

Video chapters

- **Chapter-1 (Basics):** Data & information, Database System vs File System, Views of Data Base, Data Independence, Instances & Schema, OLAP Vs OLTP, Types of Data Base, DBA, Architecture.
- **Chapter-2 (ER Diagram):** Entity, Attributes, Relationship, Degree of a Relationship, Mapping, Weak Entity set, Conversion from ER Diagram to Relational Model, Generalization, Specification, Aggregation.
- **Chapter-3 (RDBMS & Functional Dependency):** Basics & Properties, Update Anomalies, Purpose of Normalization, Functional Dependency, Closure Set of Attributes, Armstrong's axioms, Equivalence of two FD, Canonical cover, Keys.
- **Chapter-4 (Normalization):** 1NF, 2NF, 3NF, BCNF, Multivalued Dependency, 4NF, Lossy-Lossless Decomposition, 5NF, Dependency Preserving Decomposition.
- **Chapter-5 (Indexing):** Overview of indexing, Primary indexing, Clustered indexing and Secondary Indexing, B-Tree.
- **Chapter-6 (Relational Algebra) :** Query Language, Select, Project, Union, Set Difference, Cross Product, Rename Operator, Additional or Derived Operators.
- **Chapter-7(SQL) :** Introduction to SQL, Classification, DDL Commands, Select, Where, Set Operations, Cartesian Product, Natural Join, Outer Join, Rename, Aggregate Functions, Ordering, String, Group, having, Trigger, embedded, dynamic SQL.
- **Chapter-8 (Relational Calculus) :** Overview, Tuple Relation Calculus, Domain Relation Calculus.
- **Chapter-9 (Transaction) :** What is Transaction, ACID Properties, Transaction Sates, Schedule, Conflict Serializability, View Serializability, Recoverability, Cascade lessness, Strict Schedule.
- **Chapter-10 (Recovery & Concurrency Control) :** Log Based Recovery, Shadow Paging, Data Fragmentation, TIME STAMP ORDERING PROTOCOL, THOMAS WRITE RULE, 2 phase locking, Basic 2pl, Conservative 2pl, Rigorous 2pl, Strict 2pl, Validation based protocol Multiple Granularity.

Syllabus

- **UNIT-1 : INTRODUCTION** Overview, Database System vs File System, Database System Concept and Architecture, Data Model Schema and Instances, Data Independence and Database Language and Interfaces, Data Definitions Language, DML, Overall Database Structure. **Data Modeling Using the Entity Relationship Model:** ER Model Concepts, Notation for ER Diagram, Mapping Constraints, Keys, Concepts of Super Key, Candidate Key, Primary Key, Generalization, Aggregation, Reduction of an ER Diagrams to Tables, Extended ER Model, Relationship of Higher Degree.
- **UNIT-2 : RELATIONAL DATA MODEL** Relational Data Model Concepts, Integrity Constraints, Entity Integrity, Referential Integrity, Keys Constraints, Domain Constraints, Relational Algebra, Relational Calculus, Tuple and Domain Calculus. **Introduction on SQL:** Characteristics of SQL, Advantage of SQL. SQL Data Type and Literals. Types of SQL Commands. SQL Operators and Their Procedure. Tables, Views and Indexes. Queries and Sub Queries. Aggregate Functions. Insert, Update and Delete Operations, Joins, Unions, Intersection, Minus, Cursors, Triggers, Procedures in SQL/PL SQL.
- **UNIT-3 : DATA BASE DESIGN & NORMALIZATION** Functional dependencies, normal forms, first, second, 3 third normal forms, BCNF, inclusion dependence, loss less join decompositions, normalization using FD, MVD, and JDs, alternative approaches to database design.
- **UNIT-4 : TRANSACTION PROCESSING CONCEPT** Transaction System, Testing of Serializability, Serializability of Schedules, Conflict & View Serializable Schedule, Recoverability, Recovery from Transaction Failures, Log Based Recovery, Checkpoints, Deadlock Handling. Distributed Database: Distributed Data Storage, Concurrency Control, Directory System.
- **UNIT-5 : CONCURRENCY CONTROL TECHNIQUES** Concurrency Control, Locking Techniques for Concurrency Control, Time Stamping Protocols for Concurrency Control, Validation Based Protocol, Multiple Granularity, Multi Version Schemes, Recovery with Concurrent Transaction, Case Study of Oracle.

What is Data ?

- Data are characteristics or attributes, often numerical, collected through observation and can be qualitative(descriptive) or quantitative(numerical). Any facts and figures about an entity is called as Data.
- Data serves a crucial role in various sectors by facilitating analysis, supporting decision-making processes, and being fundamental to research activities.



What is Information ?

- Data becomes information when analyzed and placed in context, providing a basis for understanding, decision-making, and further analysis. Processed Data is called information.



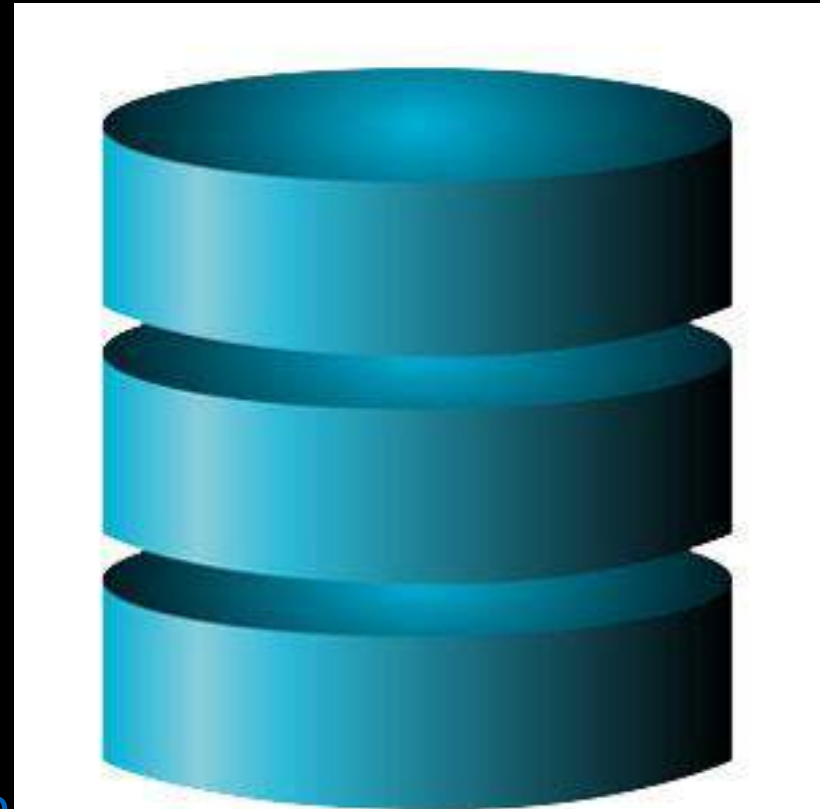
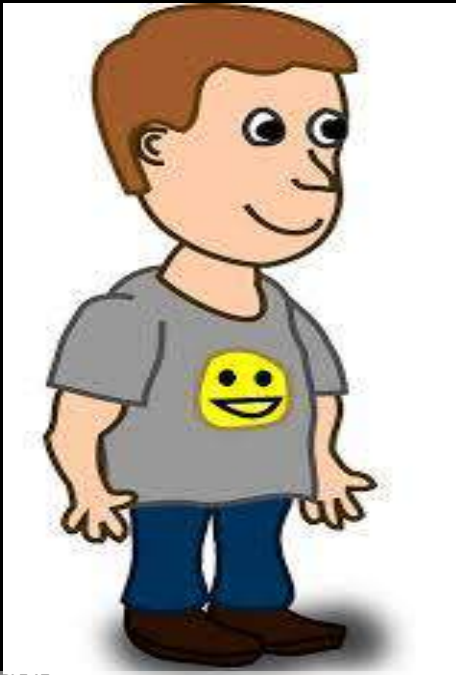
What is Data Base?

- A database is a structured collection of data, facilitating easy access, management, and updates., generally stored and accessed electronically from a computer system.



What is Data Base Management System?

- A DBMS is software facilitating efficient data storage, retrieval, and management in databases.
- Ensures data safety and integrity, while offering accessibility and concurrency control.
- Supports functions like data querying, reporting, and analytics for informed decision-making.

