DATABASE DESIGN COURSE

INTRODUCTION TO DATABASES

- Data: A distinct unit of information. Data is information that has been translated into a form that is efficient for movement or processing.
- Database: An organized collection of data used for the purpose of modelling some type of organization or organizational data. Any application used to store or collect data in an organized manner for a specific purpose.
- Types of Databases: Operational and Analytical.

Operational: Primarily used for OLTP(Online Transactional Processes) where there is need to collect, modify and maintain data on a daily basis. Constantly changing database

Analytical: Primarily used for OLAP(Online Analytical Processes) where there is need to store historical data. Data isn't updated, modified, deleted.

- Relational Databases: A relational DB stores data in relations which is presented to user as tables. Relations are composed of tuples(or records) and attributes(or fields).
  - Physical order of records or fields in a table is irrelevant
  - Each record is identified by a field(or group of fields) that contain unique values
  - This enables physical data independence
  - Relation between tables is established by matching values of shared/common fields

- SQL : A standard language used to create, modify, maintain and query a relational DB.

Student Table

Attribute 1	Attribute 2	Attribute 3	Attribute 4
Student_ID	First_name	Last_name	Course_ID
1	John	Joe	1
2	Rick	Grimes	3
3	John	Joe	2
4	Amy	Jones	2

Tuple 1

Tuple 2

Tuple 3

Tuple 4

Course Table

Course_ID	Course_name
1	DBMS
2	Java
3	Python

ADVANTAGES OF RDB

- Multiple levels of Integrity: At table level it helps avoid duplicate records using primary keys(discussed later) and at relationship level it ensures shared columns are valid using referential integrity(discussed later).
- Logical and physical data independence: Logical and physical layers are independent of changes made to each other(discussed later)
- Enforcing data consistence and accuracy: Constraints and integrity ensure that data is accurate and consistent to certain degree.
- Ease of retrieval of data: Join tables to retrieve data from multiple tables.

RDBMS(Relational Database Management System)

- A software application program that enables you to create, modify, maintain and manipulate a Relational DB.
  - MySQL
  - PostgreSQL
  - Oracle
  - Microsoft SQL Server
  - IBM DB2

Database Design: What and Why

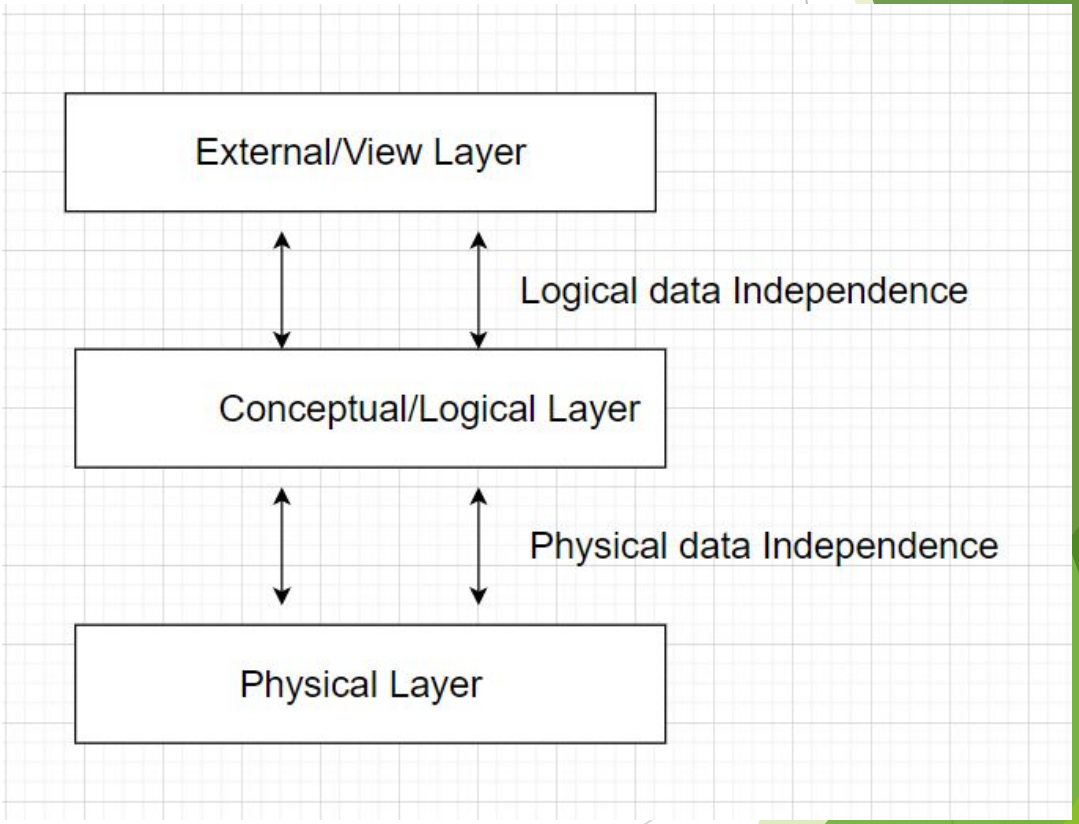
- Database Design: Process of designing a database to facilitate its development and implementation. It includes deciding the naming, correlation, datatypes, constraints and structure of database and tables.
  - Crucial to consistency and integrity of data that is retrieved from a DB.
  - Might lead to inaccurate results if not done properly.
  - Has adverse impact on business.
  - Analogous to deciding the blueprint and design of house before construction.

3-Layer Architecture of DBMS

View of data in User Interface on the basis of conceptual level tables.

Data is represented in the form of
views
and tables.

Consists of information about the location of DB objects in the data store is kept in secondary storage devices such as tapes and disks.



Data Independence

- The ability to change the schema at one level of a database system without having to change the schema at the next higher level is called "Data Independence".
- Physical data independence: Any change in the physical location of tables and indexes should not affect the conceptual level or external level
- Conceptual data independence: Adding or deleting an attribute of a table should not affect the user’s view of the table.

