1. Should be a positive integer (data-type)
2. Value must be inbetween 20 abd 70 (set of values)

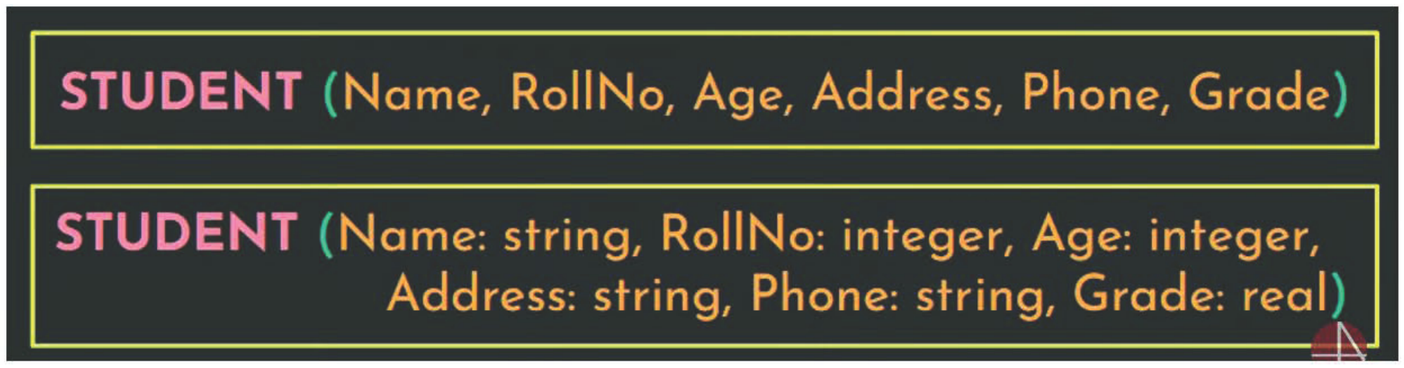
The above are called ***logical*** **definitions** of domains.

- Usa_phone_numbers. The set of ten-digit phone numbers valid in the United States.
- Local_phone_numbers. The set of seven-digit phone numbers valid within a particular area code in the United States. The use of local phone numbers is quickly becoming obsolete, being replaced by standard ten-digit numbers.
- Social_security_numbers. The set of valid nine-digit Social Security numbers. (This is a unique identifier assigned to each person in the United States for employment, tax, and benefits purposes.)
- Names: The set of character strings that represent names of persons.
- Grade_point_averages. Possible values of computed grade point averages; each must be a real (floating-point) number between 0 and 4.
- Employee_ages. Possible ages of employees in a company; each must be an integer value between 15 and 80.
- Academic_department_names. The set of academic department names in a university, such as Computer Science, Economics, and Physics.
- Academic_department_codes. The set of academic department codes, such as 'CS', 'ECON', and 'PHYS'.

Relation schema:

Describes a relation.
Made up of a relation name R and a list of attributes A1, A2, ...AN

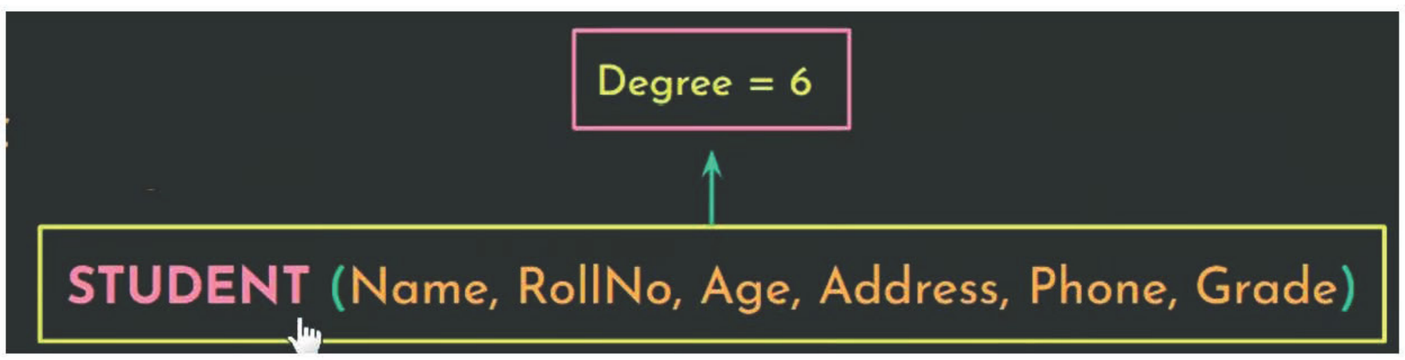
it is denoted by ***R*(*A*1, *A*2, … , *An*),**



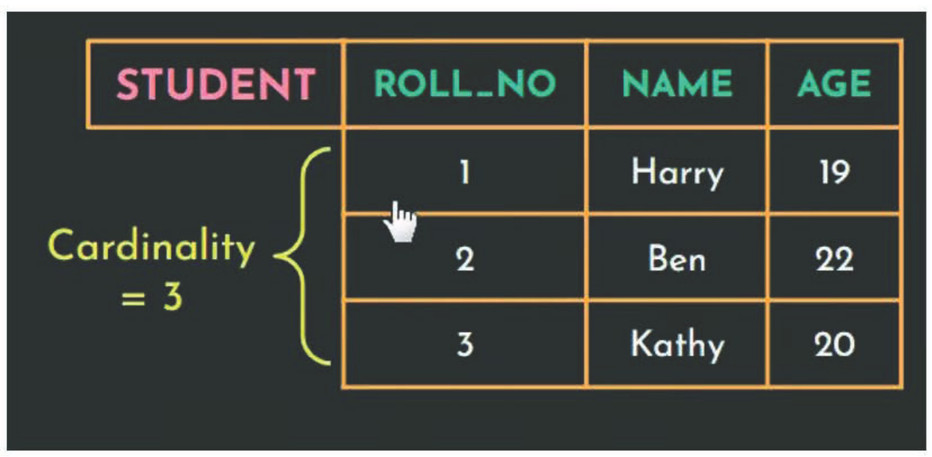
R is called the **name** of this relation
Each **attribute** *Ai* is the name of a role played by some domain *D* in the relation schema *R*.
D is called the **domain** of *Ai* and is denoted by **dom**(***Ai***).

More precisely, we can specify the following previously defined domains for some of the attributes of the STUDENT relation:
dom(Name) = Names;
dom(Ssn) = Social_security_numbers;
dom(HomePhone) =USA_phone_numbers,
dom(Office_phone) = USA_phone_numbers
dom(Gpa) =Grade_point_averages

Degree (or arity) of a relation: No of attributes in a relation schema



Cardinality: Total number of tuples present in a relation.



Relation state (or relation instance): Set of tuples at a given time
For a relation schema *R*(*A*1, *A*2, … , *An*) is denoted by *r*(*R*), is a set of *n*-tuples *r* = {*t*1, *t*2, … , *tm*}

The earlier definition of a relation can be *restated* more formally using set theoryconcepts as follows.
A relation (or relation state) *r*(*R*) is a **mathematical relation** of degree *n* on the domains dom(*A*1), dom(*A*2), … , dom(*An*), which is a **subset** of the **Cartesian product** (denoted by ×) of the domains that define *R*:

$$r(R) \subseteq (\text{dom}(A1) \times \text{dom}(A2) \times \ldots \times (\text{dom}(An))$$

Characteristics of Relations

1. Ordering of Tuples in a Relation

A relation is defined as a *set* of tuples.
Mathematically, elements of a set have **no order** among them; hence, tuples in a relation do not have any particular order. In other words, a relation is not sensitive to the ordering of tuples.

However, in a file, records are physically stored on disk, so there always is an order among the records. This ordering indicates first, second, *i*th, and last records in the file.

STUDENT	ROLL_NO	NAME	AGE
	1	Harry	19
	2	Ben	22
	3	Kathy	20

Changing order of tuples doesnot change the meaning of the relation

STUDENT	ROLL_NO	NAME	AGE
	2	Ben	22
	1	Harry	19
	3	Kathy	20

2. Ordering of Values within a Tuple

n-tuple is an ordered list of n values, so the ordering of values in a tuple is important
In the above relation, the values to be inserted has to be in order **INSERT (roll_no, name, age)**

Alternative defination of a tuple : A tuple is defined as a set of (<attribute>,<value>) pair,
When you write a tuple in this form then ordering of attributes is not important.

t = <(RollNo, 2), (Name, Ben), (Age,22)>

3. Values and NULLs in the Tuples.

Each value in a tuple is an **atomic** value; that is, it is not divisible into components within the framework of the basic relational model.
Hence, composite and multivalued attributes are not allowed. This model is sometimes called the **flat relational model.**

multivalued attributes are represented by separate relations
composite attributes are represented only by their simple component attributes as given in relation
eg : address -> composite(state, city, pincode ....)

STUDENT	ROLL_NO	NAME	ADDRESS
	2	Ben	Bengaluru, Karnataka-560051

⇨

STUDENT	ROLL_NO	NAME	CITY	STATE	PINCODE
	2	Ben	Bengaluru	Karnataka	560051

NULL -> Used to represent the values of attributes that may be
1. unknown (attribute is applicable but the value is not kown)
2. not applicable (attribute is not applicable)

STUDENT	ROLL_NO	NAME	AGE	MOBILE
	1	Harry	19	null (NA)
	2	Ben	22	6523214523
	3	Kathy	null (uknown)	25253645[illegible]

4. Interpretation of a relation

The relation schema can be interpreted as a declaration or a type of **assertion**

STUDENT (Roll_No, Name, Age, Mobile)

This schema declares OR asserts each student entity has a Name, Age, Roll_no and Mobile

Each tuple can be interpreted as a fact
Eg: The above table asserts that there is a STUDENT whose name is Harry, Age is 19 and roll_no is 1

Some relations represent facts about *entities,* whereas other relations may represent facts about *relationships.*

Relational Model Constraints and Relational Database Schemas

There are generally many restrictions or **constraints** on the actual values in a database state

1. inherent model-based constraints or implicit constraints : Constraints that are inherent/already present in the data model
eg : Duplicate tuples/rows are not allowed in a relationship

2. Schema based or explicit constraints: Defined directly in the schemas of the data model.
eg : Age of the employee has to be between 25 and 60 years

3. Application based or semantic constraints: Cannot be directly defined in the schemas and are difficult to express and enforce within the data model. They must be expressed and enforced by the application programs.

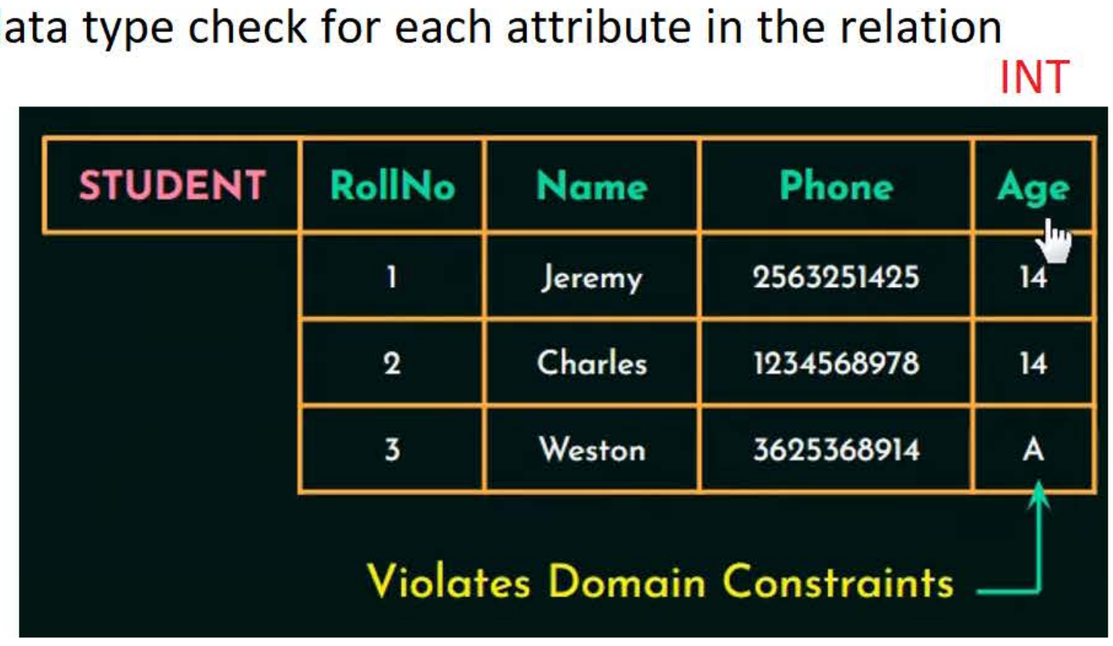
Another important category of constraints is *data dependencies*, which include *functional dependencies* and *multivalued dependencies*. They are used mainly for testing the "goodness" of the design of a relational database and are utilized in a process called *normalization*.

'Schema-based Constraints

1. Domain Constraints:

Domain constraints specify that within each tuple, the value of each attribute *A* must be an atomic value from the domain dom(*A*).

Performs data type check for each attribute in the relation



2. Key Constraints:

key : An attribute that can uniquely identify each tuple in a relation is called a key.
A *relation* is defined as a *set of tuples.* By definition, all elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means that no two tuples can have the same combination of values for *all* their attributes.

A **superkey(SK)** is a set of attributes that can help identify each tuple in a relation uniquely.
It is a **subsets of attributes** of a relation schema *R* with the property that no two tuples in any relation state *r* of *R* should have the same combination of values for these attributes.

Each relation has atleast one superkey by default ie set of all attributes

A key satisfies 2 constraints:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This *uniqueness* property also applies to a superkey.
2. It is a *minimal superkey*—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold.

A key is a superkey but not vice versa

Candidate key : A candidate key is a subset of the super key that can uniquely identify a tuple.
A table can have more than one candidate key.

3. Constraints on Null Values:

Specifies whether null values are permitted or not (NOT NULL).

4. Entity Integrity Constraints

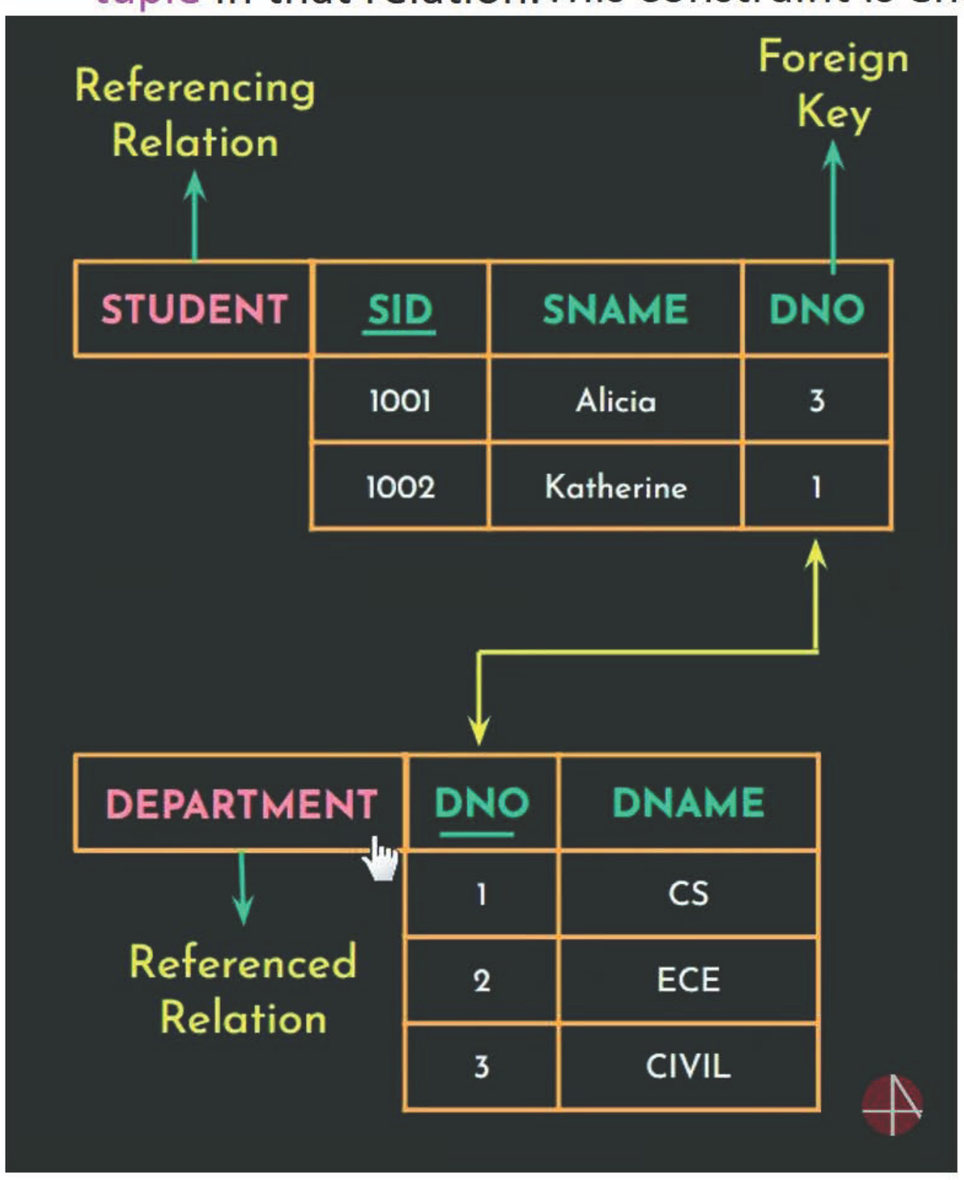
States that no primary key value can be null.

This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

5. Referential Integrity Constraint:

Specified between 2 relations and is used to maintain the consistency among tuples in the two relations

It states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.This constraint is enforced through foreign key



Foreign key must satisfy the following:

1. same domain.
2. Value of FK in a tuple either occurs as value of PK or is null

t1[FK] = t2[PK] --> tuple *t*1 **refers to** the tuple *t*2.

Relational Databases and Relational Database Schemas

A **relational database schema** *S* is a set of relation schemas *S* = {*R*1, *R*2, … , *Rm*} and a set of **integrity constraints** IC.

A **relational database state** DB of *S* is a set of relation states DB = {*r*1, *r*2, … , *rm*} such that each *ri* is a state of *Ri* and such that the *ri* relation states satisfy the integrity constraints specified in IC

OR

A **relational database schema** is a set of relation schemas and a set of integrity constraints.
A **relational database state** is a set of relation states such that every relational database state satisfies the integrity constraints specified on relational database schema.