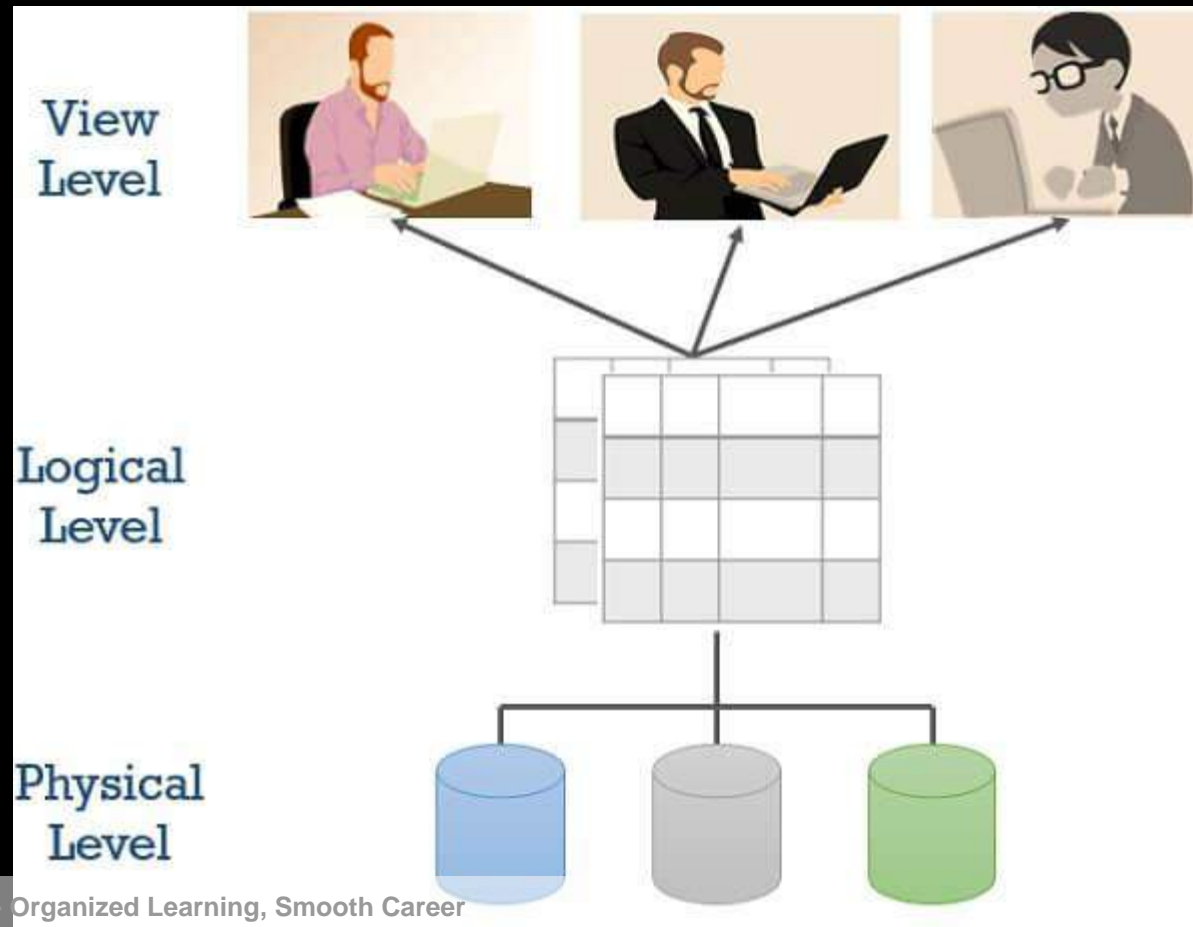


BLE AT:

Aspect	File System	Database Management System
Data Access	Slower data retrieval due to unstructured querying capabilities.	Structured querying capabilities allow for quicker data access.
Data Isolation	Challenges in correlating data across separate files leading to data isolation.	Facilitates data integration, reducing data isolation issues.
Data Integrity	Risk of inadvertent data alterations or deletions creating integrity problems.	Features to prevent unauthorized data alterations, maintaining integrity.
Atomicity Problem	Potential for data inconsistency due to incomplete operations, leading to atomicity problems.	Supports transaction properties like atomicity, ensuring operations are either completed fully or not at all.
Concurrent Access Anomalies	Conflicts and inconsistencies from simultaneous data access/modifications, causing concurrent access anomalies.	Advanced concurrency controls to manage multiple users accessing the database simultaneously, reducing anomalies.

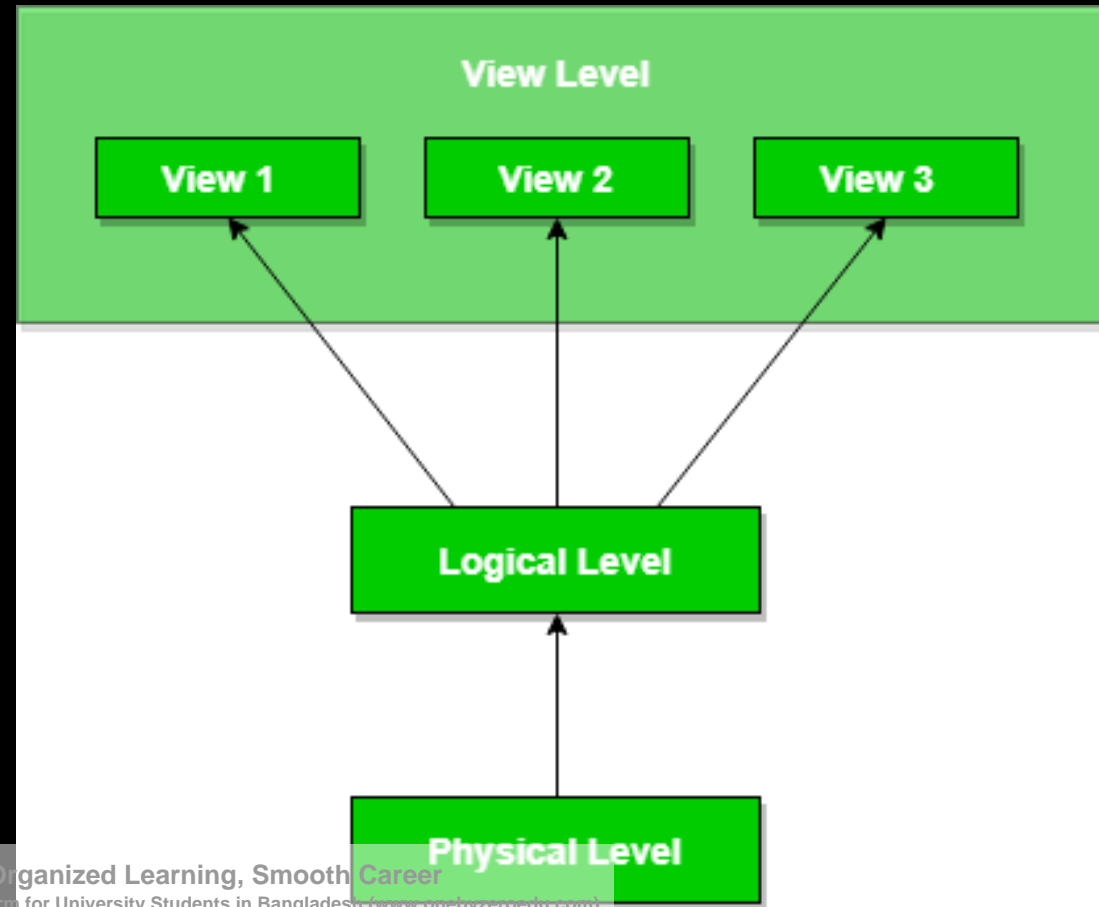
View of Data Base (Data Abstraction)