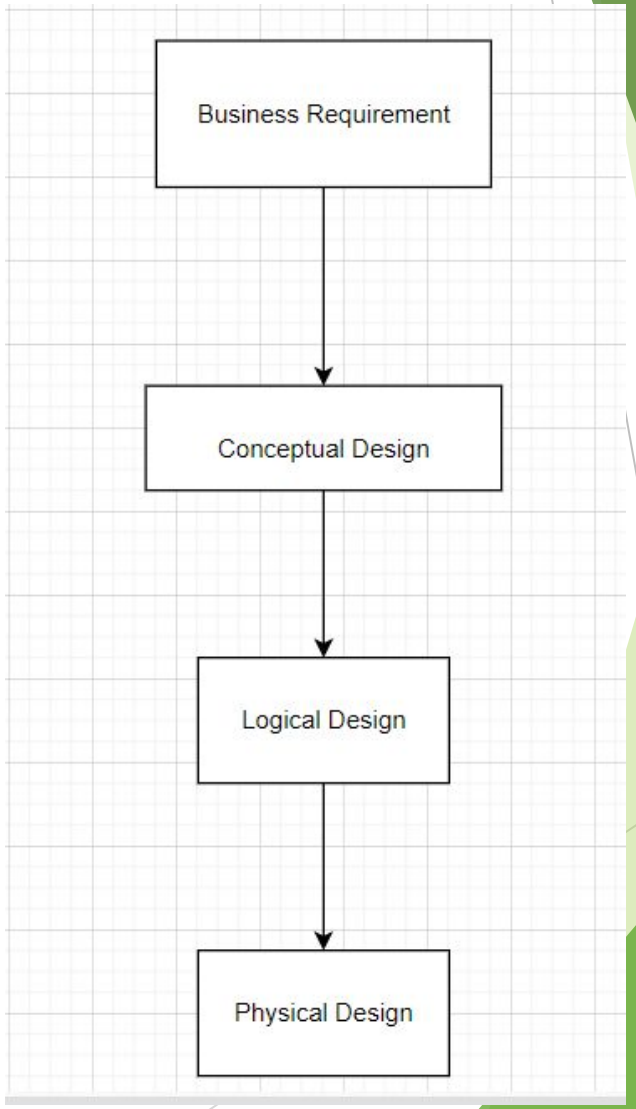
Phases of DB design

Understanding the business in terms of events, entities and functions.

ER model, constraints, relationships etc.

Table structure, ER model to Relational model

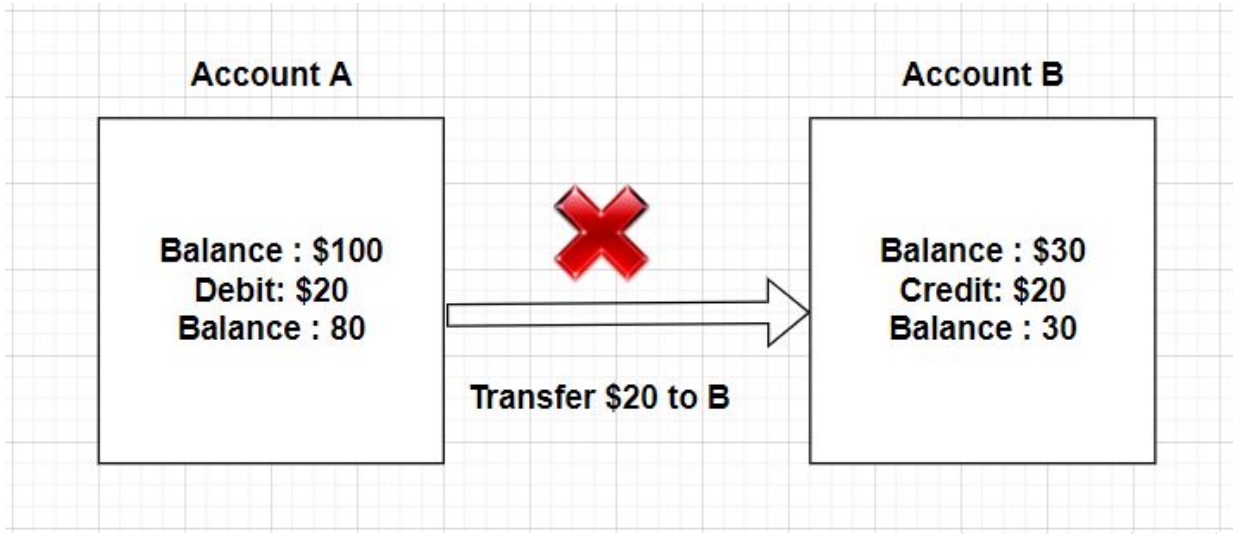
Data in relational model to implementation in DBMS