When we refer to a relational database, we implicitly include both its schema and its current state. A database state that does not obey all the integrity constraints is called **not valid**, and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a **valid state.**

Update Operations and Dealing with Constraint Violations

The operations of the relational model can be categorized into **retrievals** and **updates**.

Update Operations

1. INSERT operation:

It is used to insert a new tuple in a relation

Insert can violate any of the four types of constraints

1. Domain constraint : It can be violated if an attribute value is given that does not appear in the corresponding domain or is not of the appropriate data type.
2. Key constraint : Key constraints can be violated if a key value in the new tuple *t* already exists in another tuple in the relation.
3. Entity integrity : Entity integrity can be violated if any part of the primary key of the new tuple is NULL.
4. Referential integrity : It can be violated if the value of any foreign key in *t* refers to a tuple that does not exist in the referenced relation.

If an insertion violates one or more constraints, the default option is to *reject the insertion.*

- *Operation*:
  Insert <'Cecilia', 'F', 'Kolonsky', NULL, '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4> into EMPLOYEE.
  Result: This insertion violates the entity integrity constraint (NULL for the primary key Ssn), so it is rejected.
  DBMS could ask the user to provide a value for Ssn and could then accept the insertion if a valid Ssn value is provided

- *Operation*:
  Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windswept, Katy, TX', F, 28000, '987654321', 7> into EMPLOYEE.
  Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEE because no corresponding referenced tuple exists in DEPARTMENT with Dnumber = 7.
  DBMS could either ask the user to change the value of Dno to some valid value (or set it to NULL), or it could ask the user to insert a DEPARTMENT tuple with Dnumber = 7 and could accept the original insertion only after such an operationwas accepted.

- *Operation*:
  Insert <'Alicia', 'J', 'Zelaya', '999887777', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, '987654321', 4> into EMPLOYEE.
  Result: This insertion violates the key constraint because another tuple with the same Ssn value already exists in the EMPLOYEE relation, and so it is rejected.

- *Operation*:
  Insert <'Cecilia', 'F', 'Kolonsky', '677678989', '1960-04-05', '6357 Windy Lane, Katy, TX', F, 28000, NULL, 4> into EMPLOYEE.
  Result: This insertion satisfies all constraints, so it is acceptable.

2. Delete Operation:

The **Delete** operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database.

EMPLOYEE	EmpID	Name	Age	Phone
	1330	Benjamin	29	6562356521
	1331	Charles	31	4523651852

PROJECT	PNo	ProjectName	DNo
	1	Global Payment	2
	2	Discover	1

WORKS_ON	EmpID	PNo	Hours
	1331	1	40
	1331	2	56
	1330	1	45

(i) Delete the EMPLOYEE tuple with EmpId = '1331'

This deletion is not acceptable, because there are tuples in WORKS_ON that refer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result

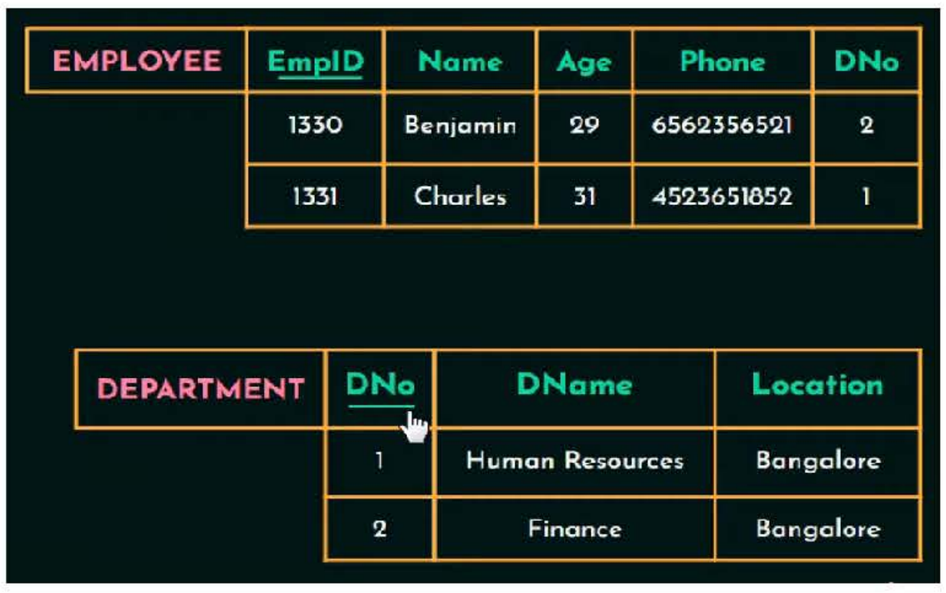
Action : Reject OR On delete cascade

(ii) Delete the WORKS_ON tuple with EmpID '1331' and P_No=1

This deletion is acceptable and deletes exactly one tuple.

3. Update Or Modify Operation

The **Update** (or **Modify**) operation is used to change the values of one or more attributes in a tuple (or tuples) of some relation *R*. It is necessary to specify a condition on the attributes of the relation to select the tuple (or tuples) to be modified.



EMPLOYEE	EmpID	Name	Age	Phone	DNo
	1330	Benjamin	29	6562356521	2
	1331	Charles	31	4523651852	1

DEPARTMENT	DNo	DName	Location
	1	Human Resources	Bangalore
	2	Finance	Bangalore

(i) Update the DNo of the EMPLOYEE tuple iwith EmpID = '1330' to 1

Updation is acceptable

(ii) Update the DNo of the EMPLOYEE tuple with EmpID = '1330 ' to 7

Unacceptable, because it violates referential integrity.

(iii) Update the EmpID of the EMPLOYEE tuple with EmpID= '1330' to '1331'

Unacceptable, because it violates primary key constraint by repeating a value that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of EmpID.

Updating an attribute that is *neither part of a primary key nor part of a foreign key* usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain.

Transaction Concept

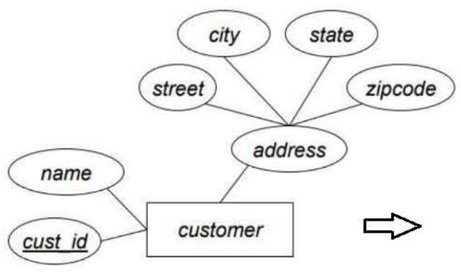
A **transaction** is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema

A single transaction may involve any number of retrieval operations and any number of update operations. These retrievals and updates will together form an atomic unit of work against the database.

Eg : a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

ER Model -> Relational Model

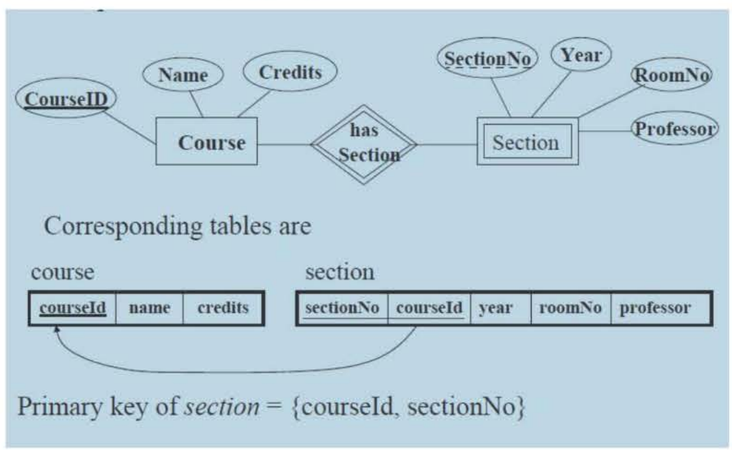
Step 1: Mapping of Regular Entity Types.



1. Attributes must be simple
2. Should have primary key

customer(*cust_id*, *name*, *street*, *city*, *state*, *zipcode*)

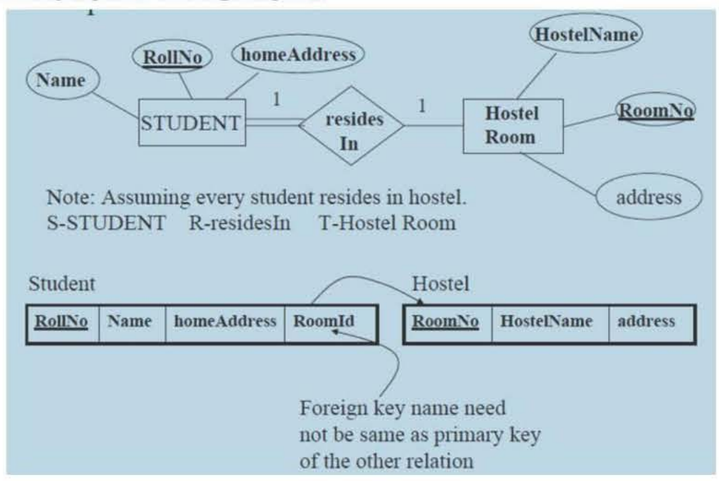
Step 2: Mapping of Weak Entity Types



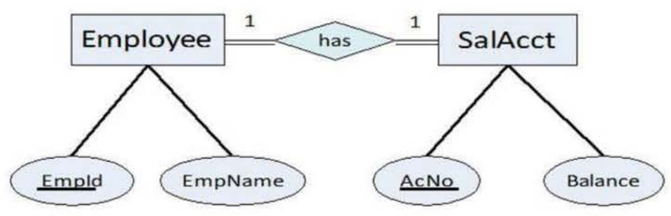
Create a foreign in weak entity refering to primary key of strong entity and make the partial key and foreign key as primary key in weak entity

Step 3: Mapping of 1:1 Relation Types

1. **Foreign Key approach : Choose one of the relations-say *S*, and include the primary key of *T* as foreign key in *S*.**

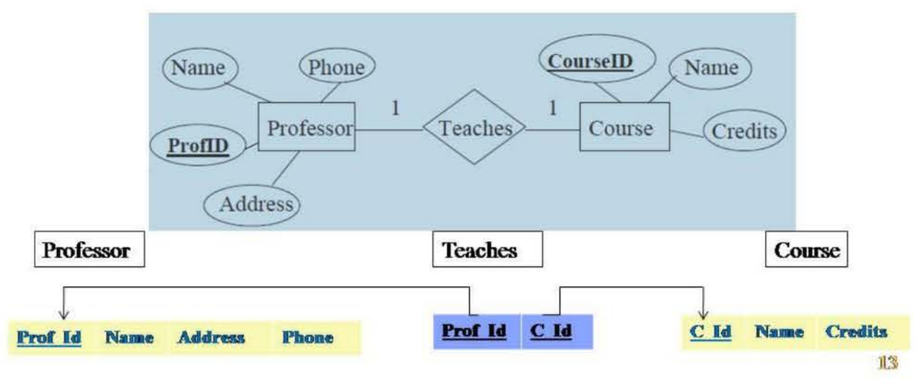


2. **Merged relation option : merging the two entity types** and the relationship into a single relation. This may be appropriate when both participations are total.

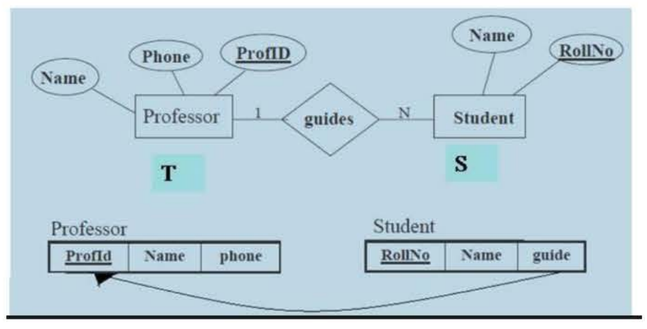


EmpId	EmpName	AcNo	Balance

3. **Cross-reference or relationship relation option:** Set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types. **(Partial on both)**

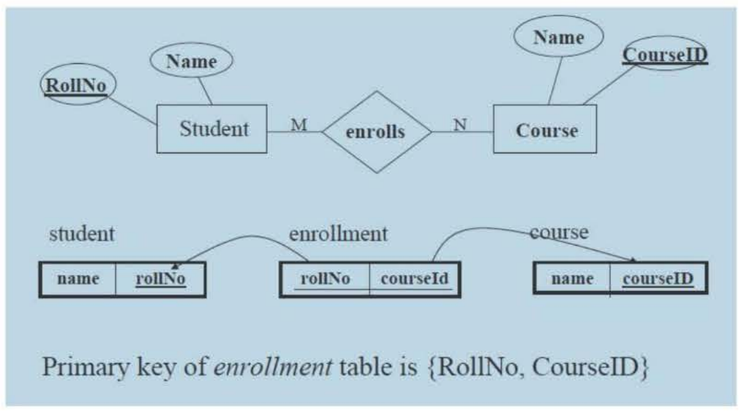


Step 4: Mapping of 1:N Relation Types



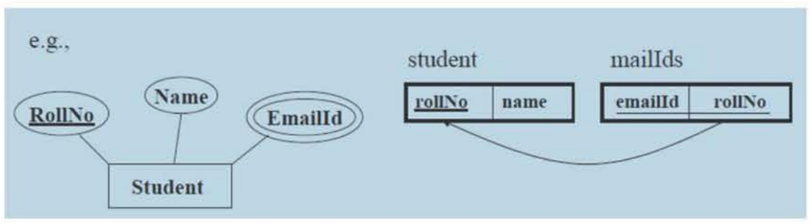
Include primary key of **1-side** as foreign key of **N-side**

Step 5: Mapping of M:N Relationship Types.



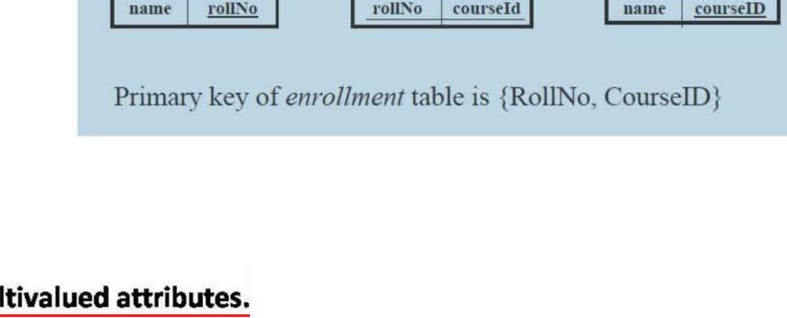
Make a new table for the relation and add 2 foreign key referencing the primary keys of the two ettities

Step 6: Mapping of Multivalued attributes.



Create a new relation for the multivalued attribute and add a foregin key referencing to primary key of entity

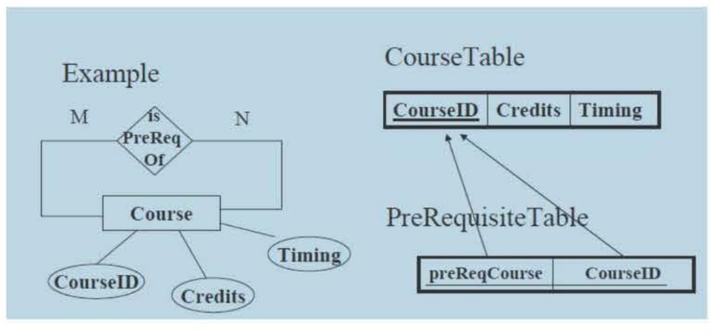
Step 7: Mapping of N-ary Relationship Types.



Create a relation for each entity with their attributes and primary keys
Create a new relation for the relationship and create foreign keys referencing to each of the related entity and the composite key of all these foreign keys is the primary key

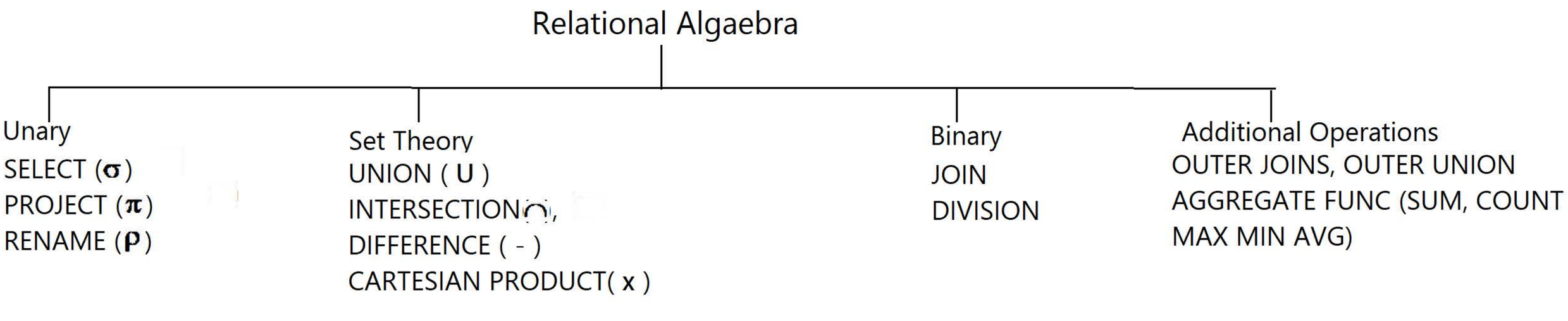
Handling Recursive relationships

- Make a table T for the participating entity set E and one table for recursive relationship R.



Relational Algaebra

Relational Operations: Basic set of operations for the relational model.
Sequence of relational algebra operations forms a relational algebra expression.



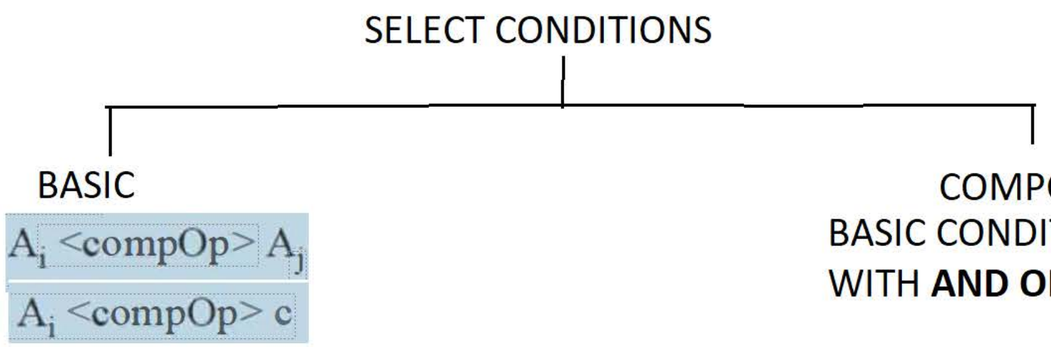
UNARY RELATIONAL OPERATIONS

1 SELECT

SELECT is used to select a subset of the tuples from a relation based on a selection condition (horizontal partition).

$$\sigma_{condition}\ (\text{Table name})$$

Select the EMPLOYEE tuples whose department number is 4: $\sigma_{DNO=4}(EMPLOYEE)$



$\sigma_{Dno=4\ AND\ Salary>25000}(EMPLOYEE)$ **Select * FROM EMPLOYEE WHERE Dno=4 AND Salary > 25000**

2. PROJECT

Keeps certain columns (attributes) from a relation and discards the other columns.
Creates a vertical partitioning

$$\pi_{<attribute\ list>}(R)$$

Example: To list each employee's first and last name and salary, the following is used:

$\pi_{FNAME, LNAME, SALARY}(EMPLOYEE)$: SELECT FNAME, LNAME, SALARY FROM EMPLOYEE

3. Sequences of Operations and the RENAME Operation

For most queries, need to apply several relational algebra operations one after the other. Either write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations.