- Physical Level
- Logical Level/ Conceptual Level
- View Level



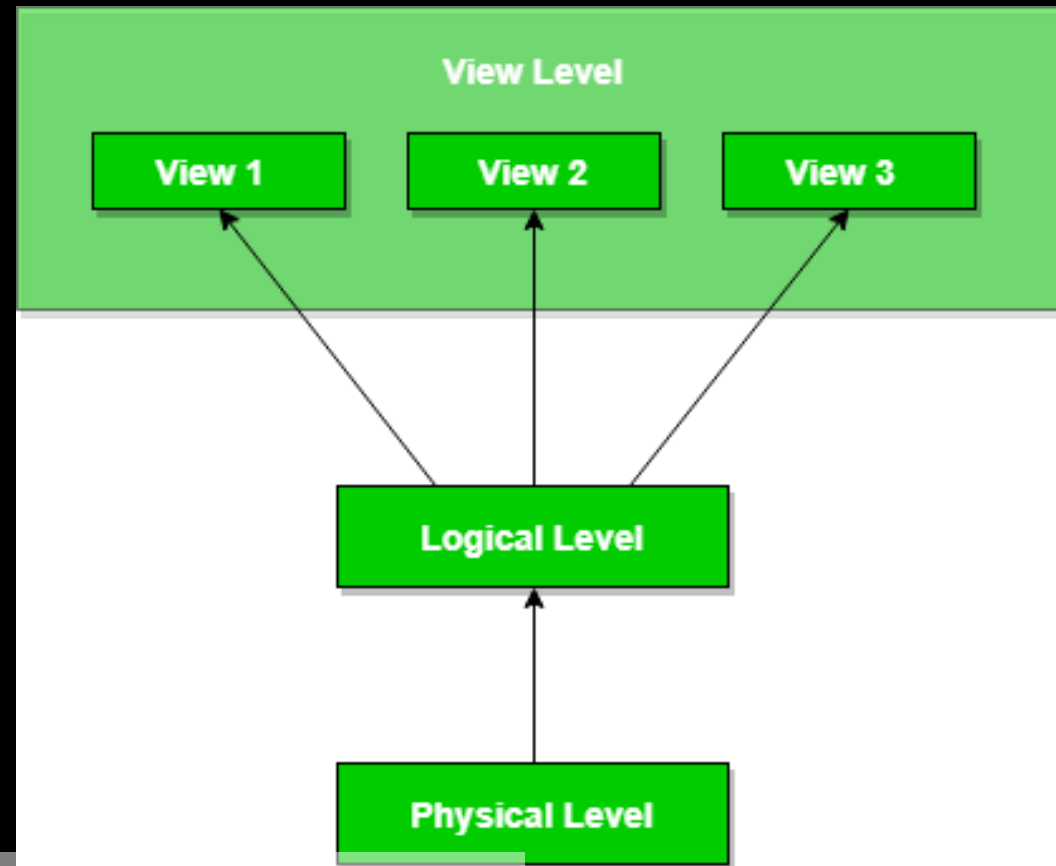
View of Data Base (Data Abstraction)

- **Physical Level**: The internal schema details data storage and access on hardware, featuring the lowest level of data abstraction with complex structures, predominantly managed by the database administrator.



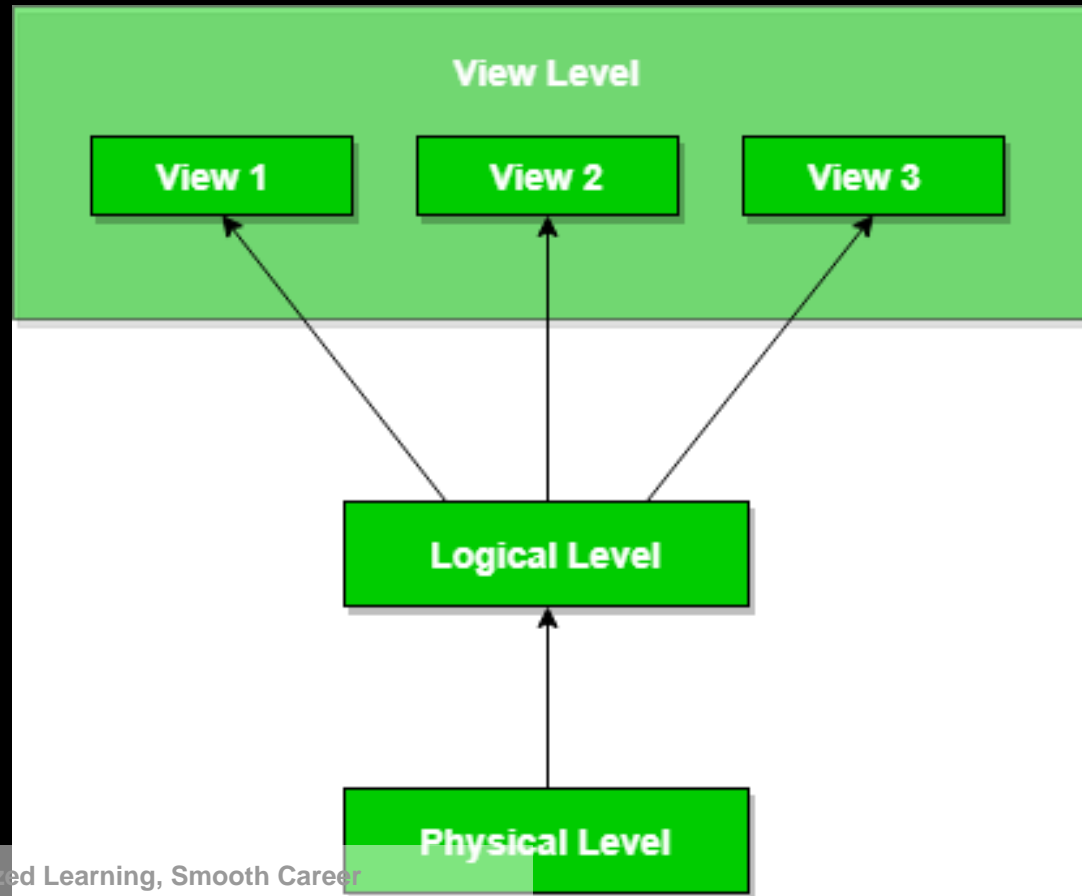
View of Data Base

- Logical Level/ Conceptual Level: Above the physical level, this level showcases data as entity sets and their relationships, detailing the types and connections between stored data in the database.



View of Data Base

- **View Level**: This is the pinnacle of data abstraction, displaying only a portion of the entire database focusing on user-interest areas. It can represent multiple views of the same data, allowing users to access information through various applications from the database.



Data independence

- Data independence is defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
- Types of data independence :
 - **Physical data independence**: a. Physical data independence is the ability to modify internal schema without changing the conceptual schema. b. Modification at the physical level is occasionally necessary in order to improve performance. c. It refers to the immunity of the conceptual schema to change in the internal schema. d. Examples of physical data independence are reorganizations of files, adding a new access path or modifying indexes, etc.
 - **Logical data independence**: a. Logical data independence is the ability to modify the conceptual schema without having to change the external schemas or application programs. b. It refers to the immunity of the external model to changes in the conceptual model. c. Examples of logical data independence are addition/removal of entities.

Instance and Schemas

- **Instance of the Database:**
 - The collection of information stored in the database at a specific moment is known as an instance of the database. It is a snapshot of the database that contains live data at that moment, showing the current state of all records and transactions.
- **Database Schema:**
 - The database schema refers to the overall design of the database, illustrating the logical structure and organization of data. It defines how data is organized and how relationships between data are handled, essentially serving as the blueprint for how the database is constructed.

Aspect	OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
Primary Function	Designed for complex data analysis and reporting.	Handles daily transactional data processing.
Database Design	Star or snowflake schema, optimizing for read operations.	Normalized schema, optimizing for write operations.
Query Complexity	Complex queries involving aggregations and computations across multiple dimensions.	Simple and standard queries focusing on CRUD operations (Create, Read, Update, Delete).
Data Volume	Deals with large volumes of data for historical analysis.	Processes a high number of small transactions.
Response Time	Slower response time due to complex queries.	Fast response time to support high transaction rates.


Types of Data Base

- **Commercial Database**: Predominantly used in the business sector to handle large volumes of transactions and customer data. A CRM system like Salesforce which handles large volumes of customer data and transactions.
- **Multimedia Database**: Stores data types such as images, audio, and video files, facilitating the management and retrieval of multimedia content. A digital asset management system like Adobe Experience Manager that facilitates the storage and retrieval of multimedia content.
- **Deductive Database**: Utilizes logic programming to derive information from data stored in a database, allowing for more complex and analytical queries. A database using Datalog (a query language) which allows for complex logical queries and information derivation.

- **Temporal Database**: Keeps track of changing data over time, allowing for queries concerning time-based data. A historical trading database in the financial sector which keeps track of stock prices over time.
- **Geological Information System (GIS)**: Stores, organizes, and analyzes geographical data, aiding in spatial analysis and mapping projects. A system like ArcGIS which enables the storage, analysis, and visualization of geographical data.