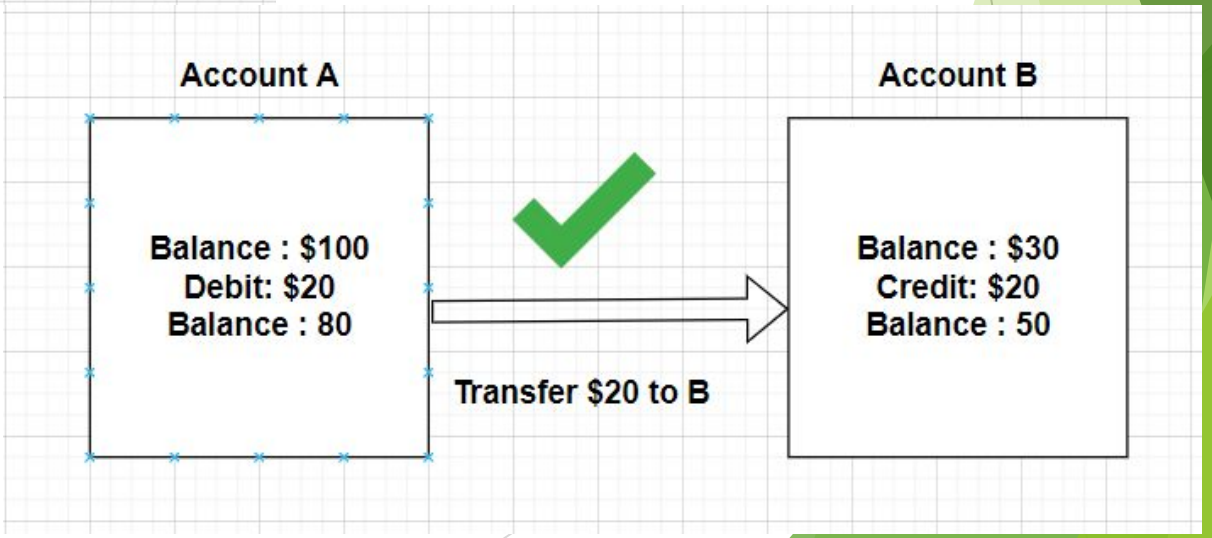


ACID properties

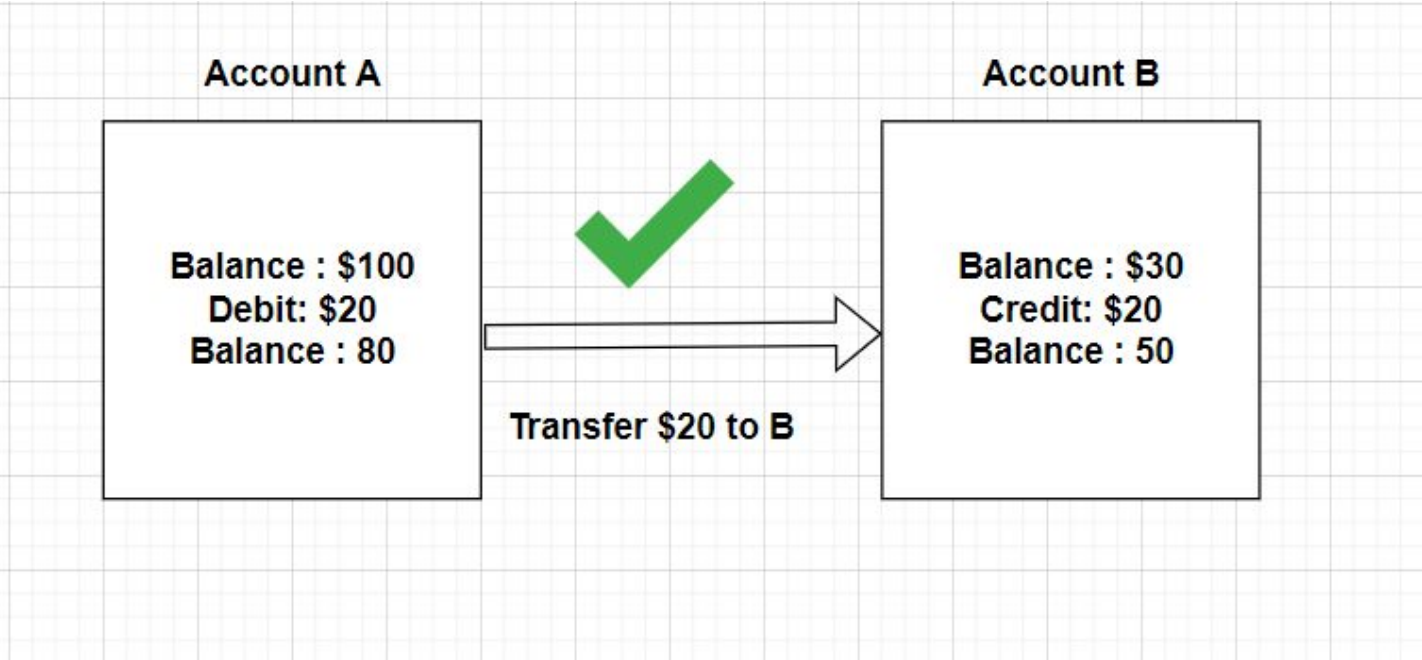
- A Database Transaction is a logical unit of processing in a DBMS which entails one or more database access or write operation.
- To maintain the integrity of data 4 types of properties are described:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Atomicity: Data remains atomic. Any operation performed on data should be performed completely or not performed at all. Partial execution is not permitted.



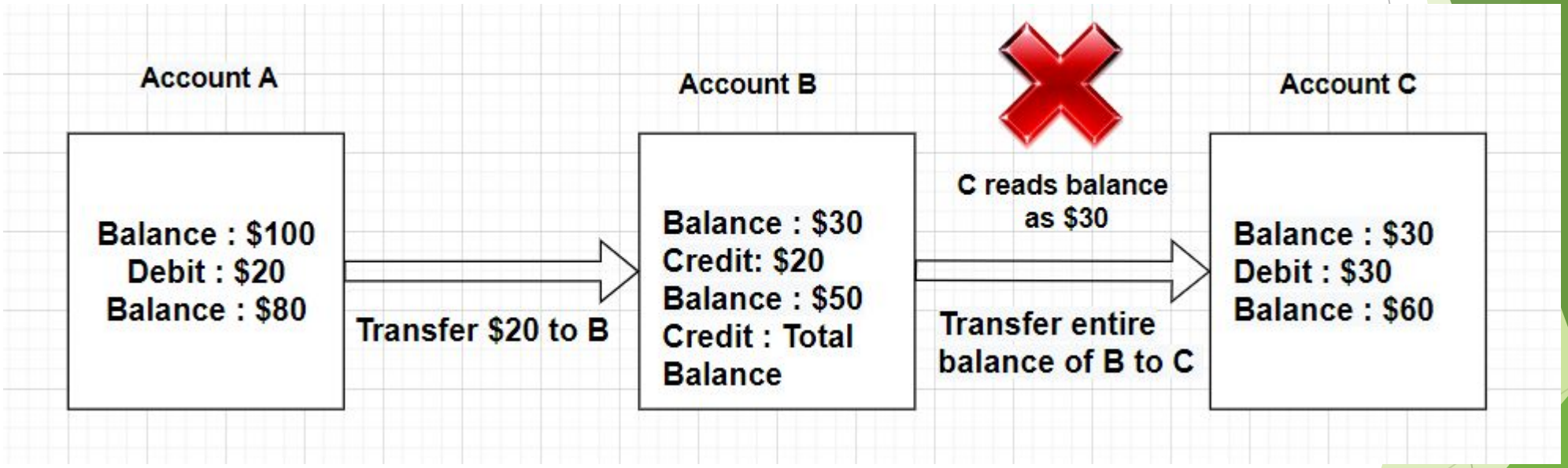


- Consistency: Value should always be preserved. Database must be conserved before and after the transaction.



Value before transaction = $100 + $30 =$130
Value after transaction = $80 + $50 =$130

- Isolation: Property of DB where no data should affect the other one when many transactions occur concurrently. Operation on one DB should start only when the operation on the first DB gets done. Both operations must occur separately. Else ensure that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.



C reads the balance of B before the first transaction is completed.
Hence isolation is violated.

- Durability: Ensures that once transaction has completed execution, updates and modifications to the DB are stored in and written to disk and they persist.

- Data modelling: process of creating a visual representation of either a whole information system or parts of it to communicate connections between data points and structures. Goal is to illustrate the types of data used and stores within the system, the relationships among these data types, the ways the data can be grouped and organized and its format and attributes.
  - ER model
  - Relational model
  - Object based model
  - Semi-structured data model

- Schema: Represents the overall logical design of a complete DB. In a DBMS application, a schema is a container for tables, views, primary keys, stored procedures etc.

- Entity Relationship Model(ER Model): An entity–relationship model (or ER model) describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types (which classify the things of interest) and specifies relationships that can exist between entities (instances of those entity types).
  - Entity: something about which we store data. E.g., Customer, Product. Need not necessarily be tangible. E.g., Departments, Courses. Entities are described by their attributes.
  - Weak entity: Depends on another entity.