EG : retrieve the first name, last name, and salary of all employees who work in department number 5.

$$\pi_{Fname, Lname, Salary}(\sigma_{Dno=5}(EMPLOYEE))$$ **--> Inline expression**

OR

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Fname, Lname, Salary}(DEP5\_EMPS)$$

- If we write:
  - RESULT (First_Name, Middle_Name, Salary)← $\pi_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)
  - The 3 attributes of DEP5_EMPS are *renamed*

RELATIONAL ALGAEBRA OPERATIONS FROM SET THEORY

1. UNION

Relations Rand S, are UNION COMPATIBLE, only if:

- Both Rand S should have same number of attributes.
- Domain of the attribute of R and domain of attribute of S, should be same and this is applicable to all the attributes

R U S returns relation instance containing all tuples that occur in either relation instance R or S, or both.

Graduates (R)

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38

Managers (S)

Number	Surname	Age
9297	O'Malley	56
7432	O'Malley	39
9824	Darkes	38

(R U S)

Graduates ∪ Managers

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38
9297	O'Malley	56

2. Intersection

R n S : returns a relation instance containing all tuples that occur in both Rand S.

Graduates (R)

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38

Managers (S)

Number	Surname	Age
9297	O'Malley	56
7432	O'Malley	39
9824	Darkes	38

(R ∩ S)

Graduates ∩ Managers

Number	Surname	Age
7432	O'Malley	39
9824	Darkes	38

3. SET DIFFERENCE

R- S: returns a relation instance containing all tuples that occur in R but not in S.

Graduates (R)

Number	Surname	Age
7274	Robinson	37
7432	O'Malley	39
9824	Darkes	38

Managers (S)

Number	Surname	Age
9297	O'Malley	56
7432	O'Malley	39
9824	Darkes	38

Graduates - Managers (R–S)

Number	Surname	Age
7274	Robinson	37

ENTITIES MUST BE UNION COMPATABLE TO CARRY OUT UNION, INTERSECTION AND DIFFERENCE OPERATIONS

4. CARTESIAN PRODUCT {CROSS PRODUCT or CROSS JOIN) X

Combining every member {tuple) from one relation {set) with every member (tuple) from the other relation (set)

X meaningless if not followed by a selection.

profDetail

eId	pname	deptNo
CS01	GIRIDHAR	1
CS02	KESHAV MURTHY	1
ES01	RAJIV GUPTHA	2
ME01	TAHIR NAYYAR	3

deptDetail

dId	dname
1	Computer Science
2	Electrical Engg.
3	Mechanical Engg.

profDept ← profDetail × deptDetail

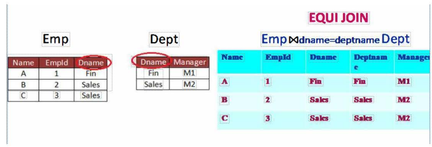
dId	dname	eId	Pname	deptNo
1	Computer Science	CS01	GIRIDHAR	1
1	Computer Science	CS02	KESHAV MURTHY	1
1	Computer Science	ES01	RAJIV GUPTHA	2
1	Computer Science	ME01	TAHIR NAYYAR	3
2	Electrical Engg	CS01	GIRIDHAR	1
2	Electrical Engg	CS02	KESHAV MURTHY	1
2	Electrical Engg	ES01	RAJIV GUPTHA	2
2	Electrical Engg	ME01	TAHIR NAYYAR	3
3	Mechanical Engg.	CS01	GIRIDHAR	1
3	Mechanical Engg.	CS02	KESHAV MURTHY	1
3	Mechanical Engg.	ES01	RAJIV GUPTHA	2
3	Mechanical Engg.	ME01	TAHIR NAYYAR	3

Binary Relational Operations : Joins

Join = Cross product + selection based on some condition
It is defined as cartisian product followed by selection
The **JOIN** operation, denoted by , is used to combine *related tuples* from two relations into single "longer" tuples.

1. Equi-join

SQL EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables.



Name	EmpId	Dname
A	1	Fin
B	2	Sales
C	3	Sales

Dname	Manager
Fin	M1
Sales	M2

Name	EmpId	Dname	Deptname	Manager
A	1	Fin	Fin	M1
B	2	Sales	Sales	M2
C	3	Sales	Sales	M2

```
SELECT *
FROM Emp, Dept
WHERE Emp.Dname = Dept.Dname
        OR
SELECT * FROM Emp
JOIN Dept
[ON (Emp.Dname = Dept.Dname)]
```

2.Natural join

A natural join is a type of join operation that creates an implicit join by combining tables based on columns with the same name and data type. Common columns are columns that have the same name in both tables.

- If joining attribute names are not same "Rename" and execute
- From the results duplicate attributes are removed.

A NATURAL JOIN can be an INNER join, a LEFT OUTER join, or a RIGHT OUTER join. The default is INNER join.

Student

Sno	Name	Sage
S1	N1	21
S2	N2	22
S3	N3	23

Course

Sno	Cname	Fees
S1	N1	1000
S3	N2	3000
S5	N3	5000

Sno	Name	Sage	Sno	Cname	Fees
S1	N1	21	S1	N1	1000
S1	N1	21	S3	N2	3000
S1	N1	21	S5	N3	5000
S2	N2	22	S1	N1	1000
S2	N2	22	S3	N2	3000
S2	N2	22	S5	N3	5000
S3	N3	23	S1	N1	1000
S3	N3	23	S3	N2	3000
S3	N3	23	S5	N3	5000

student x course

Sno	Name	Sage	Cname	Fees
S1	N1	21	N1	1000
S3	N3	23	N2	3000

Student * Course

```
SELECT *
FROM Student
NATURAL JOIN Course;
```

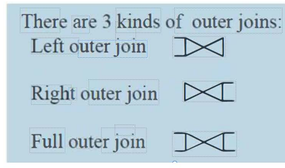
3. Cross Join

Join without any condition It is just a cross product of the two tables

```
SELECT col names
FROM table1
CROSS JOIN table2;
```

4. Outer Join

OUTER joins are used to keep all the tuples in R, or all those in S, or all those in both relations in the result of the join, regardless of whether or not they have matching tuples in other relation.



1. Left outer join

Keeps every tuple in first or left relation R, if no matching tuple is found in S, then attributes of S in the join result are filled or "padded1' with null values.

R

COL1	COL2
A	1
B	2
D	3
E	4
F	5

S

COLA	COLB
A	1
C	2
D	3
E	4

R ⟕ COL1=COLA S

COL1	COL2	COLA	COLB
A	1	A	1
B	2	NULL	NULL
D	3	D	3
E	4	E	4
F	5	NULL	NULL

2.Right outer join

Keeps every tuple in second or right relation S in the result,

R

COL1	COL2
A	1
B	2
D	3
E	4
F	5

S

COLA	COLB
A	1
C	2
D	3
E	4

R ⟖ COL1=COLA S

COL1	COL2	COLA	COLB
A	1	A	1
NULL	NULL	C	2
D	3	D	3
E	4	E	4

3. Full outer join

Keeps all tuples in both left and right relations when no matching tuples are found, padding them with null values as needed

R

COL1	COL2
A	1
B	2
D	3
E	4
F	5

S

COLA	COLB
A	1
C	2
D	3
E	4

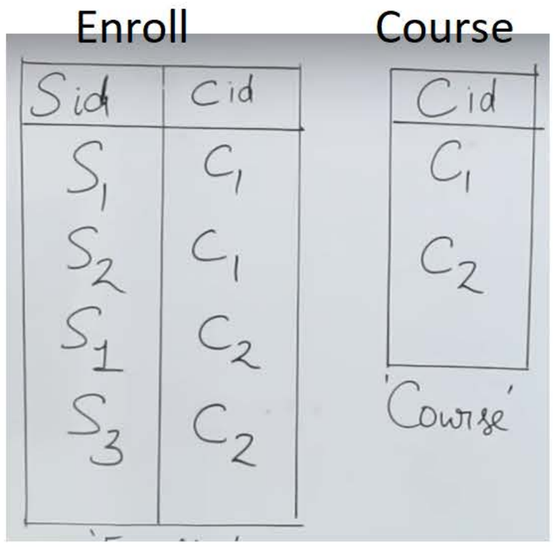
R ⟗ COL1=COLA S

COL1	COL2	COLA	COLB
A	1	A	1
B	2	NULL	NULL
D	3	D	3
E	4	E	4
F	5	NULL	NULL
NULL	NULL	C	2

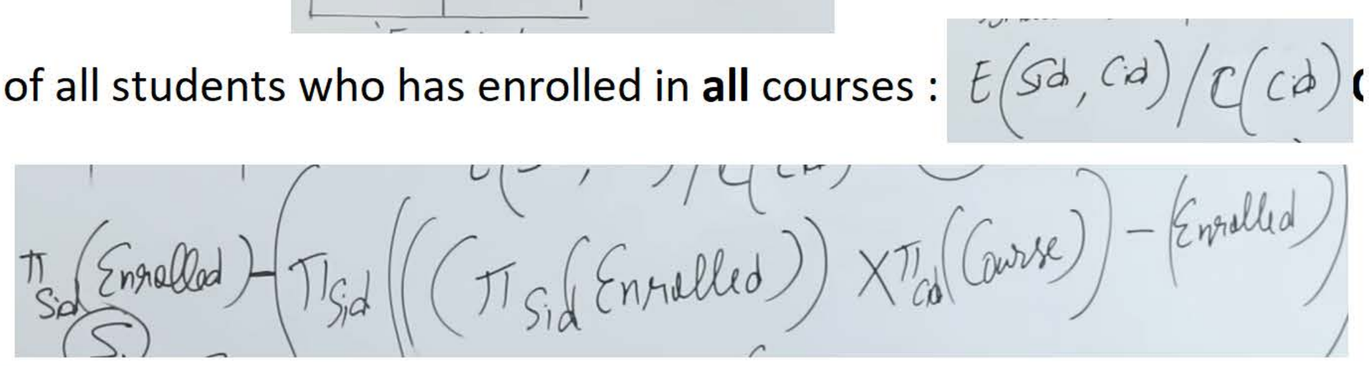
Operation(Symbols)	Purpose
Select(σ)	The SELECT operation is used for selecting a subset of the tuples according to a given selection condition
Projection(π)	The projection eliminates all attributes of the input relation but those mentioned in the projection list.
Union Operation(∪)	UNION is symbolized by symbol. It includes all tuples that are in tables A or in B.
Set Difference(-)	– Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B.
Intersection(∩)	Intersection defines a relation consisting of a set of all tuple that are in both A and B.
Cartesian Product(X)	Cartesian operation is helpful to merge columns from two relations.
Inner Join	Inner join, includes only those tuples that satisfy the matching criteria.
Theta Join(θ)	The general case of JOIN operation is called a Theta join. It is denoted by symbol θ.
EQUI Join	When a theta join uses only equivalence condition, it becomes a equi join.
Natural Join(⋈)	Natural join can only be performed if there is a common attribute (column) between the relations.
Outer Join	In an outer join, along with tuples that satisfy the matching criteria.
Left Outer Join(⟕)	In the left outer join, operation allows keeping all tuple in the left relation.
Right Outer join(⟖)	In the right outer join, operation allows keeping all tuple in the right relation.
Full Outer Join(⟗)	In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition.

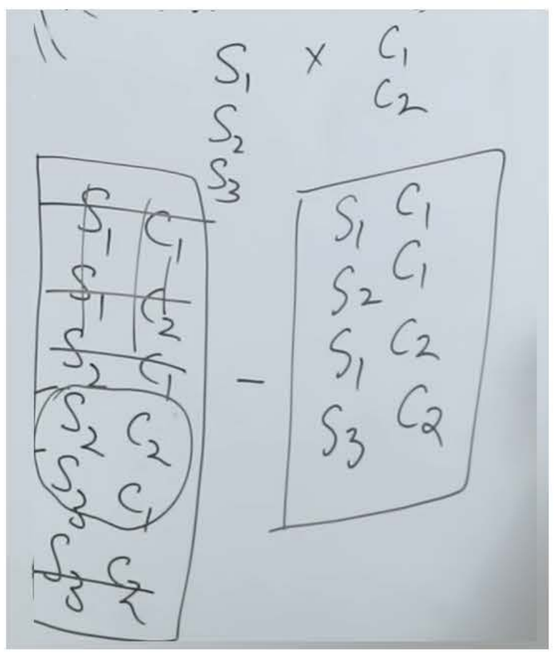
DIVISION OPERATION

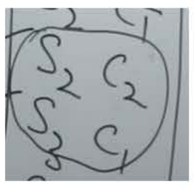
The division operator is used for queries which involve the 'all'.
R1 ÷ R2 = tuples of R1 associated with all tuples of R2.

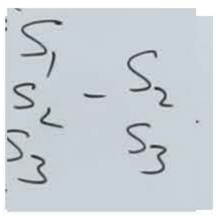


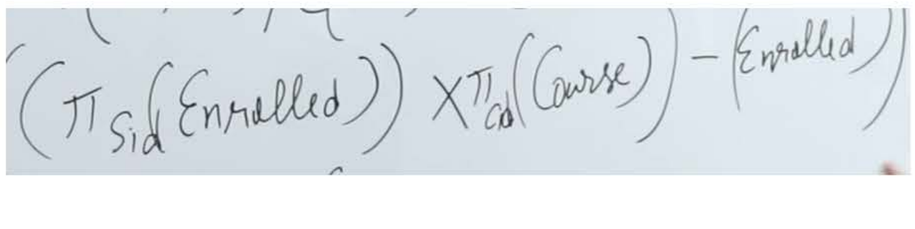
Reteieve SID of all students who has enrolled in **all** courses : $E(Sid, Cid) / C(Cid)$

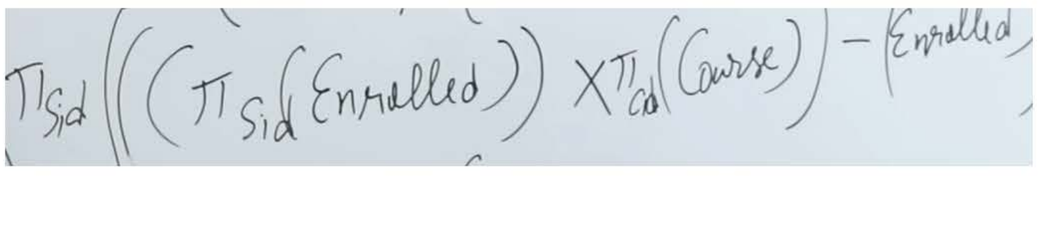












AGGREGATE FUNCTIONS

Most common aggregate functions:

sum	sums the values in the collection
avg	computes average of values in the collection
count	counts number of elements in the collection
min	returns minimum value in the collection
max	returns maximum value in the collection

Specified with symbol $\mathcal{F}$

$\mathcal{F}_{\text{<function\_list>}}(R)$

- Find the minimum Salary:

$\mathcal{F}_{\text{MIN (salary)}}(EMP)$

MIN(salary)
35000

Find the average Salary:

$\mathcal{F}_{\text{AVG (salary)}}(EMP)$

AVG(salary)
51000

Count the number of employees in the CS department:

$\mathcal{F}_{\text{COUNT (name)}}(\sigma_{\text{Dept = 'CS'}}(EMP))$

$\mathcal{F}_{\text{COUNT Ssn, AVERAGE Salary}}(EMPLOYEE)$.

Count just counts the number of rows, without removing duplicates $\mathcal{F}_{\text{COUNT-DISTINCT}}$
Can append -distinct to any aggregate function to specify elimination of duplicates

Using Grouping with Aggregation

$\text{<grouping attribute>}\,\mathcal{F}_{\text{<function\_list>}}(R)$

Ex: For each department, retrieve DNO, COUNT SSN, and AVERAGE SALARY

$\text{Dno}\,\mathcal{F}_{\text{COUNT Ssn, AVERAGE Salary}}(EMPLOYEE)$.

COMPONENTS OF SQL

Data Definition Language(DDL)
schema definition, relation definition, integrity constraints, views and schema modification.

Data Manipulation Language(DML)
inserting, updating

Embedded SQL and Dynamic SQL
SQL commands can be embedded in a high-level host language such as C, C++ or Java

Transaction Control
Specifies how transactions can be started / stopped1 how a set of concurrently executing transactions can be managed.

Authorization
Specifies how to restrict a user/ set of users to access only certain parts of data1 perform only certain types of queries etc.

Basic data types

Numeric data types
Integer numbers: INTEGER, INT, and SMALLINT
Floating-point numbers: FLOAT or REAL, and DOUBLE PRECISION
Formatted numbers:
: DECIMAL(*i, j*)—or *DEC(i, j)* or *NUMERIC (i, j)*
i -> total number of decimal digits
j -> number of digits after the decimal point

Character-string data types
Fixed length: CHAR(n), CHARACTER(n)
Varying length: VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n)
CHARACTER LARGE OBJECT or CLOB for large text values, such as documents. Maximum length in kilobytes (K), megabytes (M), or gigabytes (G).
CLOB(20M)

Boolean data type
DATE and TIME data type
TIMESTAMP, INTERVAL ETC.