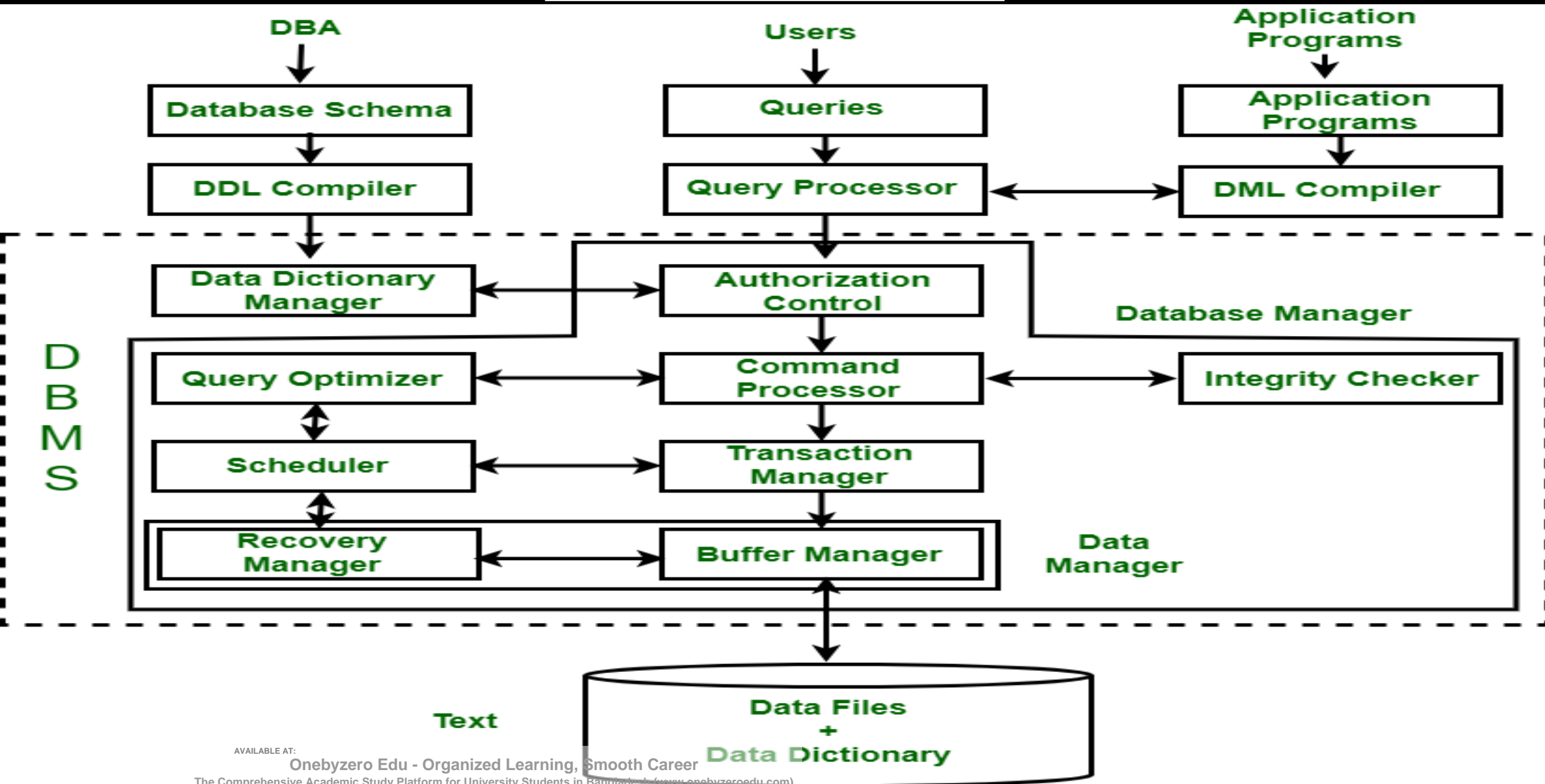
DBA(Database Administrator)

Database administrators hold authority over data and the programs facilitating data access. Their roles/functions are:

- **Schema Definition**: DBA outlines the original database schema. Achieved through writing definitions translated to permanent labels in the data dictionary by the DDL compiler.
 - **Storage Structure and Access Method Definition**: Responsible for forming appropriate storage structures and access methods. Achieved through writing definitions translated by the data storage and definition language compiler.
 - **Schema and Physical Organization Modification**: Involves altering the database schema or physical storage organization. Changes are implemented by writing definitions that modify the relevant internal system tables.
 - **Granting of Authorization for Data Access**: DBA grants varied types of data access authorization to different database users.
 - **Integrity Constraint Specification**: DBA implements and maintains integrity constraints to ensure data accuracy and consistency.
- 



DBMS Architecture



AVAILABLE AT:

Onebyzero Edu - Organized Learning, Smooth Career
The Comprehensive Academic Study Platform for University Students in Bangladesh (www.onebyzeroedu.com)

1. Query Processor: This is the component of a DBMS that interprets and executes user queries. It comprises several sub-components including:

1. **DML Compiler:** Processes Data Manipulation Language (DML) statements into low-level instructions that can be executed.
2. **DDL Interpreter:** Processes Data Definition Language (DDL) statements into metadata tables.
3. **Embedded DML Pre-compiler:** Translates DML statements embedded in application programs into procedural calls.
4. **Query Optimizer:** Determines the most efficient way to execute a query by evaluating different query plans.

2. Storage Manager: Also known as the Database Control System, it is responsible for managing the data stored in the database, ensuring its consistency and integrity. It includes the following sub-components:

1. **Authorization Manager:** Manages access controls and privileges.
2. **Integrity Manager:** Ensures that data modifications adhere to integrity constraints.
3. **Transaction Manager:** Manages concurrent access to the database and maintains database consistency during transactions.
4. **File Manager:** Manages file space and data structures representing information in the database.
5. **Buffer Manager:** Manages data cache and data transfer between main memory and secondary storage.

3. Disk Storage: Represents the storage aspect of a DBMS, encompassing the following components:

1. **Data Files:** Files where the actual data is stored.
2. **Data Dictionary:** Repository containing information about the structure and characteristics of database objects.
3. **Indices:** Data structures that facilitate faster data retrieval.

A Database Management System (DBMS) consists of three primary components:

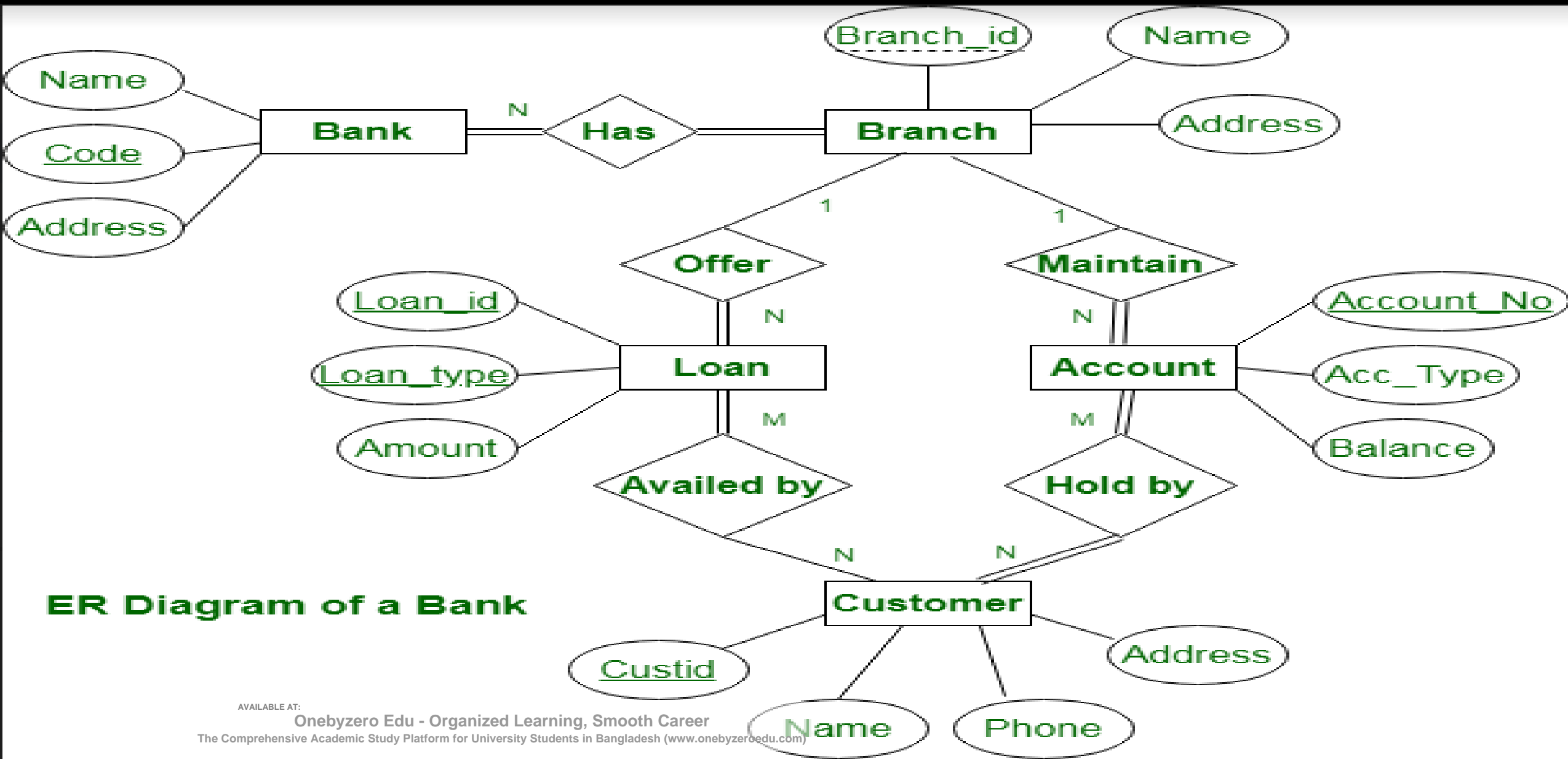
1. **Internal Level**: Concerns the physical storage of data in databases, overseeing data storage on hardware devices, and managing low-level aspects like data compression and indexing.
2. **Conceptual Level**: Represents the logical layout of the database, detailing the schema with tables and attributes and their interrelations. It's independent of specific DBMS implementations, focusing on organizing and connecting data elements.
3. **External Level**: Embodies the user interface of the database, facilitating data access and interaction through user-friendly views and interfaces tailored to various user groups.

ER Diagram

- Developed by Dr. Peter Chen in 1976, this conceptual level method, grounded in real-world perceptions, facilitates diagrammatic data representation, simplifying comprehension for non-technical users.
- The E-R data model, central to database design, encapsulates entities and their attributes within an enterprise schema, serving as a clear, standardized tool for translating real-world enterprise interactions into a conceptual schema.



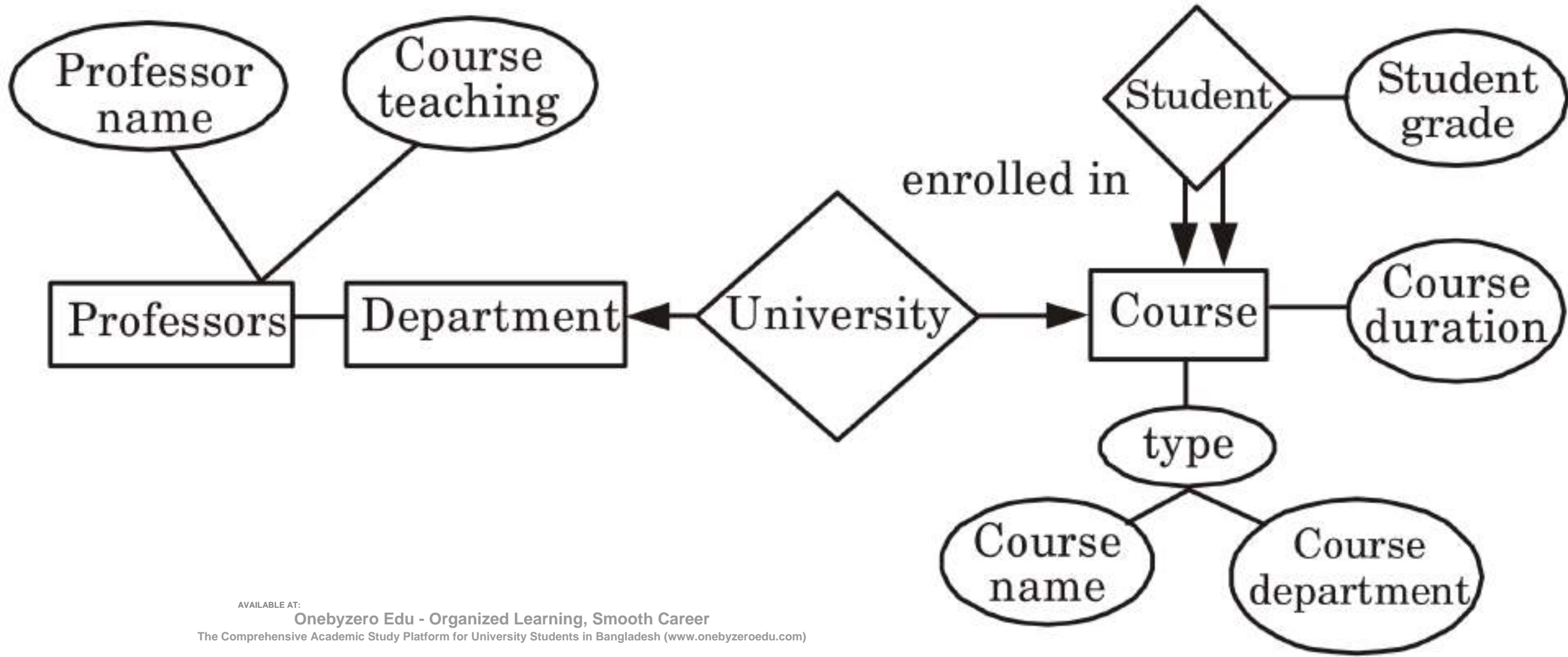
ER diagram for bank



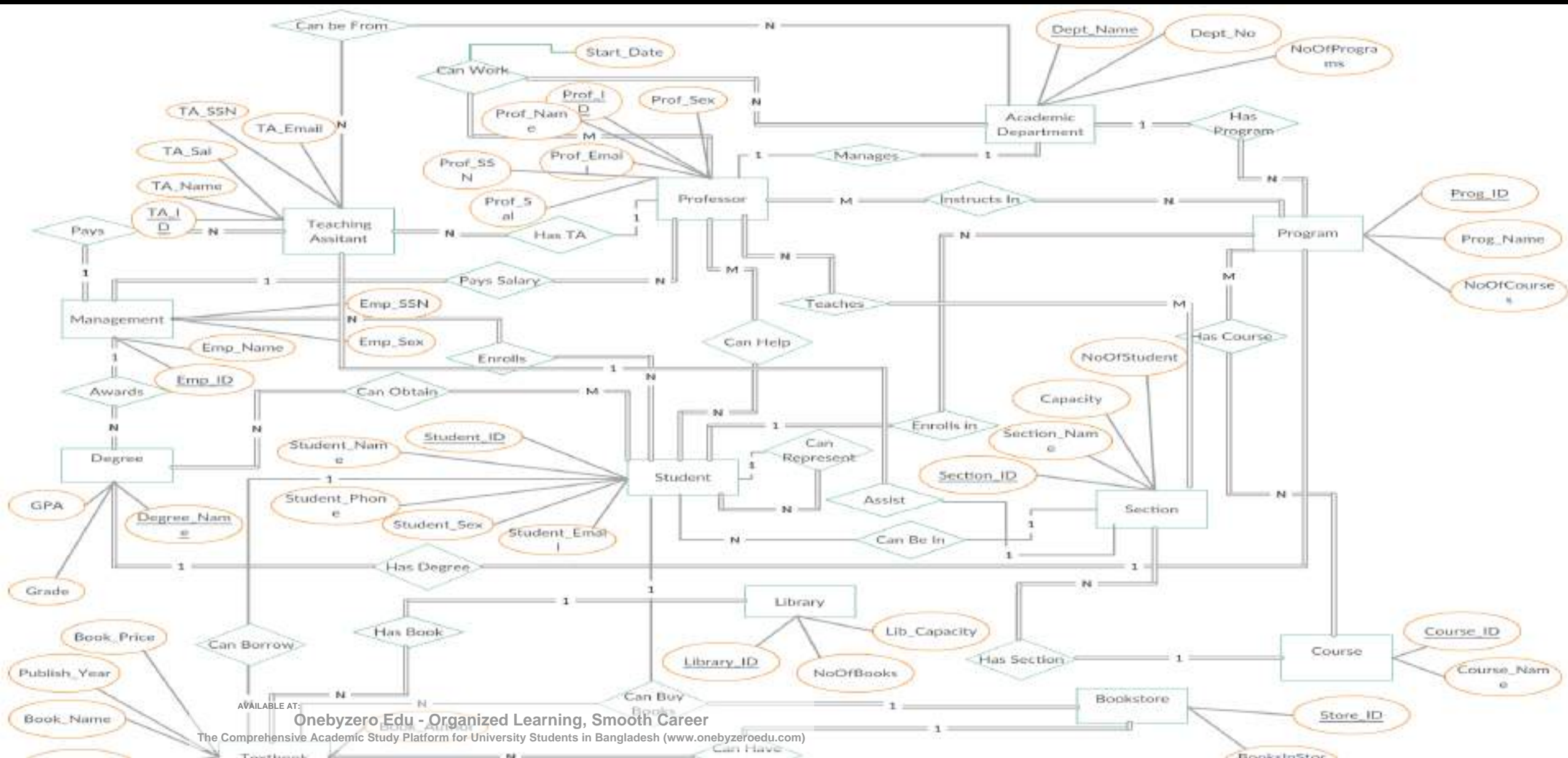
AVAILABLE AT:

Onebyzero Edu - Organized Learning, Smooth Career
The Comprehensive Academic Study Platform for University Students in Bangladesh (www.onebyzeroedu.com)

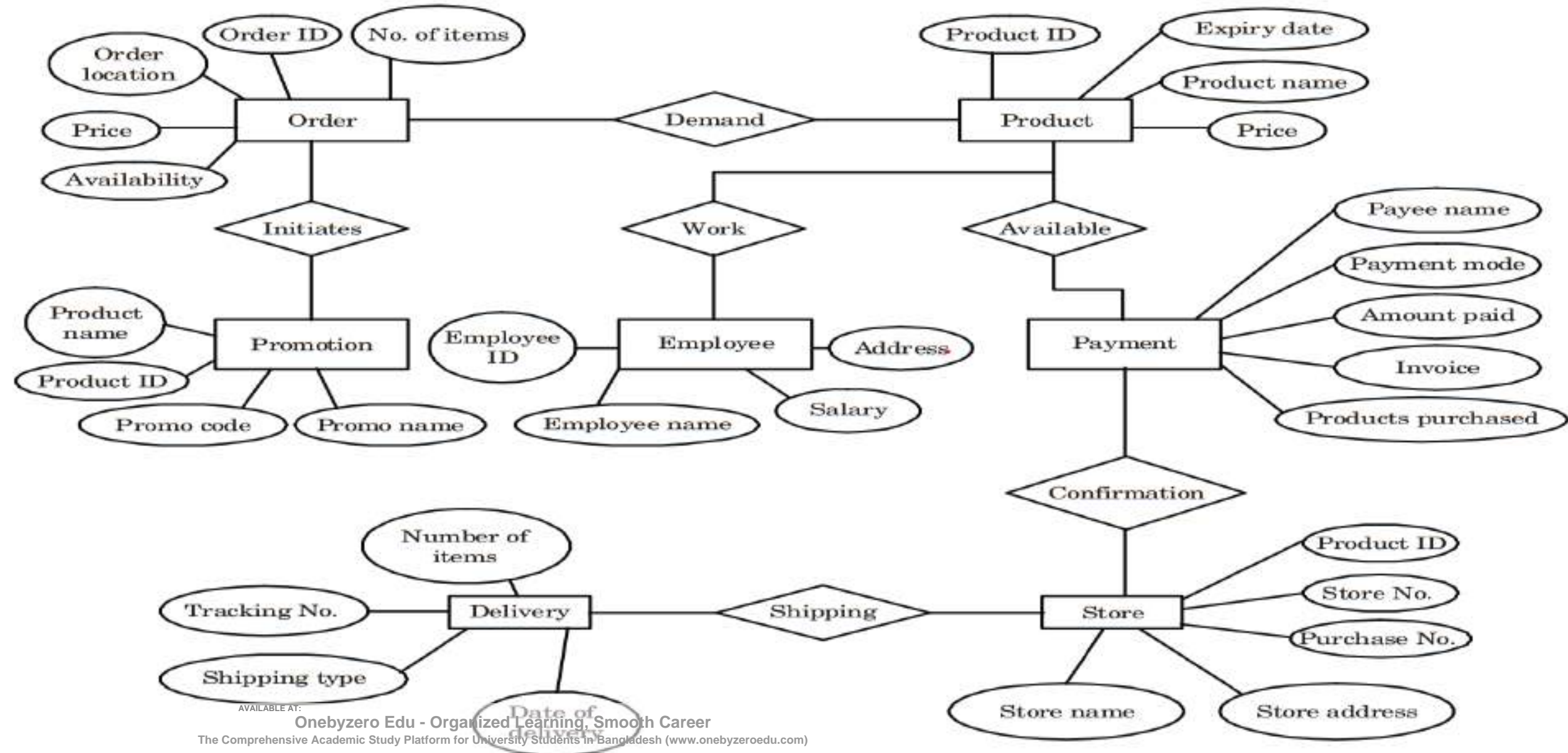
ER diagram for University System



ER diagram for University System



ER diagram for Marketing Company



ENTITY

- An entity is a thing or an object in the real world that is distinguishable from other object based on the values of the attributes it possesses.
- An entity may be **concrete**, such as a person or a book, or it may be **abstract**, such as a course, a course offering, or a flight reservation.

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250



- *Types of Entity*
 - **Tangible** - *Entities which physically exist in real world. E.g. - Car, Pen, locker*
 - **Intangible** - *Entities which exist logically. E.g. – Account, video.*



- In ER diagram we cannot represent an entity, as entity is an instant not schema, and ER diagram is designed to understand schema
- In a relational model entity is represented by a row or a tuple or a record in a table.

Column (attribute) Table (relation)

Row (tuple)

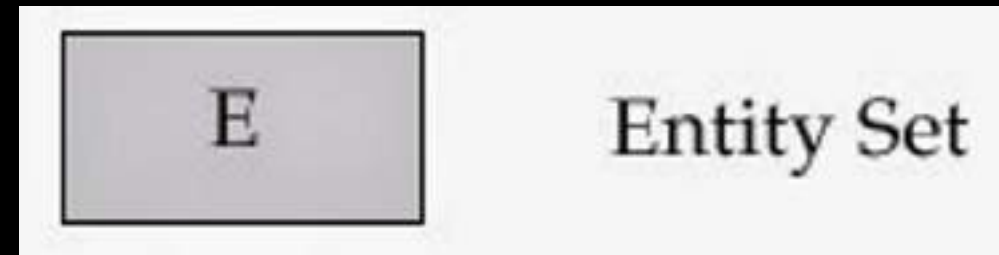
Primary key

Data value

CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

- **ENTITY SET**- Collection of same type of entities that share the same properties or attributes.
- In an ER diagram an entity set is represented by a rectangle
- In a relational model it is represented by a separate table