- Attributes: must be single-valued. In case of multiple values create new instance.

Student_ID	Phone_no
1	85555, 96555
2	85495, 10005

Student_ID	Phone_no_1	Phone_no_2
1	85555	96555
2	85495	10005

- ER diagram: A way to document the entities in a DB, along with the attributes that describe them.
- Types of attributes: Key attributes, multi-valued etc.

- Key attribute: Main attribute of an entity. Represents the primary key. Represented by an ellipse with text underlined.

  E.g., Student_ID

- Multi-valued attribute: Attribute that takes up multiple values. Denoted by concentric ellipses.

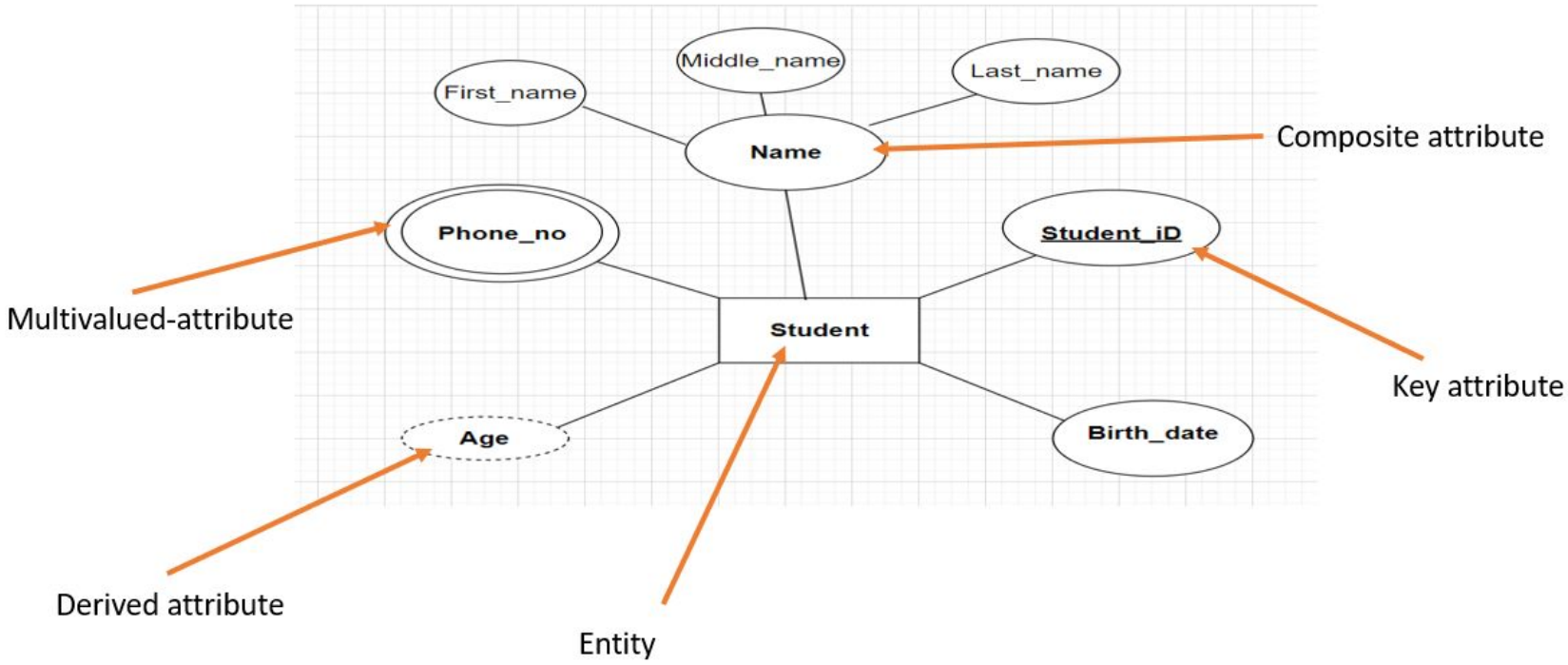
  E.g., Multiple phone numbers for one Student.

- Derived attribute: Attribute that can be derived from other attributes.

  E.g., Age can be derived from birth_date.

- Composite attribute: Attribute constituted by multiple attributes.

  E.g., Address=>House_no+Street_no+city+PIN


Middle_name
First_name
Last_name
Name
Composite attribute
Phone_no
Student_iD
Multivalued-attribute
Student
Key attribute
Age
Birth_date
Derived attribute
Entity

- Relational Model: In relational model, the data and relationships are represented by collection of inter-related tables. Each table is a group of column and rows, where column represents attribute of an entity and rows represents records.
- Boyce-Codd rules:
  - **Information** All information in a relational database must be logically represented as column values in rows within tables.
  - **Guaranteed Access** Every value in a table is guaranteed to be accessible through a combination of table name, primary key value, and column name.
  - **Systematic Treatment of Nulls** Nulls must be represented and treated in a systematic way, independent of data type.
  - **Dynamic On-Line Catalog Based on the Relational Model** The metadata must be stored and managed as ordinary data, that is, in tables within the database. Such data must be available to authorized users using the standard database relational language.
  - **Comprehensive Data Sublanguage** The relational database may support many languages. However, it must support one well defined, declarative language with support for data definition, view definition, data manipulation (interactive and by program), integrity constraints, authorization, and transaction management (begin, commit, and rollback).
  - **View Updating** Any view that is theoretically updatable must be updatable through the system.
  - **High-Level Insert, Update and Delete** The database must support set-level inserts, updates, and deletes.
  - **Physical Data Independence** Application programs and ad hoc facilities are logically unaffected when physical access methods or storage structures are changed.

- Boyce-Codd rules (continued)….
  - **Logical Data Independence:** Application programs and ad hoc facilities are logically unaffected when changes are made to the table structures that preserve the original table values (changing order of column or inserting columns).
  - **Integrity Independence:** All relational integrity constraints must be definable in the relational language and stored in the system catalog, not at the application level.
  - **Distribution Independence:** The end users and application programs are unaware and unaffected by the data location (distributed vs. local databases).
  - **Nonsubversion:** If the system supports low-level access to the data, there must not be a way to bypass the integrity rules of the database.
  - **Rule Zero:** All preceding rules are based on the notion that in order for a database to be considered relational, it must use its relational facilities exclusively to manage the database.