Specifying Attribute Constraints and Attribute Defaults

NOT NULL

Default value
– **DEFAULT** <value>

CHECK clause
CHECK clauses at the end of CREATE TABLE statement
```
CHECK (Dnumber>0 AND Dnumber<21)
```

Specifying Key and Referential Integrity Constraints

PRIMARY KEY clause
Dnumber INT PRIMARY KEY;
PRIMARY KEY (A_1,A_2,….A_k)

UNIQUE clause
Specifies alternate (secondary) keys
Dname VARCHAR(15) UNIQUE;
UNIQUE (B_1,B_2,….B_k)

FOREIGN Key
FOREIGN KEY (A_1) REFERENCES r_2 (B_1)

actions to be taken on the referencing tuple
SET NULL | SET DEFAULT | CASCADE

Select student name and department name for DeptNo=4

```
Select STUDENT.name, DEPARTMENT.name
FROM STUDENT, DEPARTMENT
Where STUDENT.deptNo=DEPARTMENT.deptId and
STUDENT.deptNo=4

Select S.name, D.name
    FROM STUDENT AS S, DEPARTMENT AS D
    Where S.deptNo=D.deptId and S.deptNo=4
```
ALIASING…

Obtain the roll numbers of students who are currently (*current year and semester*) not enrolled for CS563 course.

```
(SELECT rollNo FROM enrollment
 WHERE sem= 5 and year = 2018 )
 EXCEPT
    (SELECT rollNo FROM enrollment
     WHERE courseId= 'CS563'and
     sem= 5 and year = 2018) ;
```
DIFFERENCE

Obtain roll numbers of students who are currently (*current year and semester*) enrolled for either CS563 or CS561 courses.

```
(SELECT rollNo FROM enrollment WHERE courseId=
'CS563' and sem= 5 and year = 2018)
 UNION
(SELECT rollNo FROM enrollment WHERE courseId=
'CS561' and sem= 5 and year = 2018) ;
```
UNION

Obtain the roll numbers of students who are currently (*current year and semester*) enrolled for both CS563 and CS561 Courses

```
(SELECT rollNo FROM enrollment WHERE
 courseId= 'CS563' and sem= 5 and year = 2018)
 INTERSECT
(SELECT rollNo FROM enrollment WHERE
  courseId= 'CS561' and sem= 5 and year = 2018 ) ;
```
INTERSECTION

LIKE comparison operator and Special characters
– Used for string **pattern matching**
– % any sequence of characters (0 or more)
– underscore (_) a single character

```
SELECT rollNo,name
FROM STUDENT
WHERE name like '%Kumar';

SELECT Name
FROM STUDENT
WHERE Year LIKE '_________12'
```
DD-MMM-YYYY)

BETWEEN comparison operator

```
SELECT name
FROM PROFESSOR
WHERE startYear between 2000 and 2010;
```

Ordering of Query Results

```
select name from professor
where startYear> 1980
order by name;

select name from professor
where startYear> 1980
order by name DESC;
```

Nested Queries,

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN
( SELECT      Pno, Hours
  FROM        WORKS_ON
  WHERE       Essn = '123456789' );
```

Correlated Query
When inner query of a nested relation refers to some attribute of outer query, then the queries are said to be correlated

```
SELECT last_name, salary, department_id
 FROM employees outer
 WHERE salary >
                (SELECT AVG(salary)
                 FROM employees
                 WHERE department_id =
                       outer.department_id group by department_id);
```

```
SELECT     product, Sum(price*quantity) AS TotalSales
FROM       Purchase
WHERE      date > '10/1/2005'
GROUP BY   product
```
GROUP BY

```
SELECT COUNT(customer_id), country
FROM Customers
GROUP BY country
HAVING COUNT(customer_id) > 1;
```
HAVING

ASSERTIONS AND TRIGGERS

Assertions and Triggers can be used to specify additional types of constraints that are outside the scope of the *built-in relational model constraints (primary and unique* keys, entity integrity, and referential integrity)

```
CREATE ASSERTION nomanager
CHECK( NOT EXISTS
   ( SELECT * FROM MANAGER
   WHERE  Department_id is NULL));
```

the above assertion ensures that there is no manager who is not assigned any department at any time.

	Assertion	Trigger
1.	We can use Assertions when we know that the given particular **condition is always true.**	We can use Triggers even particular condition may or **may not be true**.
2.	When the SQL condition is not met then there are chances to an entire table or even **Database to get locked up.**	Triggers can **catch errors** if the condition of the query is not true.
3.	Assertions are not **linked to specific table or event**. It performs task specified or defined by the user.	It helps in maintaining the **integrity constraints in the database tables,** especially when the primary key and foreign key constraint are not defined.
4.	Assertions do not maintain **any track of changes made in table.**	Triggers **maintain track** of all changes occurred in table.
5.	Assertions have **small syntax** compared to Triggers.	They have **large Syntax** to indicate each and every specific of the created trigger.
6.	**Execute after significant database updates**	**Always whenever a condition is true, trigger executes**

Triggers are mainly needed for the **following purpose**

⇒ Implementing and maintaining complex integrity constraints.
⇒ Recording the changes for auditing purposes.
⇒ Automatically passing signal to other programs that action needs to be taken whenever specific changes are made to a relation.

Three components
- **Event:** When this event happens, the trigger is activated
- **Condition (optional):** If the condition is true, the trigger executes, otherwise skipped
- **Action:** The actions performed by the trigger

Trigger: Events

- **Three event types**
  - Insert
  - Update
  - Delete
- **Two triggering times**
  - Before the event
  - After the event
- **Two granularities**
  - Execute for each row
  - Execute for each statement

If the trigger exists, then drop it first

```
Create or Replace Trigger EmpSal
Before Update On Employee
For Each Row
Begin
    If (new.salary > (old.salary * 1.1)) Then
        If (new.rank is null or  new.rank <> old.rank) Then
                RAISE_APPLICATION_ERROR(-20004, 'rank field not correct');
        End If;
    End If;
End;
/
```
Compare the old and new salaries
Make sure to have the "/" to run the command

```
ALTER TRIGGER Setnull_phone DISABLE;

ALTER TRIGGER Setnull_phone ENABLE;

DROP TRIGGER Setnull_phone;
```

Views (Virtual Tables) in SQL

Virtual relation, whose contents are derived from existing relation and it does not exist in physical form.

Contents of views are determined by executing a query based on any relation and it doesn't form part of database schema.

```
CREATE VIEW <view_name>
AS <subquery>;
```

```
CREATE VIEW BOOK_1
AS SELECT *
   FROM BOOK
   WHERE Category IN ('Textbook', 'Language Book');
```

queries are performed on these views

```
SELECT *
FROM BOOK_1;
```

```
SELECT *
FROM BOOK_2
WHERE B_Price > 30;
```

```
SELECT B_Title, B_Category
FROM BOOK_2
WHERE B_Price BETWEEN 30 AND 50;
```

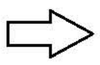
Views can be based on more than one relation

BOOK

ISBN	Book_title	Category	Price	Copyright_date	Year	Page_count	P_ID

PUBLISHER

P_ID	Pname	Address	State	Phone	Email_ID



```
CREATE VIEW BOOK_3
AS SELECT Book_title, Category, Price, BOOK.P_ID, Pname, State
   FROM BOOK, PUBLISHER
   WHERE BOOK.P_ID = PUBLISHER.P_ID;
```

Views based on more than one relation are **complex views.** They are time **consuming to execute, especially if multiple queries** are involved in view definition.

View Implementation, View Update, and Inline Views

Avoid problem by having "view materialization".

Physically creating view table when the view is first queried or created.

Incremental update (immediate, lazy when needed, periodic) have been developed for updating views when base tables are updated.

View exists physically as long as it is queried.

If not queried for a certain period of time, **it is automatically removed.**

Restrictions on Updating Views

Updates on views defined on **joining of more than one** table are not allowed.

Updates on views defined with **'group by'** clause and **aggregate functions** is not permitted, as a tuple in view will not have a corresponding tuple in base relation.

Updates on views which do not include **Primary Key** of base table, are also not permitted.

Allowed Updates on Views

- Defined on single base table
- Not defined using 'group by' clause and aggregate functions
- Include Primary Key of base table

CLUSTERED VS NON-CLUSTERED INDEX

What is an Index?

An Index is a key built from one or more columns in the database that speeds up fetching rows from the table or view. This key helps a Database like Oracle, SQL Server, MySQL, etc. to find the row associated with key values quickly.

Two types of Indexes are:

- Clustered Index
- Non-Clustered Index

What is a Clustered index?

Cluster index is a type of index which sorts the data rows in the table on their key values. In the Database, there is only one clustered index per table.

A clustered index defines the order in which data is stored in the table which can be sorted in only one way. So, there can be an only a single clustered index for every table. In an RDBMS, usually, the primary key allows you to create a clustered index based on that specific column.

What is Non-clustered index?

A Non-clustered index stores the data at one location and indices at another location. The index contains pointers to the location of that data. A single table can have many non-clustered indexes as an index in the non-clustered index is stored in different places.

For example, a book can have more than one index, one at the beginning which displays the contents of a book unit wise while the second index shows the index of terms in alphabetical order.
A non-clustering index is defined in the non-ordering field of the table. This type of indexing method helps you to improve the performance of queries that use keys which are not assigned as a primary key. A non-clustered index allows you to add a unique key for a table.

Characteristic of Clustered Index

- Default and sorted data storage
- Use just one or more than one columns for an index
- Helps you to store Data and index together
- Fragmentation
- Operations
- Clustered index scan and index seek
- Key Lookup

Characteristics of Non-clustered Indexes

- Store key values only
- Pointers to Heap/Clustered Index rows
- Allows Secondary data access
- Bridge to the data
- Operations of Index Scan and Index Seek
- You can create a nonclustered index for a table or view
- Every index row in the nonclustered index stores the nonclustered key value and a row locator

Parameters	Clustered	Non-clustered
Use for	You can sort the records and store clustered index physically in memory as per the order.	A non-clustered index helps you to creates a logical order for data rows and uses pointers for physical data files.
Storing method	Allows you to stores data pages in the leaf nodes of the index.	This indexing method never stores data pages in the leaf nodes of the index.
Size	The size of the clustered index is quite large.	The size of the non-clustered index is small compared to the clustered index.
Data accessing	Faster	Slower compared to the clustered index
Additional disk space	Not Required	Required to store the index separately
Type of key	By Default Primary Keys Of The Table is a Clustered Index.	It can be used with unique constraint on the table which acts as a composite key.
Main feature	A clustered index can improve the performance of data retrieval.	It should be created on columns which are used in joins.

Difference between unique and primary key

Parameter	PRIMARY KEY	UNIQUE KEY
Basic	Used to serve as a unique identifier for each row in a table.	Uniquely determines a row which isn't primary key.
NULL value acceptance	Cannot accept NULL values.	Can accepts NULL values.
Number of keys that can be defined in the table	Only one primary key	More than one unique key
Index	Creates clustered index	Creates non-clustered index
Auto Increment	A Primary key supports auto increment value.	A unique key does not supports auto increment value.
Modification	We cannot change or delete values stored in primary keys.	We can change unique key values.

SQL Injection

SQL injection is a code injection technique that might destroy your database.
SQL injection is one of the most common web hacking techniques.
SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL in Web Pages

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string. The variable is fetched from user input (getRequestString):

Example

```
txtUserId = getRequestString("UserId");
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

SQL Injection Based on 1=1 is Always True

Look at the example above again. The original purpose of the code was to create an SQL statement to select a user, with a given user id.
If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

UserId: 105 OR 1=1

Then, the SQL statement will look like this:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.
The SQL statement above is much the same as this:

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

A hacker might get access to all the user names and passwords in a database, by simply inserting 105 OR 1=1 into the input field.

SQL Injection Based on ""="" is Always True

A hacker might get access to user names and passwords in a database by simply inserting " OR ""=" into the user name or password text box:

User Name:
" or ""="

Password:
" or ""="

The code at the server will create a valid SQL statement like this:

Result

```
SELECT * FROM Users WHERE Name ="" or ""="" AND Pass ="" or ""=""
```

The SQL above is valid and will return all rows from the "Users" table, since OR ""="" is always TRUE.

Difference between DELETE and TRUNCATE

DELETE is a DML(Data Manipulation Language) command and is used when we specify the row(tuple) that we want to remove or delete from the table or relation. The DELETE command can contain a WHERE clause. If the **WHERE** clause is used with the DELETE command then it removes or deletes only those rows(tuple) that satisfy the condition otherwise by default it removes all the tuples(rows) from the table. Remember that DELETE logs the row deletions.

Syntax: DELETE command

```
DELETE FROM TableName
WHERE condition;
```

TRUNCATE is a DDL(Data Definition Language) command and is used to delete all the rows or tuples from a table. Unlike the DELETE command, the TRUNCATE command does not contain a WHERE clause. In the TRUNCATE command, the transaction log for each deleted data page is not recorded. Unlike the DELETE command, the TRUNCATE command is fast. We cannot roll back the data after using the TRUNCATE command.

Syntax: TRUNCATE command

```
TRUNCATE TABLE  TableName;
```

MySQL | Grant / Revoke Privileges

Granting Privileges

We have already learned about how to create users in MySQL using MySQL | create user statement. But using the Create User Statement only creates a new user but does not grant any privileges to the user account. Therefore to grant privileges to a user account, the GRANT statement is used. **Syntax:**

```
GRANT privileges_names ON object TO user;
```

Parameters Used:

- **privileges_name**: These are the access rights or privileges granted to the user.
- **object:**It is the name of the database object to which permissions are being granted. In the case of granting privileges on a table, this would be the table name.
- **user:**It is the name of the user to whom the privileges would be granted.

Privileges: The privileges that can be granted to the users are listed below along with the

Privilege	Description
SELECT	*select statement on tables*
INSERT	*insert statement on the table*
DELETE	*delete statement on the table*
INDEX	*Create an index on an existing table*
CREATE	*Create table statements*
ALTER	*Ability to perform ALTER TABLE to change the table definition*
DROP	*Drop table statements*
ALL	*Grant all permissions except GRANT OPTION*
UPDATE	*Update statements on the table*
GRANT	*Allows to grant the privilege that*

1. **Granting SELECT Privilege to a User in a Table**: To grant Select Privilege to a table named "users" where User Name is Amit, the following GRANT statement should be executed.

```
GRANT SELECT ON Users TO'Amit'@'localhost;
```

1. **Granting more than one Privilege to a User in a Table**: To grant multiple Privileges to a user named "Amit" in a table "users", the following GRANT statement should be executed.

```
GRANT SELECT, INSERT, DELETE, UPDATE ON Users TO 'Amit'@'localhost;
```

1. **Granting All the Privilege to a User in a Table**: To Grant all the privileges to a user named "Amit" in a table "users", the following Grant statement should be executed.

```
GRANT ALL ON Users TO 'Amit'@'localhost;
```

1. **Granting a Privilege to all Users in a Table**: To Grant a specific privilege to all the users in a table "users", the following Grant statement should be executed.

```
GRANT SELECT  ON Users TO '*'@'localhost;
```

2. **Granting Privileges on Functions/Procedures**: While using functions and procedures, the Grant statement can be used to grant users the ability to execute the functions and procedures in MySQL. **Granting Execute Privilege**: Execute privilege gives the ability to execute a function or procedure. **Syntax:**

```
GRANT EXECUTE ON [ PROCEDURE | FUNCTION ] object TO user;
```

Checking the Privileges Granted to a User: To see the privileges granted to a user in a table, the SHOW GRANTS statement is used. To check the privileges granted to a user named "Amit" and host as "localhost", the following SHOW GRANTS statement will be executed:

```
SHOW GRANTS FOR  'Amit'@localhost';
```

Output:

```
GRANTS FOR  Amit@localhost
GRANT USAGE ON *.* TO `SUPER`@localhost`
```

Revoking Privileges from a Table

The Revoke statement is used to revoke some or all of the privileges which have been granted to a user in the past. **Syntax:**

```
REVOKE privileges ON object FROM user;
```

Parameters Used:

- **object:** It is the name of the database object from which permissions are being revoked. In the case of revoking privileges from a table, this would be the table name.
- **user:** It is the name of the user from whom the privileges are being revoked.

What is Cursor in SQL ?

Cursor is a Temporary Memory or Temporary Work Station. It is Allocated by Database Server at the Time of Performing DML(Data Manipulation Language) operations on Table by User. Cursors are used to store Database Tables. There are 2 types of Cursors: Implicit Cursors, and Explicit Cursors. These are explained as following below.

1. **Implicit Cursors:**
   Implicit Cursors are also known as Default Cursors of SQL SERVER. These Cursors are allocated by SQL SERVER when the user performs DML operations.
2. **Explicit Cursors :**
   Explicit Cursors are Created by Users whenever the user requires them. Explicit Cursors are used for Fetching data from Table in Row-By-Row Manner.

SQL QUERIES

write an sql query to get the third maximum salary of an employee from employee table

Assuming you have an "Employee" table with columns "EmployeeID" and "Salary", you can use the following SQL query to get the third maximum salary:

```sql
SELECT Salary
FROM Employee
ORDER BY Salary DESC
LIMIT 1 OFFSET 2;
```

Explanation:

- The `ORDER BY Salary DESC` clause sorts the salaries in descending order.
- The `LIMIT 1 OFFSET 2` clause selects only the third row (salary) from the sorted result set. The `OFFSET 2` skips the first two rows (salaries), and the `LIMIT 1` selects the third row (salary).

How can you create an empty tabel from an existing table

You can create an empty table from an existing table in MySQL by using the `CREATE TABLE` statement with the `LIKE` keyword to copy the structure of the existing table. Here's an example:

```sql
CREATE TABLE new_table LIKE existing_table;
```

This statement creates a new table called `new_table` with the same structure as `existing_table`, but without any data.

You can also add additional table options to the `CREATE TABLE` statement, such as indexes, primary keys, and foreign keys, to further customize the new table.

Once the new table has been created, you can insert data into it using the `INSERT INTO` statement or by using other SQL commands to manipulate the data as needed.

How to fetch common records from two tables

To fetch common records from two tables, you can use the `INNER JOIN` clause in your SQL query. Here's an example:

Suppose you have two tables called "Table1" and "Table2" with a common column called "ID", and you want to fetch records that exist in both tables. You can use the following SQL query:

```sql
SELECT Table1.*, Table2.*
FROM Table1
INNER JOIN Table2
ON Table1.ID = Table2.ID;
```

How to fetch alternate records from a table

To fetch alternate records from a table, you can use the `ROW_NUMBER()` function along with a modulo operation in your SQL query to number the rows in the table, and then filter by even or odd row numbers. Here's an example:

Suppose you have a table called "Employee" with a column called "EmployeeID", and you want to fetch alternate rows from the table. You can use the following SQL query:

```vbnet
SELECT *
FROM (
  SELECT *, ROW_NUMBER() OVER (ORDER BY EmployeeID) AS rownum
  FROM Employee
) AS numbered_rows
WHERE numbered_rows.rownum % 2 = 0;
```

Explanation:

- The `ROW_NUMBER()` function assigns a sequential number to each row in the table, based on the order specified in the `ORDER BY` clause.
- The `AS rownum` alias gives a name to the row number column.
- The `WHERE` clause filters the result set to only include rows with even row numbers, using the modulo operator `%` to check for divisibility by 2.

Note that you can customize the ordering of the rows by changing the `ORDER BY` clause,

How to fetch unique records from a table

To fetch unique records from a table, you can use the `DISTINCT` keyword in your SQL query

```sql
SELECT DISTINCT CustomerID
FROM Orders;
```

What is the command used to fetch first five characters of the string

To fetch the first five characters of a string, you can use the `LEFT` function in your SQL query. Here's an example:

```sql
SELECT LEFT(ProductName, 5) AS ShortName
FROM Products;
```

Explanation:

- The `LEFT` function returns the specified number of characters from the beginning of a string.