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250



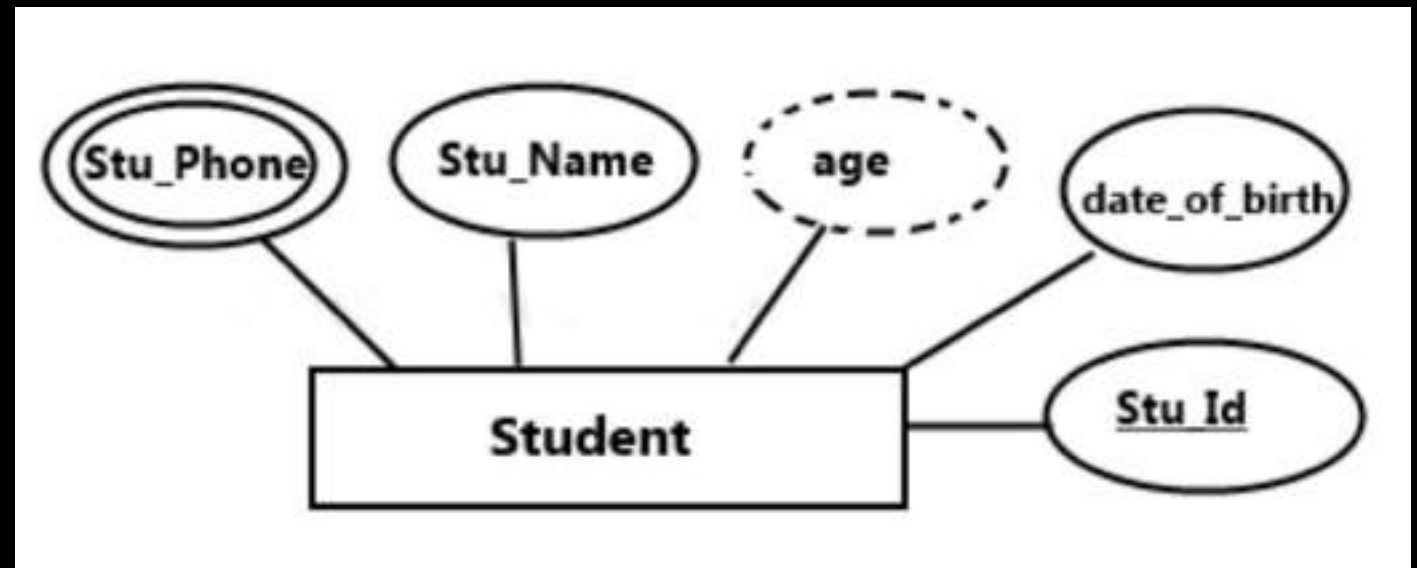
ATTRIBUTES

- Attributes are the units defines and describe properties and characteristics of entities.
- Attributes are the descriptive properties possessed by each member of an entity set.
for each attribute there is a set of permitted values called domain.

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250

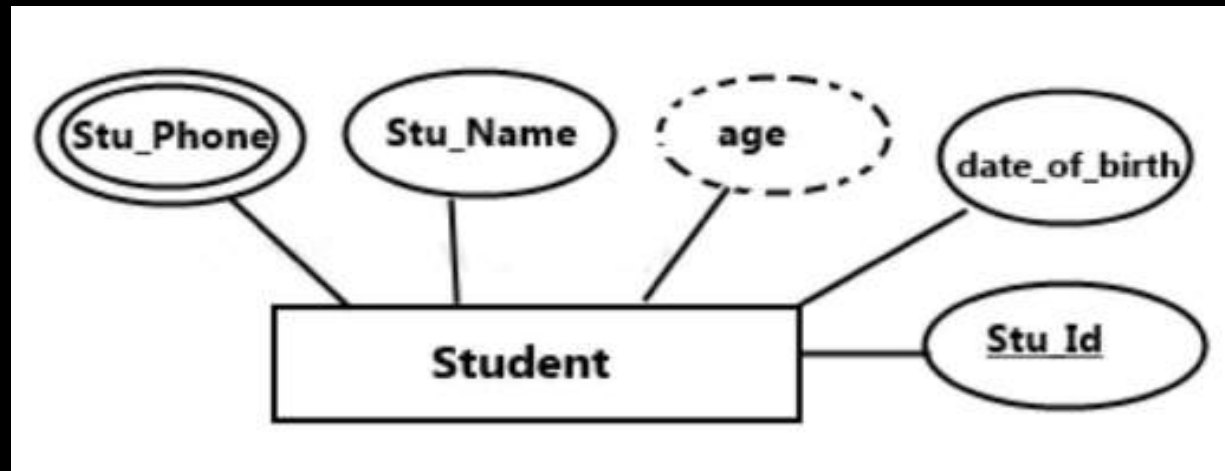
- In an ER diagram attributes are represented by ellipse or oval connected to rectangle.
- While in a relational model they are represented by independent column. e.g. Instructor (ID, name, salary, dept_name)

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250



Types of Attributes

- **Single valued**- Attributes having single value at any instance of time for an entity. E.g. – Aadhar no, dob.
- **Multivalued** - Attributes which can have more than one value for an entity at same time. E.g.
 - Phone no, email, address.
 - A multivalued attribute is represented by a double ellipse in an ER diagram and by an independent table in a relational model.
 - Separate table for each multivalued attribute, by taking mva and pk of main table as fk in new table



Customer			
Customer ID	First Name	Surname	Telephone Number
123	Rabri Devi	Singh	555-861-2025, 192-122-1111
456	Imarti Devi	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	Barfi Devi	Doe	555-808-9633

Customer			
Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

जुगाड़ technology



Customer				
Customer ID	First Name	Surname	Telephone Number1	Telephone Number2
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

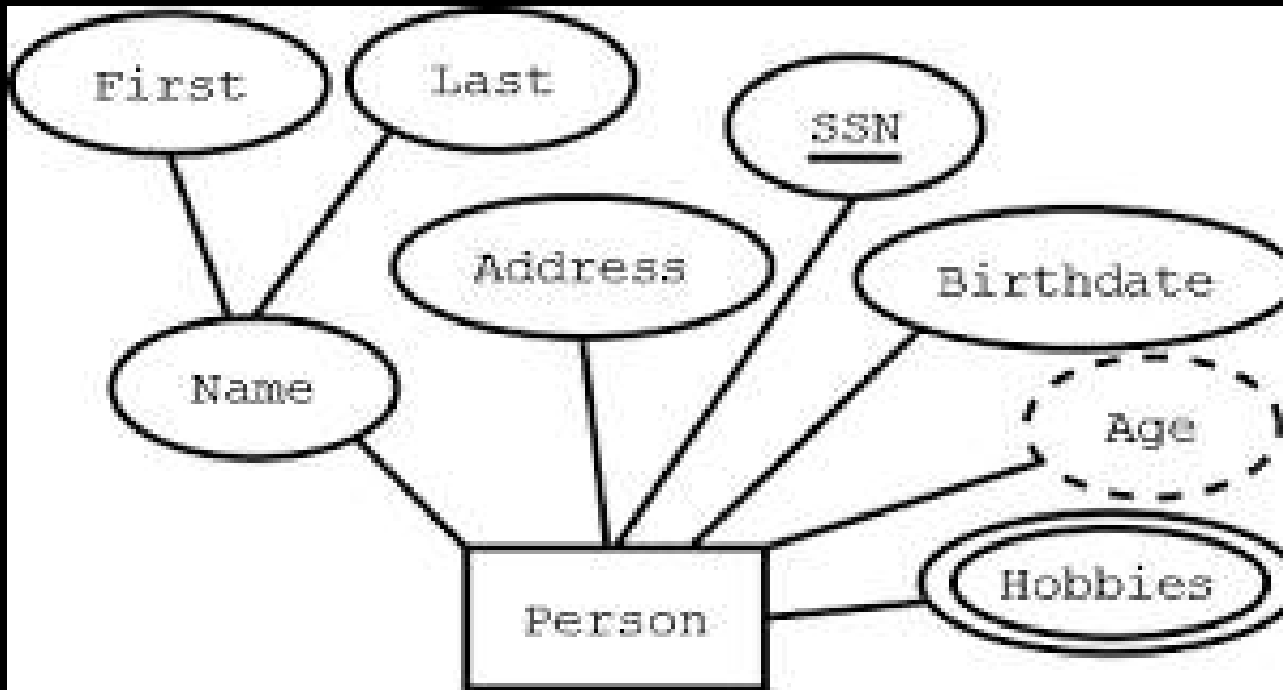
Customer			
Customer ID	First Name	Surname	Telephone Number
123	Rabri Devi	Singh	555-861-2025, 192-122-1111
456	Imarti Devi	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	Barfi Devi	Doe	555-808-9633