- Relational model concepts:
  - **Attribute:** Attributes are the properties that define a relation. e.g.; **Employee_ID**, **Name**
  - **Relation Schema:** A relation schema represents name of the relation with its attributes. e.g.; Employee (Employee_ID, Name, Phone_no, City_ID) is relation schema for Employee. If a schema has more than 1 relation, it is called Relational Schema.
  - **Tuple:** Each row in the relation is known as tuple.
  - **Relation Instance:** The set of tuples of a relation at a particular instance of time is called as relation instance. Table Employee shows the relation instance of Employee at a particular time. It can change whenever there is insertion, deletion or updation in the database.
  - **Degree:** The number of attributes in the relation is known as degree of the relation. The Employee relation defined above has degree 4.

Employee

Employee_ID	Name	Phone_no	City_ID
1	Raj	45555	1
2	Jon	85422	1
3	Annie	36388	3
4	Dan		2

City

City_ID	City_name
1	Delhi
2	Surat
3	Konkan

- **Cardinality:** The number of tuples in a relation is known as cardinality. The **Employee** relation defined above has cardinality 4.
- **Column:** Column represents the set of values for a particular attribute. The column Employee_ID is extracted from relation Employee.
- **NULL Values:** The value which is not known or unavailable is called NULL value. It is represented by blank space. e.g.; Phone_no of Employee having Employee_ID 4 is NULL.

- Constraints in Relational model: Constraints are conditions that are imposed on data in relations. Any insertion, updation or deletion operations must satisfy these constraints before being performed else these will fail.

- Types of Constraints:
  - **Domain Constraints:** These are constraints at attribute level. An attribute can only take values which lie within the range of defined values. E.g., Phone_no of Employee can only be Integer then having an alphabet is not allowed.
  - **Key Integrity:** Every relation in the database should have at least one set of attributes which defines a tuple uniquely. Those set of attributes is called key. E.g., Employee_ID in Employee is a key. No two employees can have same Employee_ID. So a key has two properties:
    - It should be unique for all tuples.
    - It can't have NULL values.
  - **Referential Integrity:** When one attribute of a relation can only take values from other attribute of same relation or any other relation, it is called referential integrity. City_ID in Employee relation can only have values present in City_ID of City relation.

Relational Algebra

- A procedural query language that takes relations as input and returns relations as output after applying operations to the relations.
- Basic operators:
  - Selection(σ): select tuples from a relation based on conditions.

    σ (condition)(relation). E.g., σ (City_ID>1)(Employee)

Employee_ID	Name	Phone_no	City_ID
3	Annie	36388	3
4	Dan		2

Selection operator does not show any result. Require projection for that.

- Projector(Π) : Used to project chosen columns.

  Π(columns separated by commas)(relation). E.g., Π(Name, City_ID)(Employee)

Name	City_ID
Raj	1
Jon	1
Annie	3
Dan	2

- Cross Product(X) : Matches every row in first table to every row in second table. Relation1 X Relation2. E.g., Employee X City

Employee_ID	Name	Phone_no	City_ID
1	Raj	45555	1
2	Jon	85422	1
3	Annie	36388	3
4	Dan		2

City_ID	City_name
1	Delhi
2	Surat
3	Konkan

Employee_ID	Name	Phone_no	City_ID	City_ID	City_name
1	Raj	45555	1	1	Delhi
2	Jon	85422	1	1	Delhi
3	Annie	36388	3	1	Delhi
4	Dan		2	1	Delhi
1	Raj	45555	1	2	Surat
2	Jon	85422	1	2	Surat
3	Annie	36388	3	2	Surat
4	Dan		2	2	Surat
1	Raj	45555	1	3	Konkan
2	Jon	85422	1	3	Konkan
3	Annie	36388	3	3	Konkan
4	Dan		2	3	Konkan

- Union(U): Union on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible** (These two relations should have the same number of attributes and corresponding attributes in two relations have the same domain). Union operator when applied on two relations R1 and R2 will give a relation with tuples that are either in R1 or in R2. Denoted by: R1 U R2

Employee

Employee_ID	Name	City_ID
1	Raj	1
2	Jon	1
3	Annie	3

Student

Student_ID	Name	City_ID
1	Neil	5
2	Jon	5
3	Annie	3

Student U Employee

Student_ID/Employee_ID	Name	City_ID
1	Neil	5
2	Jon	5
3	Annie	3
1	Raj	1
2	Jon	1

- Minus(-): Minus on two relations R1 and R2 can only be computed if R1 and R2 are **union compatible**. Minus operator when applied on two relations as R1-R2 will give a relation with tuples that are in R1 but not in R2. Denoted by R1-R2

Employee - Student

Student_ID/Employee_ID	Name	City_ID
1	Raj	1
2	Jon	1

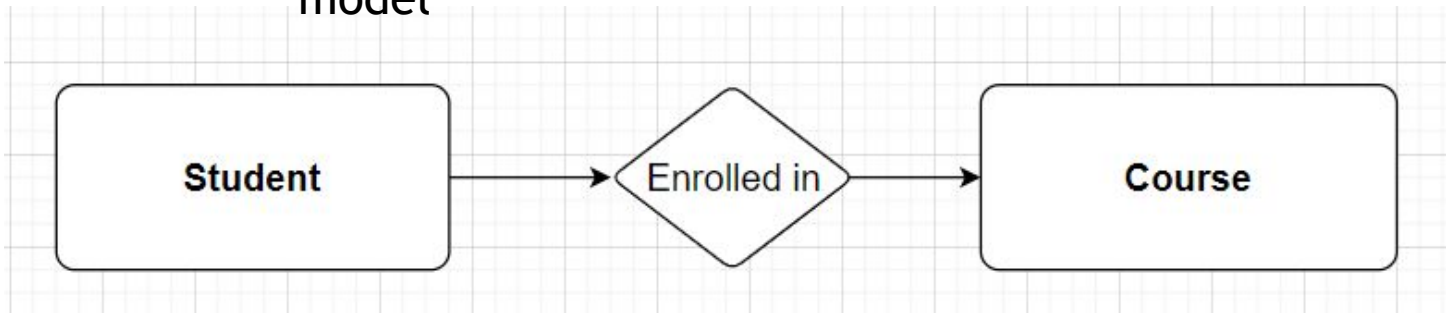
- Rename(ρ): Rename operator is used to give another name to a relation.

Joins in DBMS

- Left Join
- Right Join
- Inner Join
- Full Join
- Cross Join

- Relationship type: represents the association between entity types. This can be established via a set of primary and foreign keys.

Representation of Relationship in ER model



- Types of relationships:
  - One-to-one
  - One-to-many
  - Many-to-many

- One-to-one relationship (1:1): When a single record in the first table is related to only one record in the second table, and single record in second table is related to only one record in first table.