RELATIONAL DATABASE DESIGN

BY SHIVKARAMAN

Implicit goals of design:

1. Information preservation: maintaining all concepts, including attribute types, entity types, and relationship types, which are described using a model such as ER model which are originally captured.

2. Minimizing redundancy: minimizing redundant storage of the same information and reducing need for multiple updates to maintain consistency across multiple copies of same information in response to real-world events that require making an update.

Informal Design Guidelines for Relational Databases

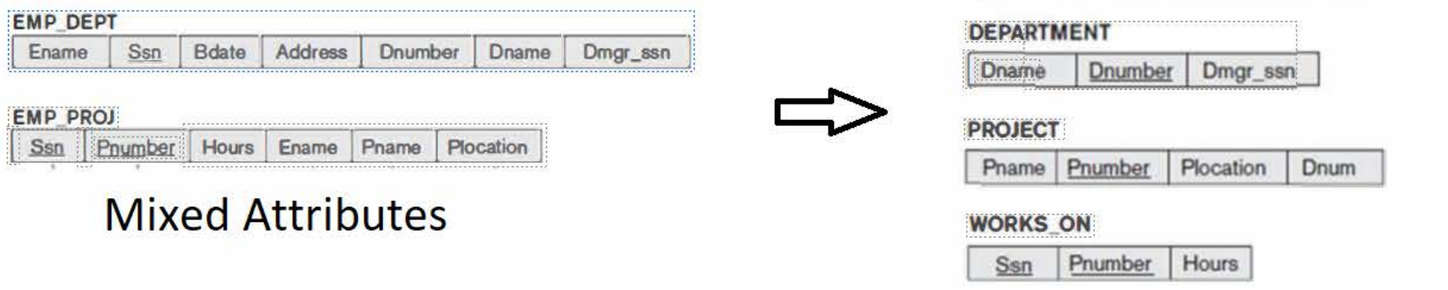
Four informal guidelines that may be used as measures to determine the quality of relation schema design.

1. Making sure that the semantics of the attributes is clear in the schema
2. Reducing the redundant information in tuples
3. Reducing the NULL values in tuples
4. Disallowing the possibility of generating spurious tuples

1. Imparting Clear Semantics to Attributes in Relations

Semantics of a relation refers to its meaning resulting from interpretation of attribute values in a tuple.
If ER and mapping procedure is followed systematically, relational schema will have clear semantics.
In general, the easier it is to explain the semantics of the relation, the better the relation schema design will be.

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



Mixed Attributes

2. Redundant Information in Tuples and Update Anomalies

Redundancy refers to storage of same data in more than one location.
One goal of schema design is to minimize the storage space => Reduce redundancy

In addition, due to redundancy, certain anomalies can arise

1. Insertion Anomaly

It leads to a situation in which certain information cannot be inserted in a relation unless some other information is stored

Eg : This table contains attributes two dirrenrent relatiosn ie empolyee and department.
So when inserting new row, we must include either the attribute values for the department that the employee works for, or NULL.

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	98765432

PROBLEM: If i want to create a new department to which employees are not yet assigned ie with no employees, it is not possible. To do this all emoloyee attriubutes has to be set NULL inorder to insert a new department with no employee as employee SSN is the primary key and it cannot be NULL.

2. Deletion anomaly:

It leads to situation in which deletion of data representing certain information results in losing data representing some other information that is associated with it

Eg: If we delete from EMP_DEPT an employee that happens to represent the last employee working for a particular department, information concerning that department is lost from the database.

3.Modification/Update Anomaly:

It leads to situation in which repeated data changes at one place results in inconsistency unless the same data is also changed at other places

Eg: In EMP _DEPT, if we change the value of one of the attributes of a particular department-say, the manager of department 5. We must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent.

Guideline: Design a schema that does not suffer from the insertion, deletion and update anomalies.
If there are any anomalies present, then note them clearly and make sure that the programs that update database will operate correctly.

3. NULL Values in tuples

Some schema designs we may group many attributes together into a "fat" relation.
If many of the attributes do not apply to all tuples in the relation, can end up with many NULLs in those tuples.

Problems with NULLs

- Wasted storage space
- Problems in understanding the meaning of the attributes
- How to account for them when aggregate operations such as COUNT or SUM are applied
- SELECT and JOIN operations involve comparisons; if NULL values are present, the results may become unpredictable

Different meanings of NULL

1. Attribute does not apply to this tuple
2. Attribute value for this tuple is unknown_
3. Value is known but absent;

Guideline As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL.

4. Generation of Spurious Tuples

A spurious tuple is, basically, a record in a database that gets created when two tables are joined badly.
In database, spurious tuples are created when two tables are joined on attributes that are neither primary keys nor foreign keys.

- **Example: Spurious Tuples**

R = (A, B, C); R decomposed into R1 and R2
R1 = (A, B); R2 = (B, C)

r

A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3

r1

A	B
a_1	b_1
a_2	b_2
a_3	b_3

r2

B	C
b_1	c_1
b_2	c_2
b_3	c_3

Lossy join

r1 * r2 — Original info is distorted

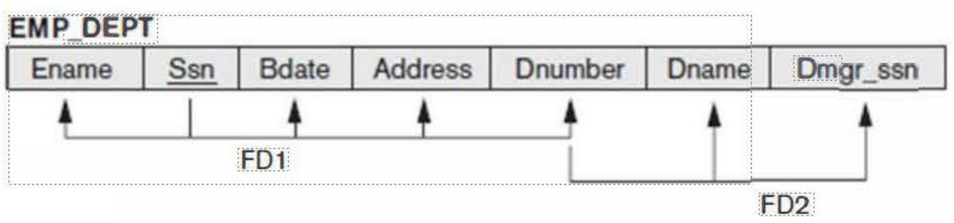
A	B	C	
a_1	b_1	c_1	
a_1	b_1	c_3	Spurious Tuples
a_2	b_2	c_2	
a_3	b_1	c_1	Spurious Tuples
a_3	b_3	c_3	

Guideline Design relation schemas to be joined with equality conditions on attributes that are appropriately related. This guarentees that no suprious attributes are created.
And Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations.

Functional Dependencies

A functional dependency is a constraint or relationship between two sets of attributes from the database.

A set of attributes, X, functionally determines another set, Y, or: there exists a functional dependency between X and Y.



Trivial FD :

FD : X -> Y

If y is subset of X then it is trivial FD

ie X n Y != NULL (intersection)

NON- TRIVIAL FD

X n Y = NULL

1. FD1: {Ename, Bdate, Address and Dnumber} are functionally dependent on SSN
2. FD2: {Dname, Dmgr_ssn} are functionally dependent on Dnumber

SSN →{Ename, Bdate, Address and Dnumber}

Dnumber →{Dname, Dmgr_ssn}

Normalization

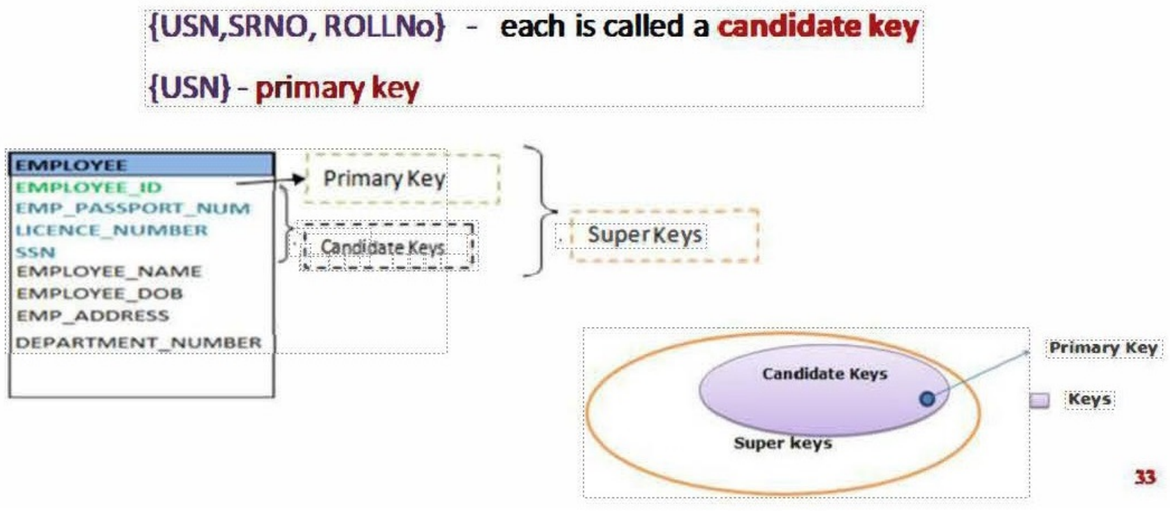
Normalization of data can be considered a process of analyzing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of minimizing redundancy and information preservation.

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies. Normalization rules divides larger tables into smaller tables and links them using relationships. The purpose of Normalisation in SQL is to eliminate redundant (repetitive) data and ensure data is stored logically.

- If a relation schema has more than one key, each is called a **candidate key**.
- One of the candidate keys is arbitrarily designated to be the **primary key**, and the others are called **secondary keys**.

{USN,SRNO, ROLLNo} - each is called a candidate key

{USN} - primary key



An attribute of relation schema R is called a **prime attribute** of R if it is a member of some candidate key of R.
An attribute is called **non-prime** if it is not a member of any candidate key

R (USN, Name, Srno, RollNo, Branch, Address)

{USN,Srno, RollNo} - candidate key

USN,Srno, RollNo are called Prime attributes

Name, Branch, Address are called non-Prime attributes

What is the Purpose of Database Normalisation?

Database normalisation is the process of transforming a database design into somehting that adheres to a common standard for databases.

Once this process is followed, which is a standard process in database design, the database is said to be “normalised”.

1. Prevent the Same Data from Being Stored in Many Places

Employee ID	Name	Department	Location
1	John Smith	Finance	Chicago
2	Mary Taylor	Customer Support	Boston
3	Rebecca Jones	Sales	Chicago
4	Tony Adams	Finance	North Chicago
5	Sarah Johnson	Sales	Portland

Department and the Location columns have redundant data.

Employee ID	Name	Department	Location
1	John Smith	2	1
2	Mary Taylor	1	2
3	Rebecca Jones	3	1
4	Tony Adams	2	3
5	Sarah Johnson	3	4

ID	Department Name
1	Customer Support
2	Finance
3	Sales

ID	Location Name
1	Chicago
2	Boston
3	North Chicago
4	Portland

2. Prevent Updates Made to Some Data and Not Others

Let’s then say that we want to update the name of one of the departments. We want to update “Finance” to say “Accounting”.

We would update the data so it looks like this:

Employee ID	Name	Department	Location
1	John Smith	Accounting	Chicago
2	Mary Taylor	Customer Support	Boston
3	Rebecca Jones	Sales	Chicago
4	Tony Adams	Accounting	North Chicago
5	Sarah Johnson	Sales	Portland

This means we need to update every value in the table.

But, if the data is normalised, like our earlier example, there would be one record in a “department” table that contains the value of “Finance”. We would change this single value to say “Accounting”. All employees that related to the department ID would then show the updated name in the systems that used this.

ID	Department Name
1	Customer Support
2	~~Finance~~ Accounting
3	Sales

The employee table is unaffected.

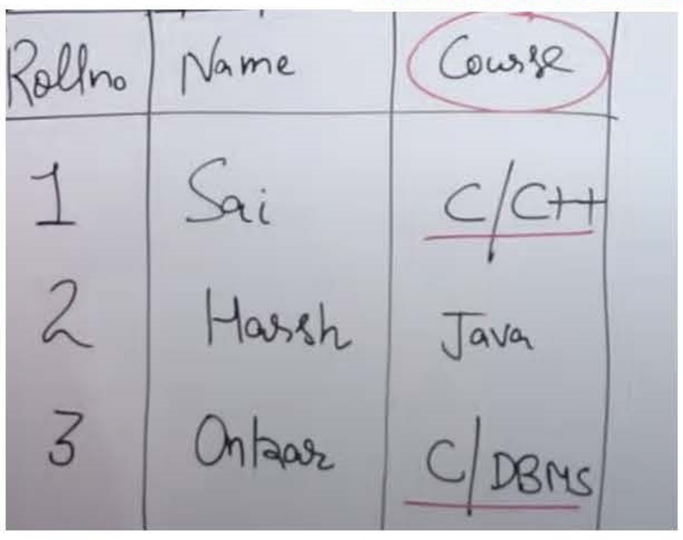
3. Insertion , Deletion and Update anomoly

4. Ensure Queries are More Efficient

First Normal Form

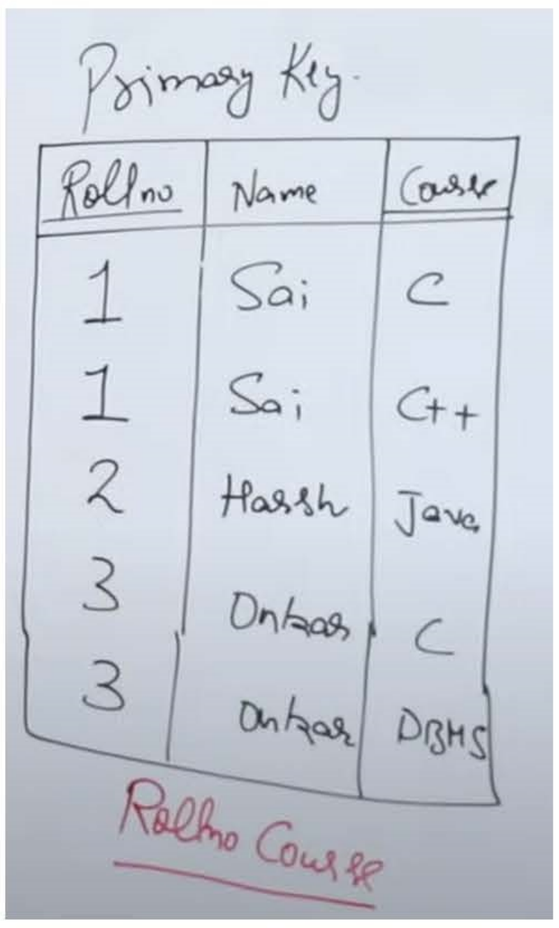
Definition : A relation schema R is in INF if and only if, all its underlying attributes contain atomic values only . ie table should not contain multivalued attributes

Multivalued attribute

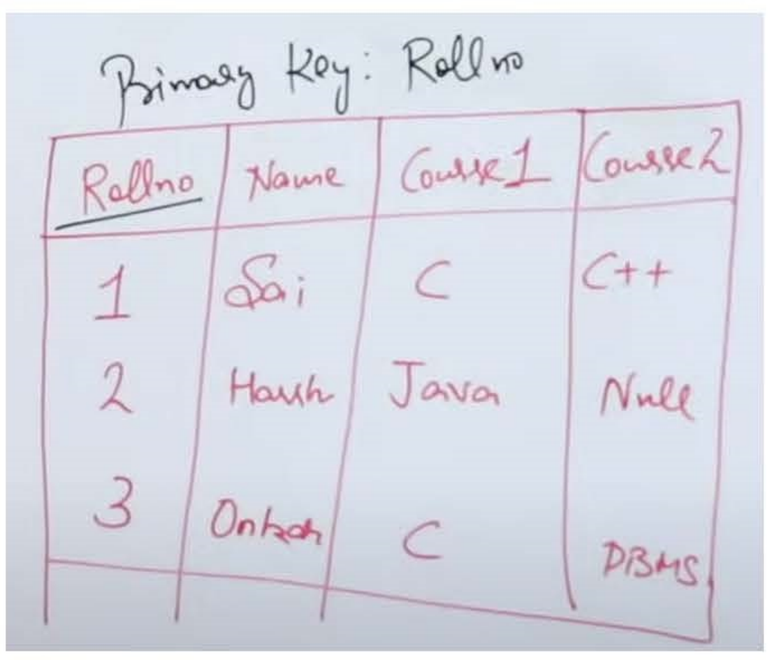


Not in 1NF

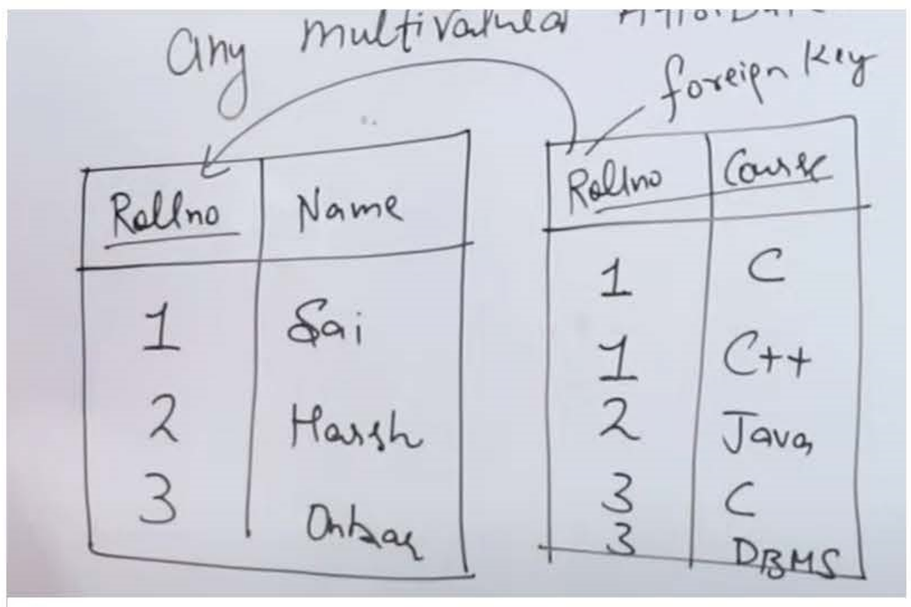
3 ways to bring this relation in 1NF



Problem : Redundancy



Problem : TOO MANY NULL VALUES
If a student has regestered for 10 courses and others have regestered for only 1 or 2, all those places should be set as NULL



Eg 2 : Compisite attribute with multivalued attribute

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

Composite attribute

(b)
EMP_PROJ

Multivalued Multivalued

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

Ssn is the primary key
Pnumber is the **partial** key of the nested relation;
that is, within each tuple, the nested relation must have unique values of Pnumber.

To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key* into it; the primary key of the new relation will combine the partial key with the primary key of the original relation.

(c)
EMP_PROJ1

Ssn	Ename

EMP_PROJ2 Multivalued Multivalued

Ssn	Pnumber	Hours

Why 1NF?

If a table is not in 1NF, it means that it contains repeating groups of data, which can lead to data redundancy and inconsistency. For example, consider a table that contains customer orders, where each row contains the customer's name, address, and a list of products they ordered. If a customer orders multiple products, their name and address will be repeated for each product, leading to data redundancy and potential inconsistencies.

To avoid these issues, the table should be normalized into separate tables, with one table containing customer information (including a unique customer ID as the primary key), and another table containing order information (including a foreign key that links each order to a specific customer). This ensures that each piece of data is stored only once and that changes to customer information can be made without affecting the order information, and vice versa.

Second Normal Form

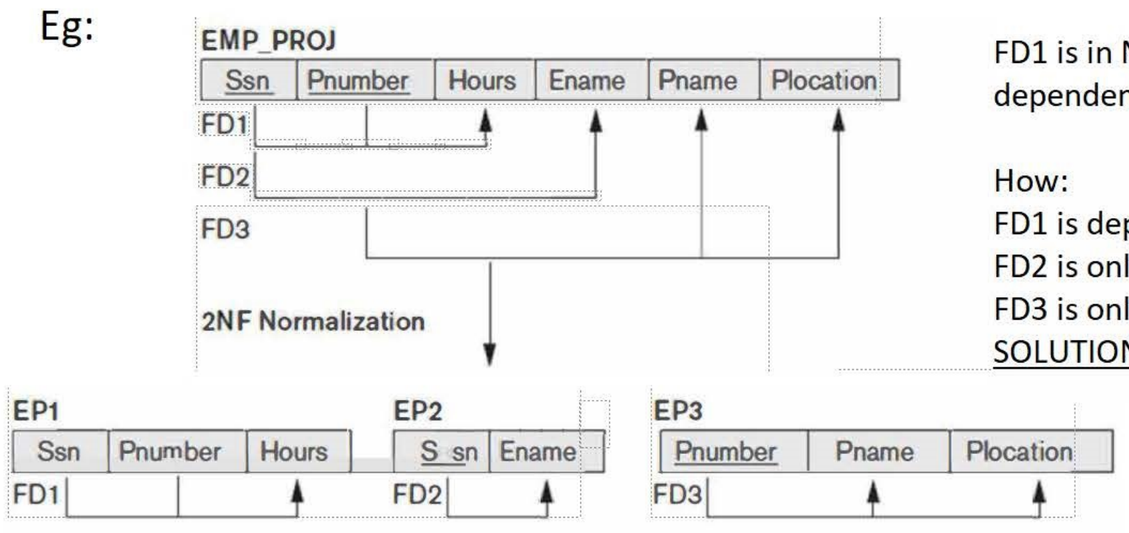
A relation schema R is in 2NF if every non-prime attribute is fully functionally dependent on every key of R

OR

A relation schema R is in 2NF if every non-prime attribute A in R is not partially dependent on any key of R.

- Second normal form (2NF) is based on the concept of full functional dependency and primary key

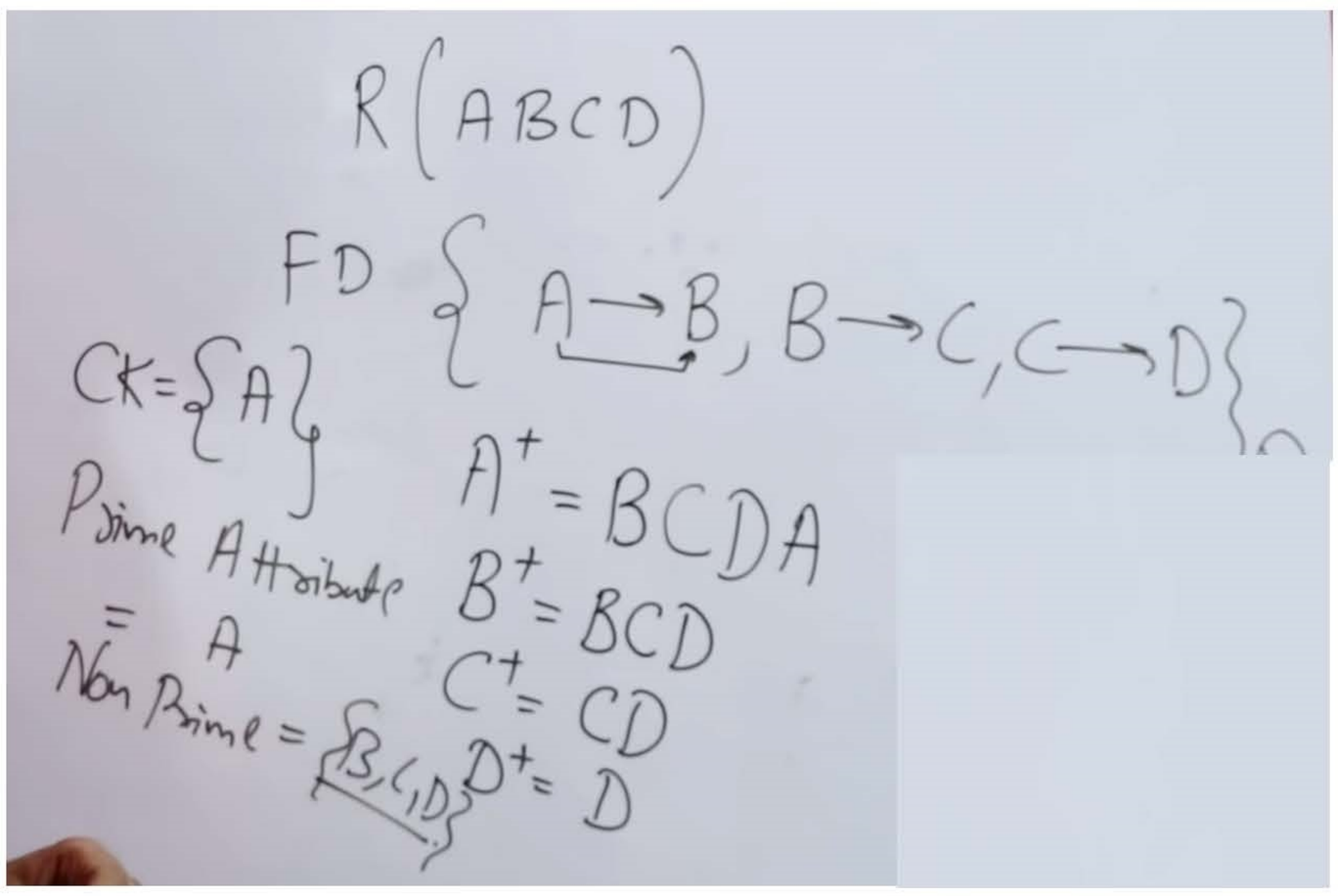
Eg:

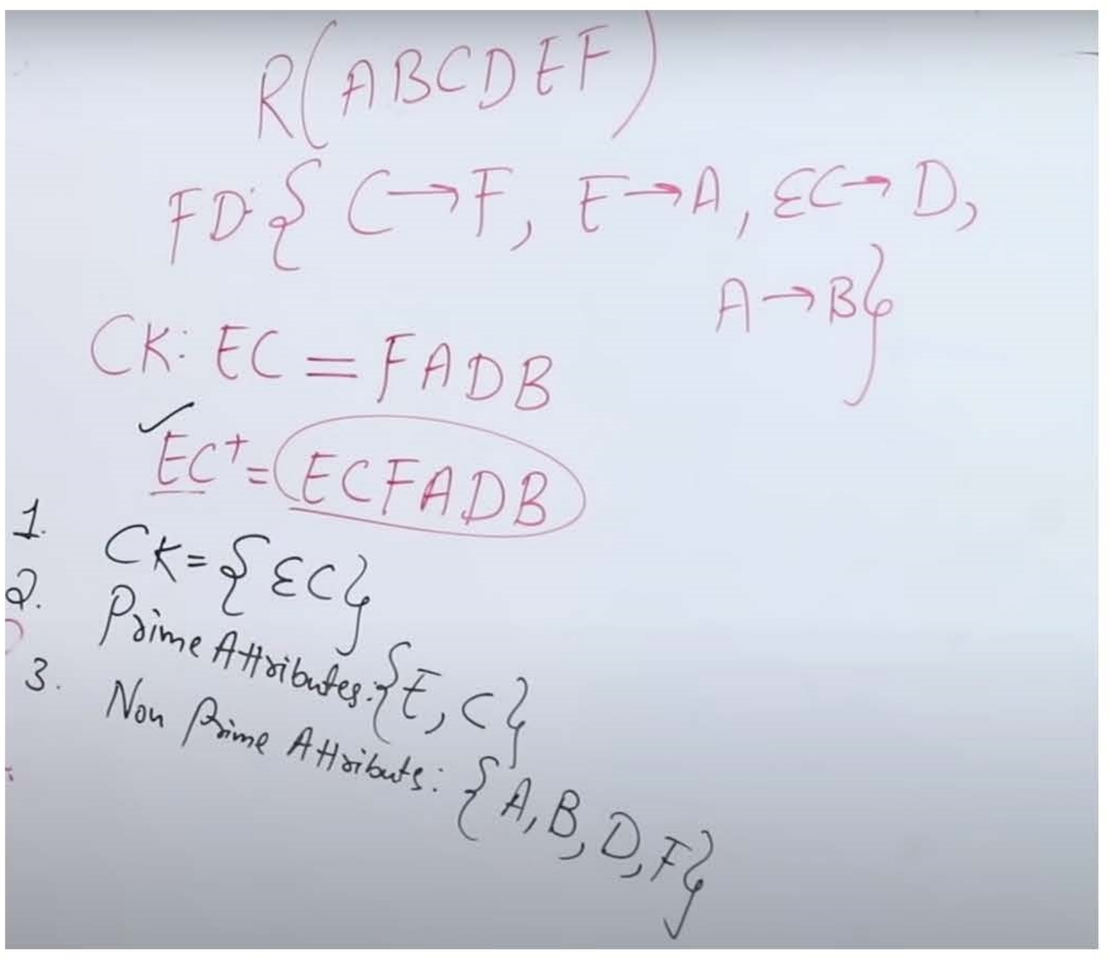


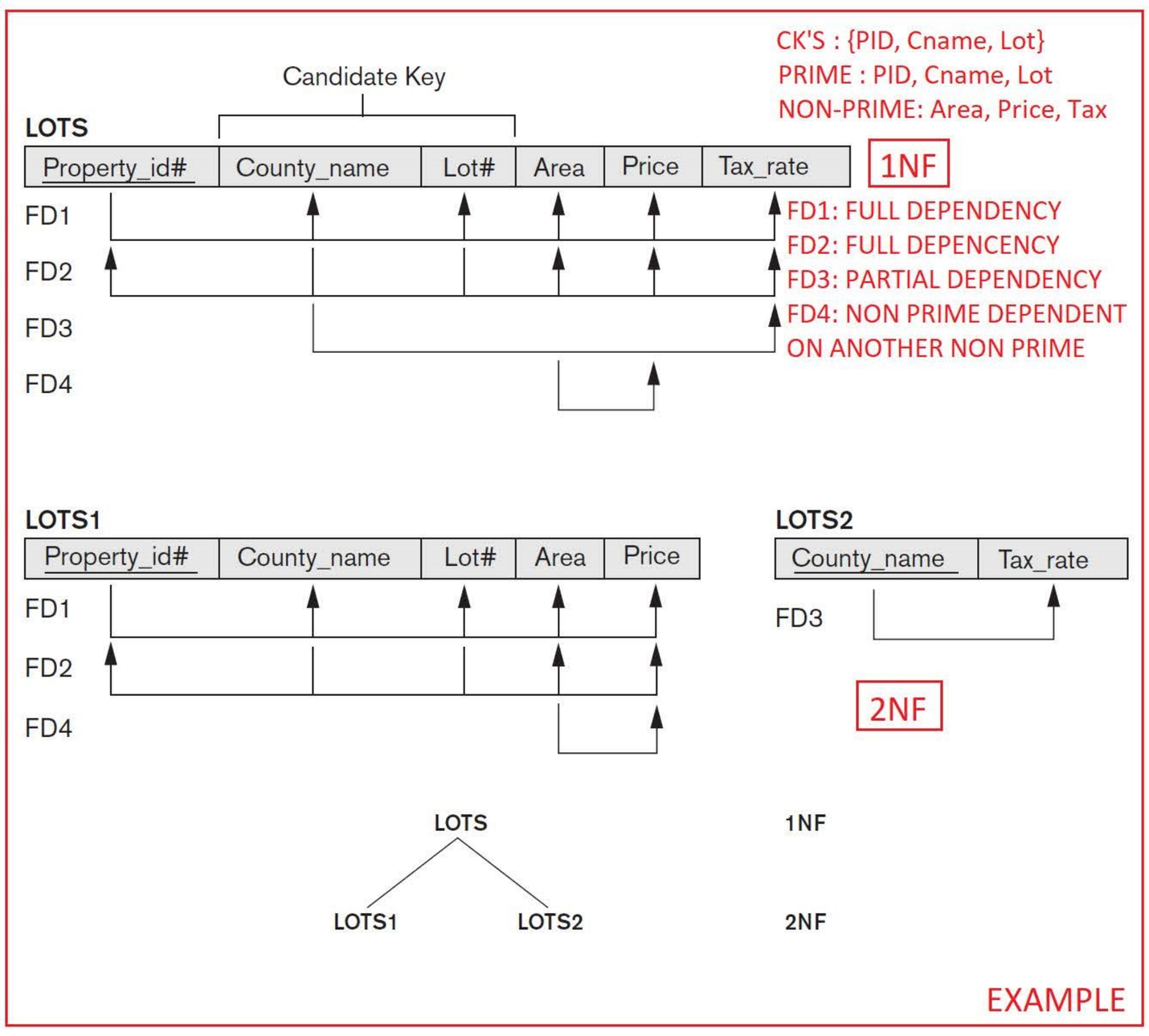
FD1 is in NF but FD2 and FD3 are not in 2NF as they are partially dependent on the candidate key

How:
FD1 is dependent on both SSN and Pnumber ir FULL DEPENDENCY
FD2 is only dependent on SSN -> PARTIAL DEPENCENCY
FD3 is only dependent on Pnumber -> PARTIAL DEPENCENCY
SOLUTION: DIVIDE INTO DIFF TABLES

Finding Closure of Functional dependency : Used to find candidate key, prime and non prime attributes







1. C->F
   1. Is C a subset of CK ie {EC} -> Yes
   2. Is F a non-prime attribute-> Yes
   3. F is dependent on C ie a non prime attribute is partially dependent on a part of canndidate key -> **PARTIAL DEPENCENCY**
2. E->A
   Same as C->F ie **PARTIAL DEPENCENCY**
3. EC->D
   EC = CK and D -> Non prime attribute
   FULL DEPENDENCY
4. A->B
   Both are non prime

NOT IN 2NF We need to resolve FD1 and FD2

Why 2NF?

The purpose of 2NF is to ensure that each table contains only data that is directly related to the primary key, and that all non-key attributes are dependent on the entire primary key. By doing so, we can avoid data redundancy and inconsistency, and ensure that our database is properly normalized.

If a table is not in 2NF, it means that it contains partial dependencies, where some non-key attributes depend on only part of the primary key. This can lead to data redundancy and inconsistency. For example, consider a table that contains customer orders, where each row contains the order ID, customer ID, customer name, and product name. In this case, the customer name depends only on the customer ID, which is only part of the primary key. If a customer changes their name, we would need to update multiple rows in the table, potentially leading to data inconsistency.

To avoid these issues, we can normalize the table into two separate tables, with one table containing customer information (including a unique customer ID as the primary key), and another table containing order information (including a unique order ID as the primary key and a foreign key that links each order to a specific customer). This ensures that each piece of data is stored only once, and changes to customer information can be made without affecting the order information, and vice versa.

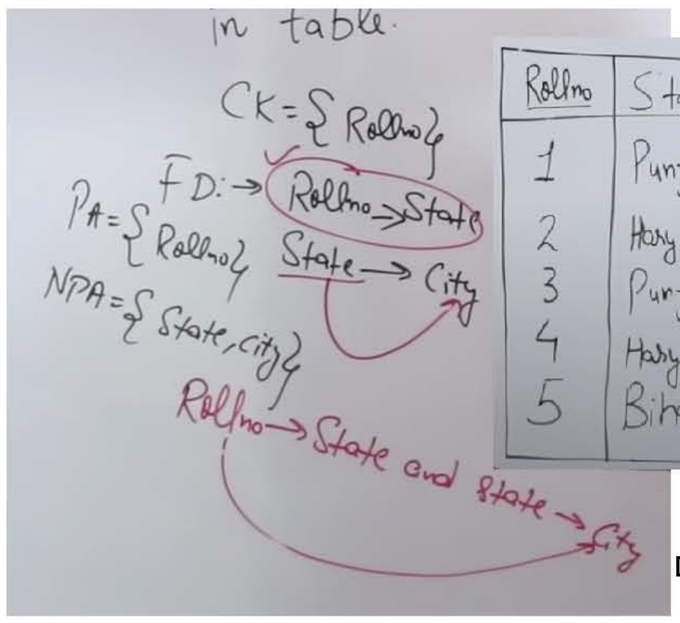
Third Normal Form

Relation schema R is in 3NF if and only if it is in 2NF and **no** non-prime attribute of R is transitively dependent on any key of R

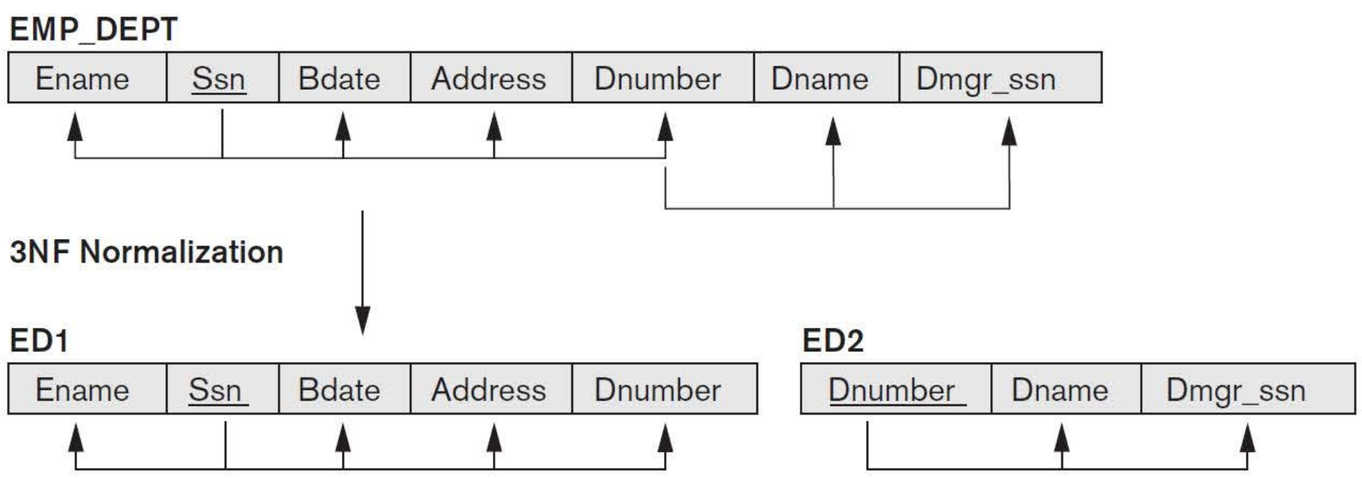
OR

A relation schema *R* is in **third normal form (3NF)** if, whenever a *nontrivial* functional dependency $X \rightarrow A$ is either
(a) *X* is a superkey
(b) *A* is a prime attribute

In simple words, FD: non-prime -> non-prime is invalid in 3NF and there should be no transitive dependencies



1. FD: STATE->CITY (non-prime->non-prime)
2. city (non-prime) is transitively dependent on Roll-no(key)



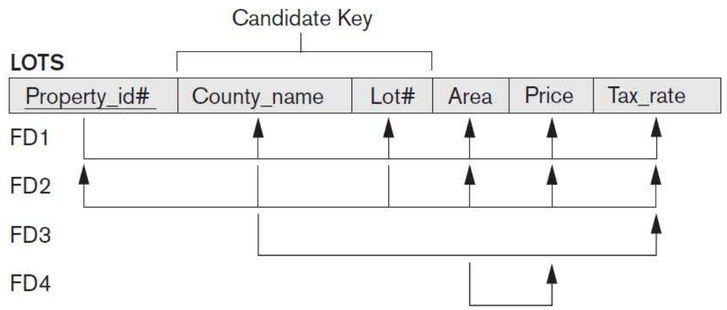
Why 3NF?