Employee

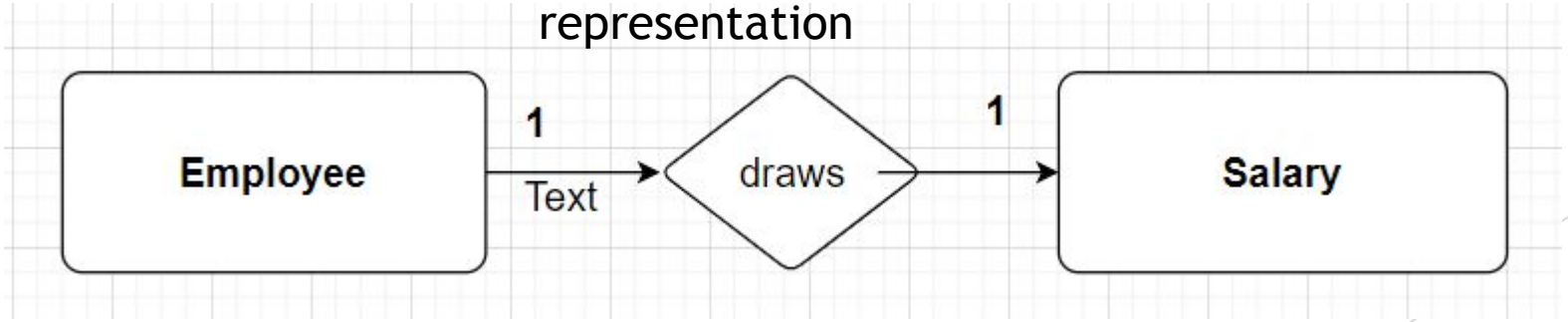
Relational model representation

Employee_ID	Name	Phone_no
1	Raj	45555
2	Jon	85422
3	Annie	36388

Salary

Employee_ID	Salary_per_month
1	3455
2	6566
3	7866

ER model representation



- One-to-many-relationship (1:M): When a single record in the first table can be related to many records in the second table , but a single record in the second table can be related to only one record.

Relational model representation

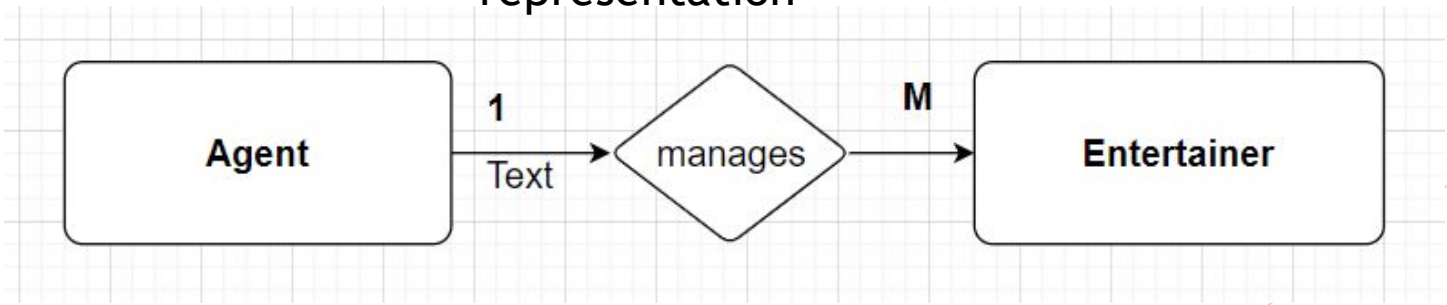
Agent

Agent_ID	Name	Phone_no
1	Raj	45555
2	Jon	85422
3	Annie	36388

Entertainer

Entertainer_ID	Name	Agent_ID
1	Emil	1
2	Rita	1
3	Dean	3

ER model representation



- Many-to-many relationship (M:N): When a single record in the first table can be related to many records in the second table , a single record in the second table can be related to many records in the first.