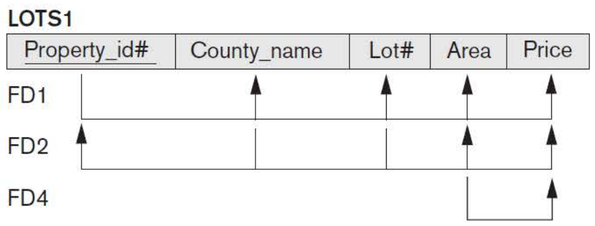
The purpose of 3NF is to ensure that each table contains only data that is directly related to the primary key, and that all non-key attributes are not dependent on other non-key attributes. By doing so, we can avoid data redundancy and inconsistency, and ensure that our database is properly normalized.

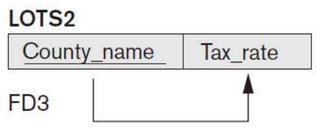
If a table is not in 3NF, it means that it contains transitive dependencies, where some non-key attributes depend on other non-key attributes. This can lead to data redundancy and inconsistency. For example, consider a table that contains customer orders, where each row contains the order ID, customer ID, customer name, product name, and product category. In this case, the product category depends only on the product name, which is not part of the primary key. If we store this information in the same table, we may end up with multiple rows for each product category, potentially leading to data redundancy.

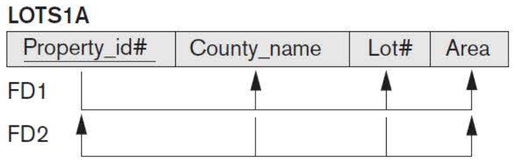
To avoid these issues, we can normalize the table into three separate tables, with one table containing customer information (including a unique customer ID as the primary key), another table containing product information (including a unique product ID as the primary key and a foreign key that links each order to a specific product), and a third table containing order information (including a unique order ID as the primary key and foreign keys that link each order to a specific customer and product). This ensures that each piece of data is stored only once, and changes to product information can be made without affecting the order or customer information, and vice versa.

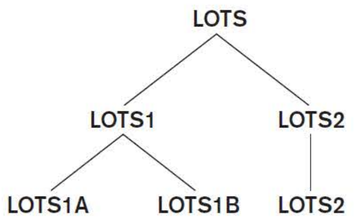
Summary











1NF

2NF

3NF

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Normal Forms: Review

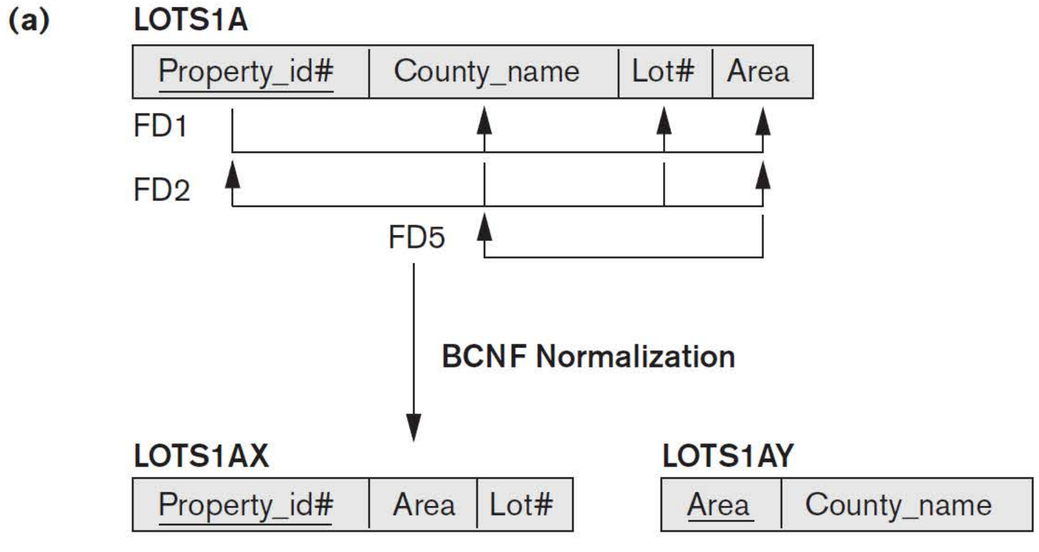
1 NF – No multivalued attributes

2 NF – 1 NF plus no partial dependencies

3 NF – 2 NF plus no transitive dependencies

BCNF

Definition. A relation schema *R* is in **BCNF** if whenever a *nontrivial* functional dependency $X \rightarrow A$ holds in *R*, then *X* is a superkey of *R*.

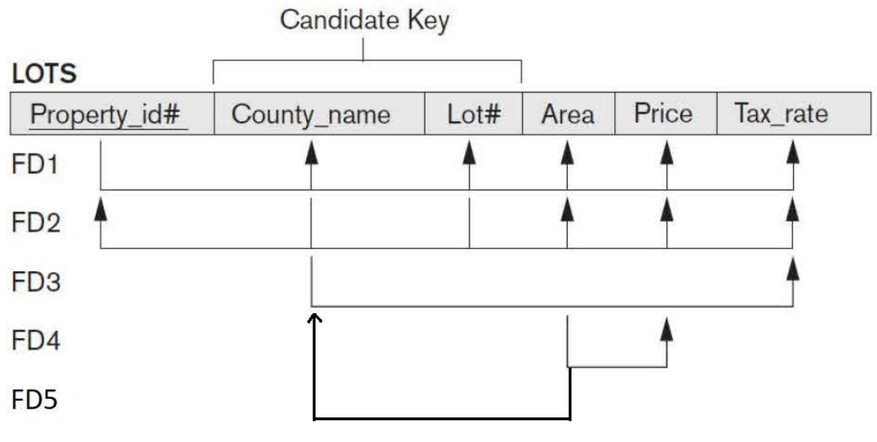


FD5: Area → County_name. If we add this to the other dependencies, the relation schema LOTS1A still is in 3NF because this f.d. conforms belongs to 3NF, County_name being a prime attribute.

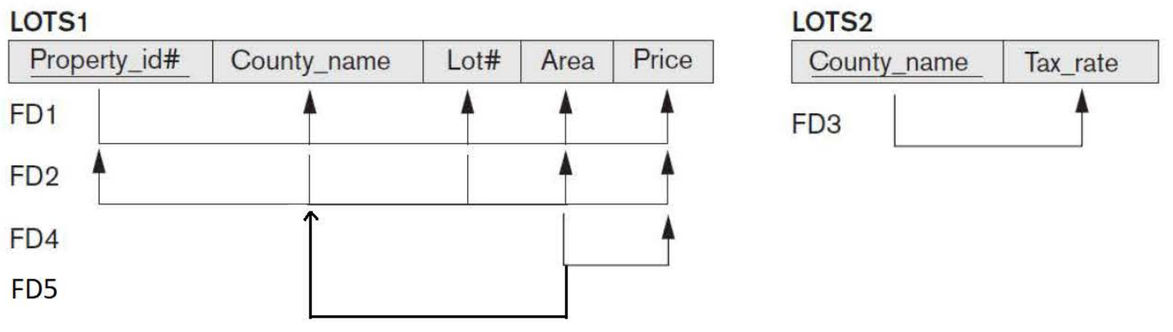
FD5 violates BCNF in LOTS1A because Area is not a superkey of LOTS1A.

We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY. This decomposition loses the functional dependency FD2because its attributes no longer coexist in the same relation after decomposition

1NF

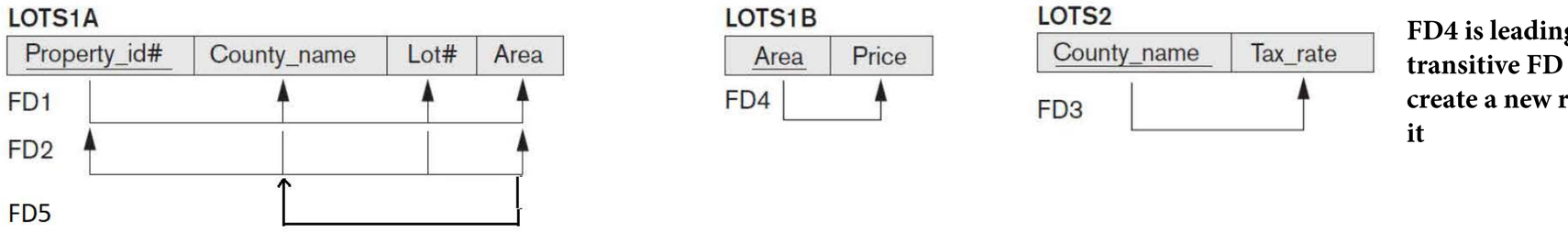


2NF



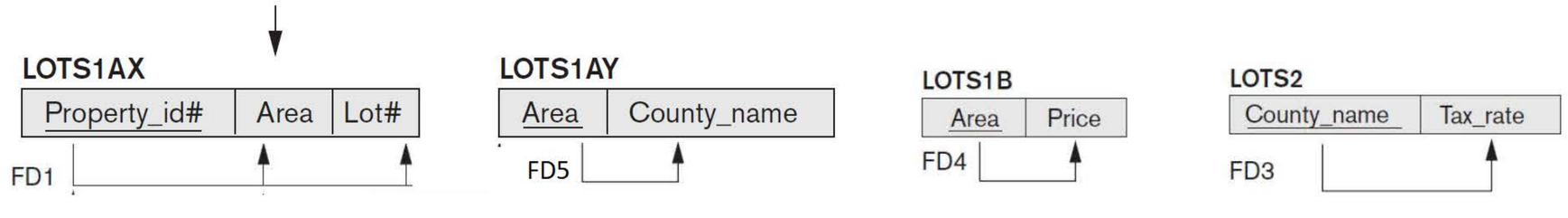
FD3: PARTIAL FUNCTIONAL DEPENDENCY
So we create a new table for the FD

3NF



FD4 is leading transitive FD => we create a new relation for it

BCNF



FD5: A candidate key is FD on a non prime attributs

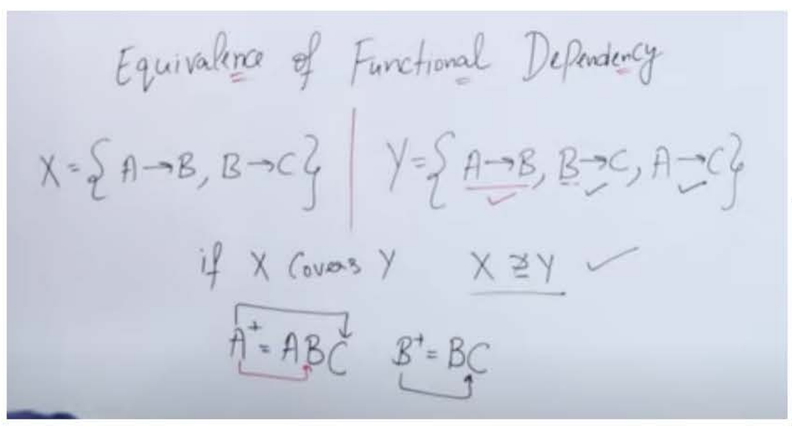
INTERFERENCE RULES

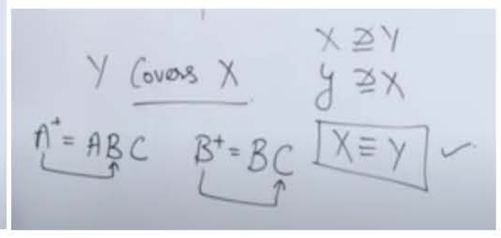
1. Reflexive rule
   $F \models \{X \rightarrow Y \mid Y \subseteq X\}$ for any X. — If Y is a subset of X then X -> Y
2. Augmentation rule
   $\{X \rightarrow Y\} \models \{XZ \rightarrow YZ\}, Z \subseteq R.$ — If X -> Y then XZ -> YZ
3. Transitive rule
   $\{X \rightarrow Y, Y \rightarrow Z\} \models \{X \rightarrow Z\}$ — If X -> Y and Y -> Z then X -> Z
4. Decomposition or Projective rule
   $\{X \rightarrow YZ\} \models \{X \rightarrow Y\}$ — If X -> YZ then X -> Yand X -> Z
5. Union or Additive rule
   $\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$ — If X -> Y and X -> Z then X -> YZ
6. Pseudo transitive rule
   $\{X \rightarrow Y, WY \rightarrow Z\} \models \{WX \rightarrow Z\}$ — If X -> Y and WY -> Z then WX -> Z

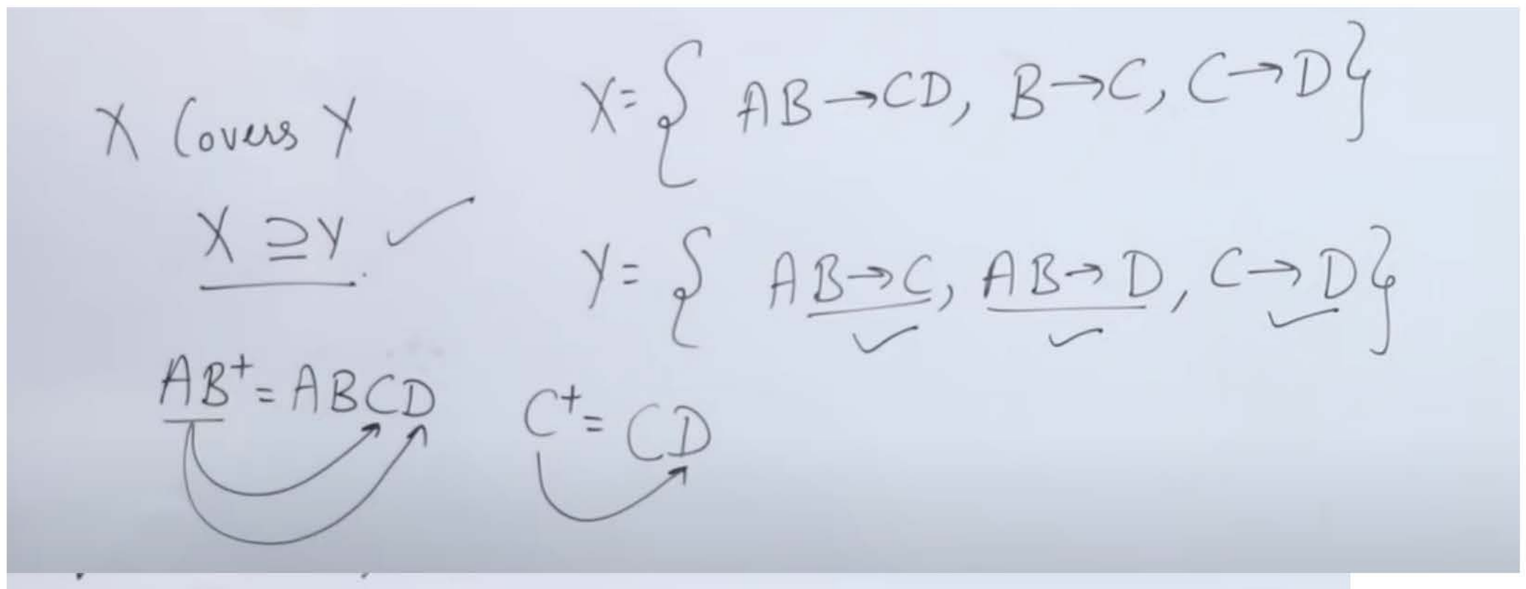
Equivalance of set of FD's

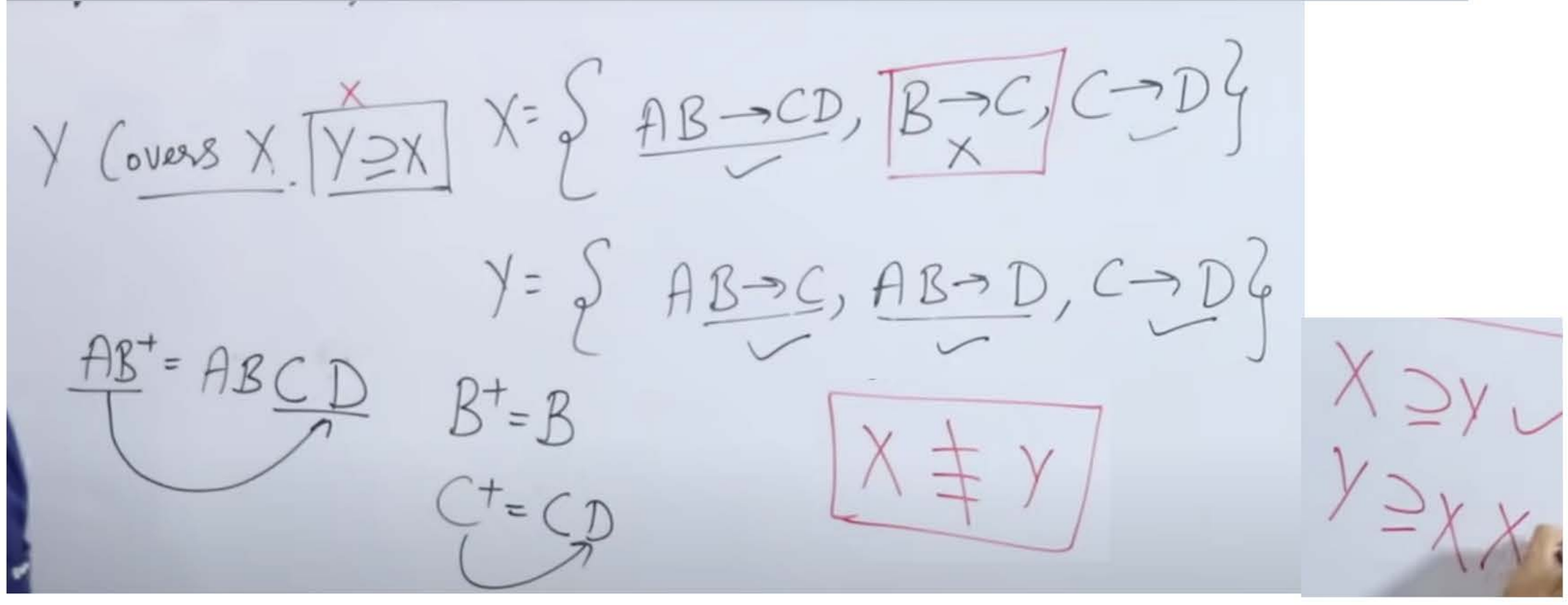
F COVERS E -> A set of functional dependencies F is said to cover another set of functional dependencies E if every FD of E appears in closure of F (F+) ie every FD in E can be inferred from F.

Equivalence: Two sets of FD's F and E are said to be equivalent if F+ = E+ ie Every FD in F can be inferred from E and every FD in E can be inferred from F -> **E covers F and F covers E**









TRANSACTION CONTROL

BY SHIVKARAMAN

Transactions, Database Items, Read and Write Operations, and DBMS Buffers

Transaction: It is a set of operations used to perform a logical unit of work
A transaction generally represents change in database . It includes one or more database access operations insert / delete / update /delete operations

These database operations can be embeded inside an application program or can be queried in a high level language like sql

If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction**; otherwise it is known as a **read-write transaction**

DB items: A **database** is basically represented as a collection of *named data items.* The size of a data item is called its **granularity**.
A**data item** can be a ***database record***, but it can also be a larger unit such as a whole ***disk block***, or even a smaller unit such as an **individual *field (attribute)*** *value* of some record in the database. The transaction processing concepts are independent of the data item granularity (size) and apply to data items in general

Operations: The basic database access operations that a transaction can include are as follows:

- **read_item(*X*).** Reads a database item named *X* into a program variable.
- **write_item(*X*).** Writes the value of program variable *X* into the database item named *X*.

The above two operations are used to retrieve the value of variables from the daabase to RAM.
When we use write(X) we are not directly updating the variable in the database, it is still the same in database

- **commit.** It updates the value in database

The **read-set** of a transaction is the set of all items that the transaction reads, and the **write-set** is the set of all items that the transaction writes

DBMS Cache: The DBMS will maintain a **database cache** which has number of **data buffers** in main memory.
Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed.
When these buffers are all occupied, and additional database disk blocks must be copied into memory, some **buffer replacement policy** is used to choose which of the current occupied buffers is to be replaced.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

Schedule — A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

Serial Schedule — It is a schedule in which transactions are aligned in such a way that one transaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.**- view equivalence and conflict equivalence**

Equivalence Schedules

An equivalence schedule can be of the following types —

1. Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

2. View Equivalence

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

For example —

If T reads the initial data in S1, then it also reads the initial data in S2.
If T reads the value written by J in S1, then it also reads the value written by J in S2.
If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

3. Conflict Equivalence

Two schedules would be conflicting if they have the following properties —

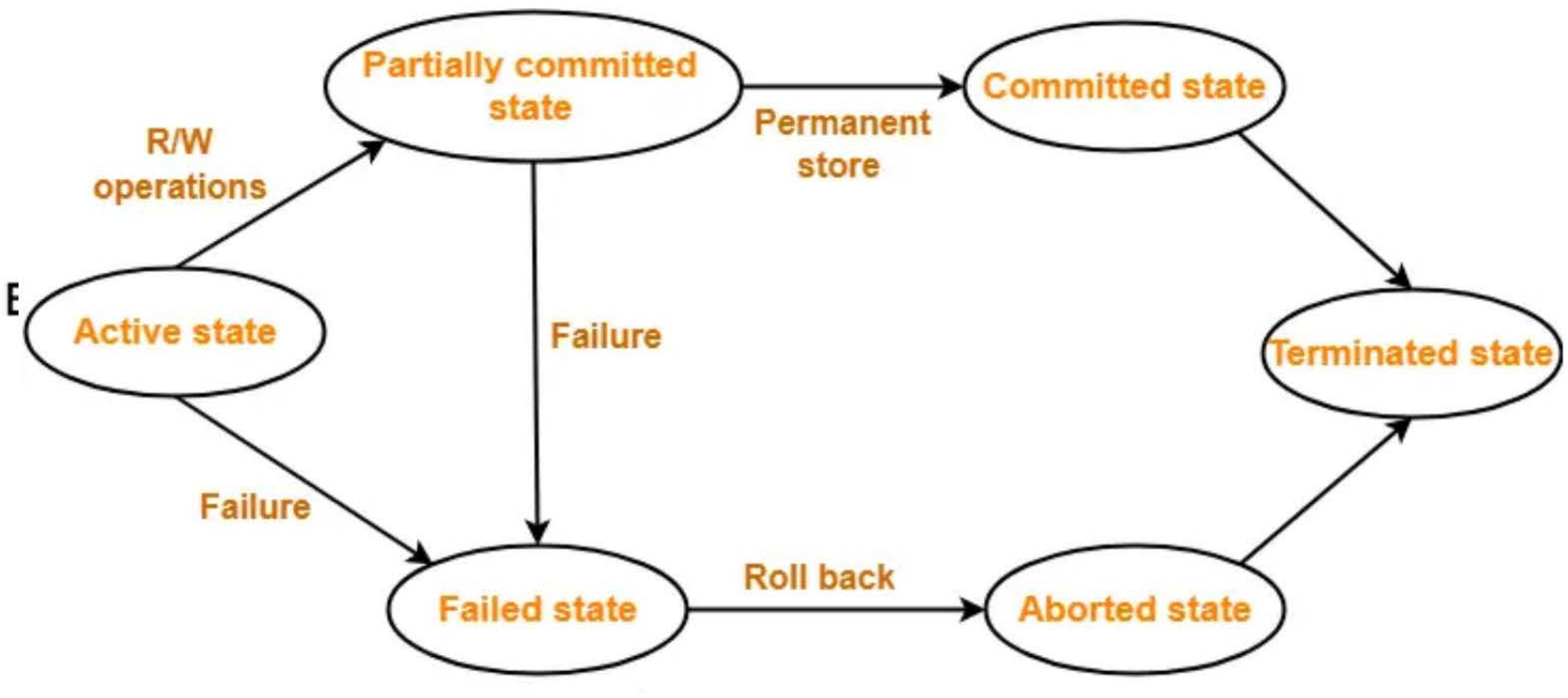
Both belong to separate transactions.
Both accesses the same data item.
At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if —

Both the schedules contain the same set of Transactions.
The order of conflicting pairs of operation is maintained in both the schedules.

States of Transactions

A transaction in a database can be in one of the following states —



`COMMIT` command

`COMMIT` command is used to permanently save any transaction into the database.

When we use any DML command like `INSERT`, `UPDATE` or `DELETE`, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the `COMMIT` command to mark the changes as permanent.

Following is commit command's syntax,

```
COMMIT;
```

`ROLLBACK` command

This command restores the database to last commited state. It is also used with `SAVEPOINT` command to jump to a savepoint in an ongoing transaction.

If we have used the `UPDATE` command to make some changes into the database, and realise that those changes were not required, then we can use the `ROLLBACK` command to rollback those changes, if they were not commited using the `COMMIT` command.

Following is rollback command's syntax,

```
ROLLBACK TO savepoint_name;
```

`SAVEPOINT` command

`SAVEPOINT` command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

```
SAVEPOINT savepoint_name;
```

In short, using this command we can **name** the different states of our data in any table and then rollback to that state using the `ROLLBACK` command whenever required.

Concurrency Control

Transactions submitted by various users may execute concurrently. Hence some sort of concurrency control is needed.

(a)

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

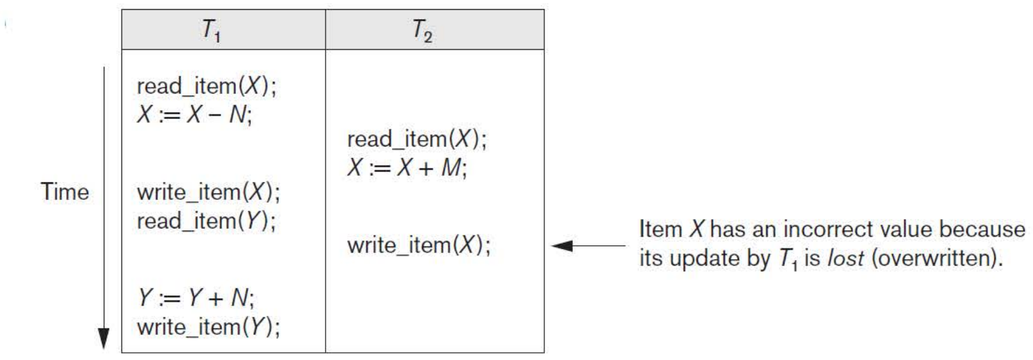
T_2

read_item(X);
$X := X + M$;
write_item(X);

1. 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 items incorrect

Suppose that transactions *T*1 and *T*2 are submitted atapproximately the same time, and suppose that their operations are interleaved as shown below



If X=80 Initiallly (No of flights) and N = 5 and M = 4
T1 -> X transfers 5 seats (N) to Y => X = 75
T2 -> X buys 4 seats (M) => X = 79
Hence if everything goes well, X should be equal to 79

Since the T1 is lost, the final value is corrupted and X = 84

2. The Temporary Update (or Dirty Read) Problem.

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value.

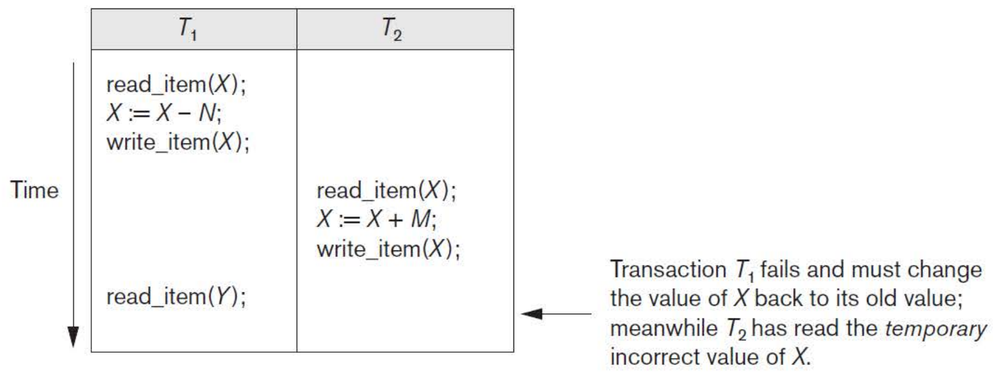
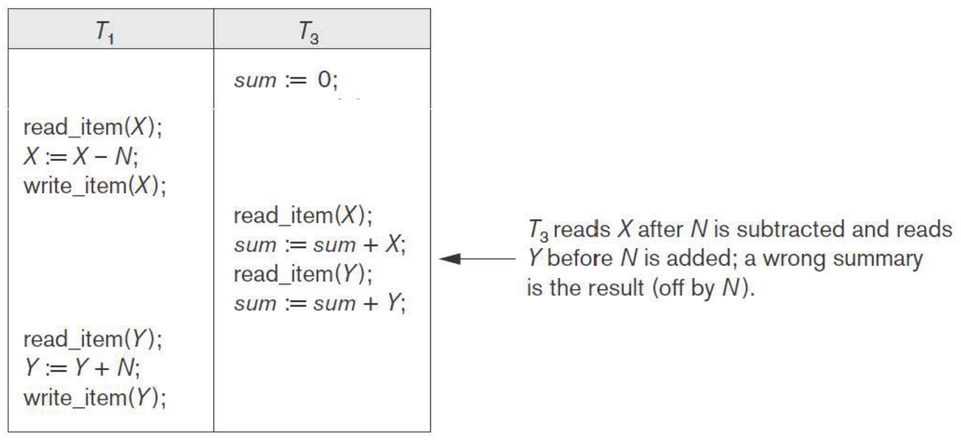


Figure shows an example where *T*1 updates item *X* and then fails before completion, so the system must roll back *X* to its original value. Before it can do so, however, transaction *T*2 reads the *temporary* value of *X*. The value of item *X* that is read by *T*2 is called ***dirty data*** because it has been created by a transaction that has not completed and committed yet; hence, this problem is also known as the ***dirty read problem*.**

3. The Incorrect Summary Problem

This problem occurs when one transaction(T1) is updating some items and other transaction(T2) with some aggrigate function kicks in. The aggregate function may calculate some values before they were updated (by T1).



Transaction T3 was suposed to start after execution of T1. But here, T3 comes into the picture reading the wrong values

4.The Unrepeatable Read Problem

This problem occurs when a transaction *T* reads the same item twice and the item is changed by another transaction *T'* between the two reads. Hence, *T* receives *different values* for its two reads of the same item.

RECOVERY AND FAILURES

Whenever a transaction is submitted to DBMS for execution, the system is responsible for making sure that all the operations of the transactions are succesfully executed and updates the database permanently (**COMMITED**) or the transaction is dropped and doesnot have any effect on the database(**ABORTED).**
If a transaction fails after execution some statements, but before executing all statements, all the statements already executed should be undone and the transaction should be aborted.

Types of Transaction failures

1. A computer failure (system crash). A hardware, software, or network error occurs in the computer system during transaction execution.
for example, main memory failure.

2. A transaction or system error. Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. Additionally, the user may interrupt the transaction during its execution.

3. Local errors or exception conditions detected by the transaction. During transaction execution, certain conditions may occur that necessitate cancellation of the transaction.
For example, data for the transaction may not be found. An exception condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled.

This exception could be programmed in the transaction itself, andin such a case would not be considered as a transaction failure.

4. Concurrency control enforcement. The concurrency control method may abort a transaction because it violates serializability or it may abort one or more transactions to resolve a state of deadlock among several transactions . Transactions aborted because of serializability violations or deadlocks are typically restarted automatically at a later time.

5. Disk failure. Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

6. Physical problems This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwritingdisks or tapes by mistake, and mounting of a wrong tape by the operator.

Properties of transactions

1. Atomicity: A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all ie **EITHER ALL OR NULL**

A failed transaction cannot be resumed, it must be restarted from beginning

2. Consistency:

A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

3. Isolation.

A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with other transactions executing concurrently.

4.Durability

The changes applied to the database by a committed transaction must persist in the database. These changes must not be lost because of any failure.

How to impliment Atomicity and Durabililty?

The recovery-management component of a database system can support atomicity and durability by a variety of schemes.
Here we are going to learn about one of the simplest scheme called Shadow copy.

Shadow copy:
In the shadow-copy scheme, a transaction that wants to update the database first **creates a complete copy of the database. All updates are done on the new database copy**, leaving the original copy, the shadow copy, untouched. If at any point the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.
This scheme is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time.

If the transaction completes, it is committed as follows:
1. First, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. (Unix systems use the flush command for this purpose.

2. After the operating system has written all the pages to disk, the database system updates the pointer db-pointer to point to the new copy of the database; the new copy then becomes the current copy of the database. The old copy of the database is then deleted.

How the technique handles transaction failures:
If the transaction fails at any time before db-pointer is updated, the old contents of the database are not affected.
We can abort the transaction by just deleting the new copy of the database.
Once the transaction has been committed, all the updates that it performed are in the database pointed to by db pointer.
Thus, either all updates of the transaction are reflected, or none of the effects are reflected, regardless of transaction failure.

How the technique handles system failures:
Suppose that the system fails at any time before the updated db-pointer is written to disk. Then, when the system restarts, it will read db-pointer and will thus see the original contents of the database, and none of the effects of the transaction will be visible on the database.
Next, suppose that the system fails after db-pointer has been updated on disk. Before the pointer is updated, all updated pages of the new copy of the database were written to disk.
Again, we assume that, once a file is written to disk, its contents will not be damaged even if there is a system failure. Therefore, when the system restarts, it will read db-pointer and will thus see the contents of the database after all the updates performed by the transaction.

DBMS - Concurrency Control

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

- Lock based protocols
- Time stamp based protocols

Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

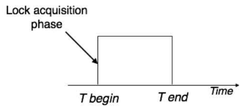
There are four types of lock protocols available −

1. Simplistic Lock Protocol

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the 'write' operation.

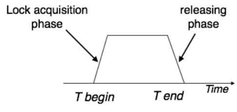
2. Pre-claiming Lock Protocol

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



3. Two-Phase Locking 2PL

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

21.1.2 Guaranteeing Serializability by Two-Phase Locking

- **Two-phase locking protocol**
  - – All locking operations precede the first unlock operation in the transaction
  - – Phases
    - Expanding (growing) phase
      - – New locks can be acquired but none can be released
      - – Lock conversion upgrades must be done during this phase
    - Shrinking phase
      - – Existing locks can be released but none can be acquired
      - – Downgrades must be done during this phase

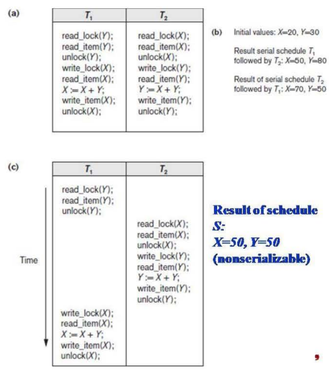


Figure 21.3 Transactions that do not obey two-phase locking (a) Two transactions $T1$ and $T2$ (b) Results of possible serial schedules of $T1$ and $T2$ (c) A nonserializable schedule S that uses locks

- If every transaction in a schedule follows the two-phase locking protocol, **schedule guaranteed to be serializable**
- Two-phase locking **may limit the amount of concurrency** that can occur in a schedule
- Some **serializable schedules will be prohibited** by two-phase locking protocol

4. Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.