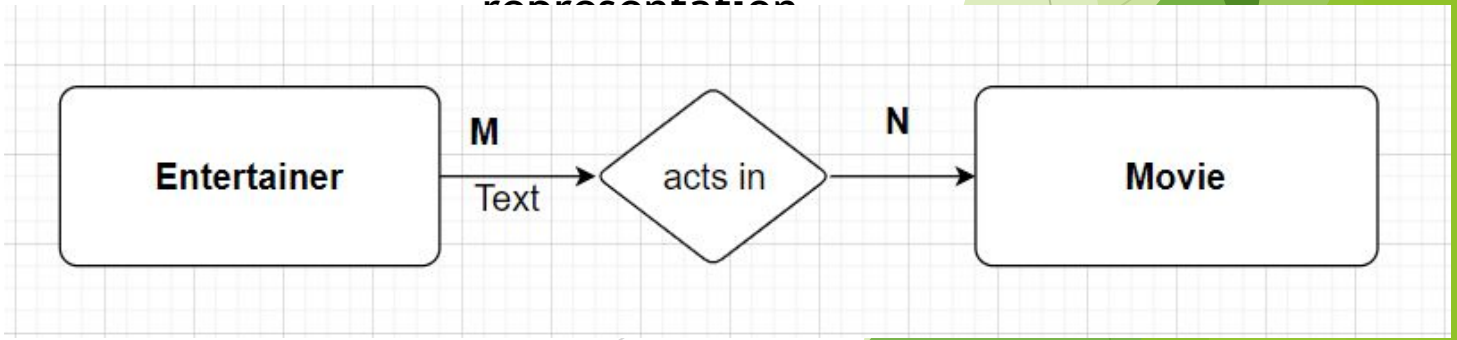
Relational model representation

Movie_ID	Movie_name
1	The last Airbender
2	Dune
3	Harry Potter 1

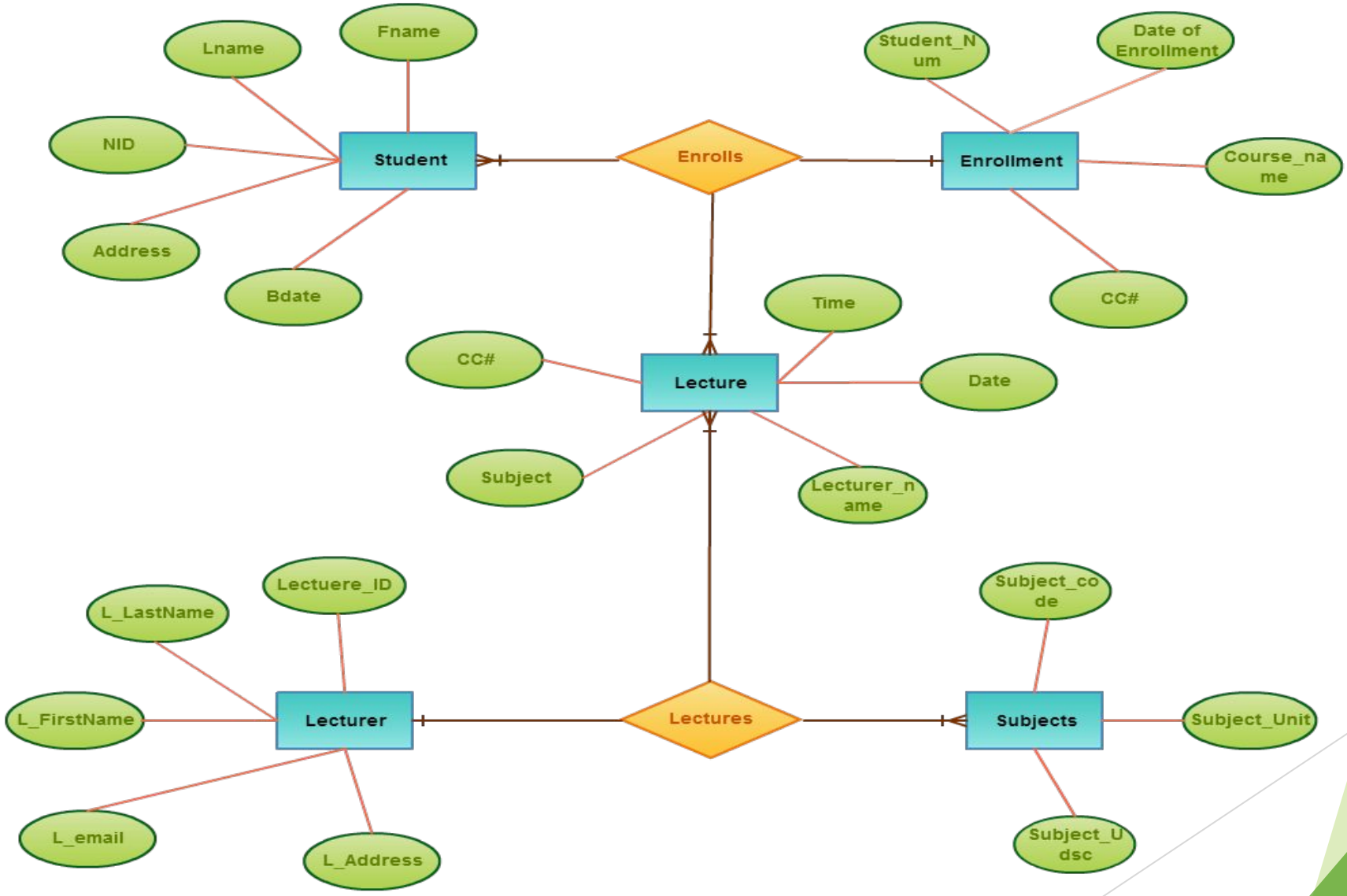
Movie_ID	Entertainer_ID
1	2
2	1
1	3
3	1
2	2

Entertainer_ID	Name	Agent_ID
1	Emil	1
2	Rita	1
3	Dean	3

ER model representation



ER DIAGRAM FOR STUDENT ENROLLMENT SYSTEM



- Functional Dependency: Describes the relationship between attributes in a relation. For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A □ B), if each value of A is associated with exactly one value of B. (A and B may each consist of one or more attributes.)
- Determinant: Refers to the attribute, or group of attributes, on the left-hand side of the arrow of a functional dependency.
- Full Functional Dependency: Indicates that if A and B are attributes of a relation, B is fully functionally dependent on A if B is functionally dependent on A, but not on any proper subset of A.
- Partial Dependency: A functional dependency A□B is a **partial dependency** if there is some attribute that can be removed from A and yet the dependency still holds.

STUD_ID	PROJ_ID	PROJ_DURATION(in weeks)	STUD_FNAME	STUD_LNAME
1	1	1	Emil	Jones
1	2	4	Emil	Jones
2	3	4	Jon	Doe
3	4	1	Jai	Singh
3	1	1	Jai	Singh
4	2	4	Emil	Doe

Examples:

Functional Dependency:
{STUD_ID,PROJ_ID}□ {PROJ_DURATION(in weeks), STUD_FNAME , STUD_LNAME}

Full Functional Dependency:
{STUD_ID,PROJ_ID}□ {PROJ_DURATION(in weeks), STUD_FNAME , STUD_LNAME}

Partial Dependency:
{STUD_ID,PROJ_ID}□ {PROJ_DURATION(in weeks)}

KEYS:

- Super key
- Candidate key
- Primary key
- Alternate key
- Foreign key

Table name: STUDENT

STU_NUM	STU_LNAME	STU_FNAME	STU_INIT	STU_DOB	STU_HRS	STU_CLASS
321452	Bowser	William	C	12-Feb-1975	42	So
324257	Smithson	Anne	K	15-Nov-1981	81	Jr
324258	Brewer	Juliette		23-Aug-1969	36	So
324269	Oblonski	Walter	H	16-Sep-1976	66	Jr
324273	Smith	John	D	30-Dec-1958	102	Sr
324274	Katinga	Raphael	P	21-Oct-1979	114	Sr
324291	Robertson	Gerald	T	08-Apr-1973	120	Sr
324299	Smith	John	B	30-Nov-1986	15	Fr

- Super key
  - a super key is any key that uniquely identifies each row. In short, the super key functionally determines all of a row's attributes
  - a super key may contain additional attributes that are not necessary for unique identification, and we are interested in identifying super keys that contain only the minimum number of attributes necessary for unique identification.
  - E.g., STU_NUM

    STU_NUM, STU_LNAME

    STU_NUM, STU_LNAME, STU_INIT
- Candidate key
  - A super key such that no proper subset is a super key within the relation.
  - There may be several candidate keys for a relation. When a key consists of more than one attribute, we call it a **composite key**.
  - A **candidate key** can be described as a super key without unnecessary attributes, that is, a minimal super key. Using this distinction, note that the composite key

    STU_NUM, STU_LNAME

    Is a super key, but it is not a candidate key because STU_NUM by itself is a candidate key.

    The combination STU_LNAME, STU_FNAME, STU_INIT, STU_PHONE

    might also be a candidate key, as long as you discount the possibility that two students share the same last name, first name, initial, and phone number.

- Primary key:
  - The candidate key that is selected to identify tuples uniquely within the relation.
  - Within a table, each primary key value must be unique to ensure that each row is uniquely identified by the primary key. In that case, the table is said to exhibit **entity integrity**. To maintain entity integrity, a **null** (that is, no data entry at all) is not permitted in the primary key.

- Alternate key:
  - The candidate keys that are not selected to be the primary key are called **alternate keys**

- Foreign key:
  - An attribute, or set of attributes, within one relation that matches the candidate key of some other (possibly the same) relation.
  - When an attribute appears in more than one relation, its appearance usually represents a relationship between tuples of the two relations.
  - A **foreign key** (**FK**) is an attribute whose values match the primary key values in the related table.

Normalization:

- **Normalization** is a process for evaluating and correcting table structures to minimize data redundancies, thereby reducing the likelihood of data anomalies.
- The purpose of normalization is to identify a suitable set of relations that support the data requirements of an enterprise. The characteristics of a suitable set of relations include thefollowing:
  - the *minimal* number of attributes necessary to support the data requirements of the enterprise;
  - attributes with a close logical relationship (described as functional dependency) are found in the same relation;
  - *minimal* redundancy with each attribute represented only once with the important exception of attributes that form all or part of foreign keys which are essential for the joining of related relations.

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

- Updation anomalies: If we want to update branch address of a branch then we have to update multiple tuples in StaffBranch relation. This won't be necessary in Normalized form where we have to update only the Branch relation.
- Insertion anomalies: To insert multiple new staff in StaffBranch relation we have to correctly enter bAddress at all the tuples. However in normalized form we only have to write the branchNo correctly.

Also to insert a new branch in the denormalized StaffBranch relation is difficult as all the columns except branchNo and bAddress have to be null. This is not the case in normalized form where this can be easily achieved with a new tuple in branch relation.

- Deletion anomalies: Deleting all the tuples associated to a particular branch leads to loss of branch details in denormalized form.

Normal Forms

- First normal form(1NF):
  - There are no repeating groups in the table. In other words, each row/column intersection contains one and only one value, not a set of values.
  - All attributes are dependent on the primary key.
  - relation is in first normal form if every attribute in that relation is **single valued attribute**.

Denormalized STUDENT table

STUD_ID	STUD_FNAME	STUD_LNAME	PHONE
1	Emil	Jones	32445, 78554
2	Jon	Doe	894756
3	Jai	Singh	34223. 56778

STUDENT table in 1NF

STUD_ID	STUD_FNAME	STUD_LNAME	PHONE
1	Emil	Jones	32445
1	Emil	Jones	78554
2	Jon	Doe	894756
3	Jai	Singh	34223
3	Jai	Singh	56778

- Second normal form(2NF):
  - Table is in 1NF.
  - Second Normal Form applies to relations with composite keys, that is, relations with a primary key composed of two or more attributes.
  - It includes no partial dependencies; that is, no attribute is dependent on only a portion of the primary key.

In other words, , no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

STUD_ID	PROJ_ID	PROJ_DURATION(in weeks)	STUD_FNAME	STUD_LNAME
1	1	1	Emil	Jones
1	2	4	Emil	Jones
2	3	4	Jon	Doe
3	4	1	Jai	Singh
3	1	1	Jai	Singh
4	2	4	Emil	Doe

(STUD_ID,PROJ_ID) together is the candidate key.
PROJ_DURATION is not a part of candidate key; hence a non-prime attribute.
But PROJ_DURATION is dependent on PROJ_ID, a portion of candidate key.
This scenario is called Partial Dependency.

Student

STUD_ID	STUD_FNAME	STUD_LNAME
1	Emil	Jones
2	Jon	Doe
3	Jai	Singh
4	Emil	Doe

Project

PROJ_ID	PROJ_DURATION(in weeks)
1	1
2	4
3	4
4	1
1	1
2	4

Assignment

STUD_ID	PROJ_ID
1	1
1	2
2	3
3	4
3	1
4	2

- Third normal form (3NF): A table is in **third normal form** (**3NF**) when:
  - It is in 2NF.
  - It contains no transitive dependencies. a **transitive dependency** is a dependency of one nonprime attribute on another nonprime attribute. The problem with transitive dependencies is that they still yield data anomalies.

STUD_ID	STUD_FNAME	STUD_LNAME	CITY	COUNTRY
1	Emil	Jones	NY	USA
2	Jon	Doe	Washington	USA
3	Jai	Singh	Delhi	India
4	Emil	Doe	Washington	USA

STUD_ID is a candidate key and the primary key.
CITY and COUNTRY are non-prime Attributes.
But COUNTRY is dependent on CITY.
This is transitive dependency.

Student

STUD_ID	STUD_FNAME	STUD_LNAME	CITY
1	Emil	Jones	NY
2	Jon	Doe	Washington
3	Jai	Singh	Delhi
4	Emil	Doe	Washington

Location

CITY	COUNTRY
NY	USA
Washington	USA
Delhi	India
Washington	USA

- Boyce-Codd Normal Form (BCNF): A table is in BCNF when every determinant in the table is a candidate key.

Student	Teacher	Subject
Jon	P.Naresh	Database
Jon	K.Das	C
Sid	P.Naresh	Database
Sid	R.Prasad	C

Dependencies:{ (student, Teacher) -> subject
(student, subject) -> Teacher
Teacher -> subject}
(student, Teacher) and (student, subject) are candidate keys.
But Teacher is a not a key.

Teacher

Teacher	Subject
P.Naresh	Database
K.Das	C
R.Prasad	C

Student

Student	Teacher
Jon	P.Naresh
Jon	K.Das
Sid	P.Naresh
Sid	R.Prasad

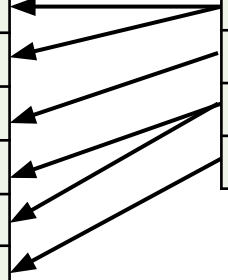
Indexes

- An **index** is an orderly arrangement used to logically access rows in a table.
- Indexes in the relational database environment work like the indexes described in the above point. From a conceptual point of view, an index is composed of an index key and a set of pointers. The **index key** is, in effect, the index's reference point. More formally, an index is an ordered arrangement of keys and pointers. Each key points to the location of the data identified by the key.
- An index can be used to retrieve data more efficiently. But indexes can also be used by a DBMS to retrieve data ordered by a specific attribute or attributes.
- When you define a table's primary key, the DBMS automatically creates a unique index on the primary key column(s) you declared.

STUD_ID as Index key
And pointers to Student table rows.

Student

STUD_ID	PROJ_ID	PROJ_DURATION(in weeks)	STUD_FNAME	STUD_LNAME
1	1	1	Emil	Jones
1	2	4	Emil	Jones
2	3	4	Jon	Doe
3	4	1	Jai	Singh
3	1	1	Jai	Singh
4	2	4	Emil	Doe



Index_key	Pointers
1	1,2
2	3
3	4,5
4	6

THE END