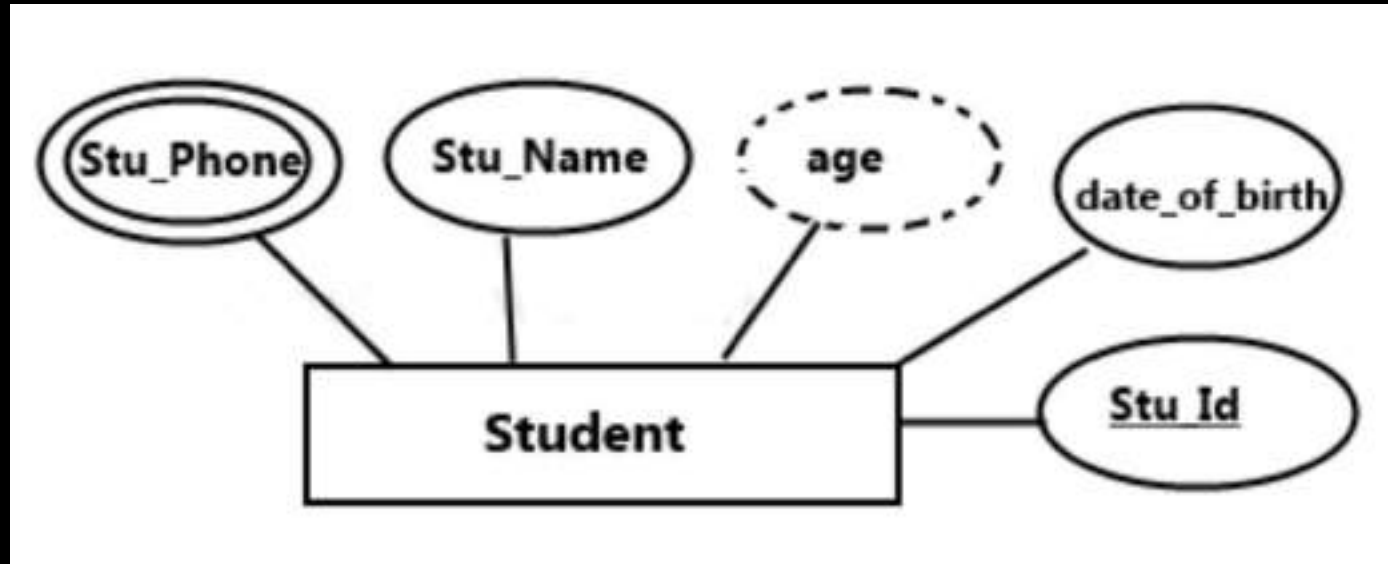
Customer Name		
Customer ID	First Name	Surname
123	Pooja	Singh
456	San	Zhang
789	John	Doe

Customer Phone Number	
Customer ID	Telephone Number
123	555-861-2025
123	192-122-1111
456	(555) 403-1659 Ext. 53
456	182-929-2929
789	555-808-9633

- **Simple** - Attributes which cannot be divided further into sub parts. E.g. Age
- **Composite** - Attributes which can be further divided into sub parts, as simple attributes. A composite attribute is represented by an ellipse connected to an ellipse and in a relational model by a separate column.

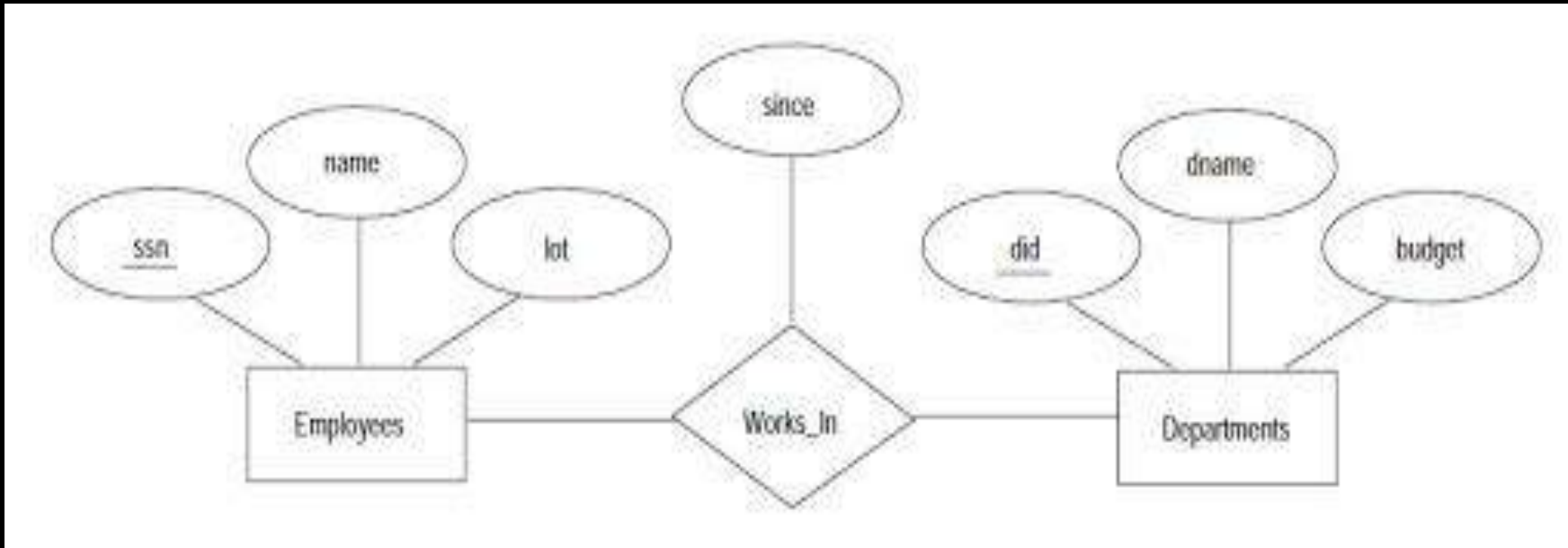


- **Stored** - Main attributes whose value is permanently stored in database. E.g. date_of_birth
- **Derived** -The value of these types of attributes can be derived from values of other Attributes. E.g. - Age attribute can be derived from date_of_birth and Date attribute.



Descriptive attribute - Attribute of relationship is called descriptive attribute.

- An attribute takes a null value when an entity does not have a value for it. The null value may indicate “not applicable” — that is, that the value does not exist for the entity.



Relationship / Association

Relationship Status: 

Family:  Select Relation:  

Featured Friends: 

Create new list - Add an existing list or group

Save Changes

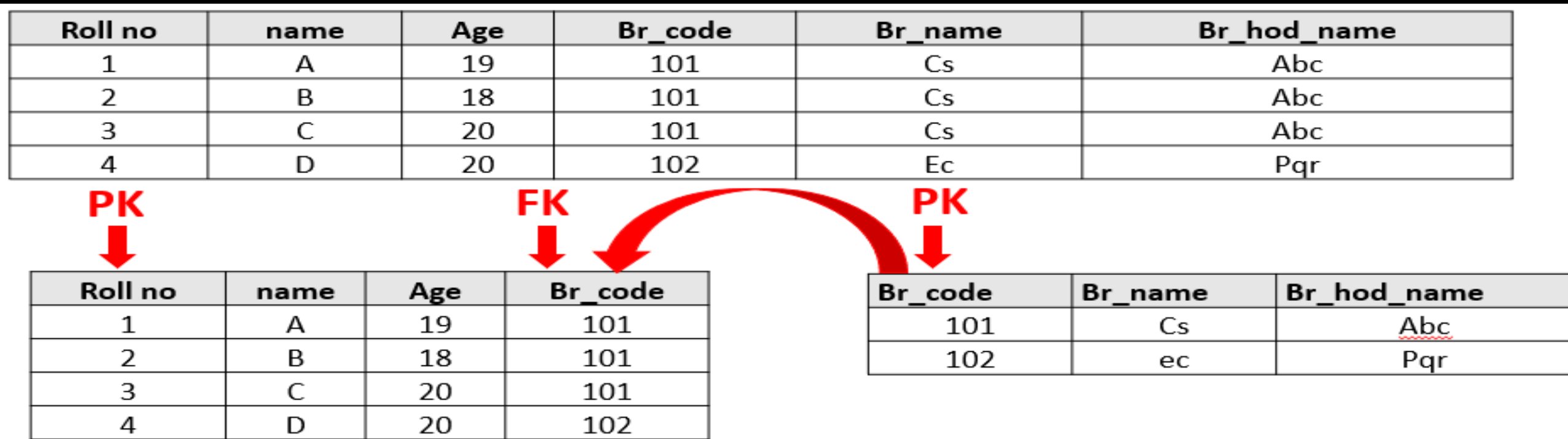
- Single
- In a relationship
- Engaged
- Married
- It's complicated
- In an open relationship
- Widowed
- Separated
- Divorced
- In a civil union
- In a domestic partnership

Relationship / Association

- Is an association between two or more entities of same or different entity set.
- In ER diagram we cannot represent individual relationship as it is an instance or data.



- In an ER diagram it is represented by a diamond, while in relational model sometimes through foreign key and other time by a separate table.



- Every relationship type has three components.
 - Name
 - Degree
 - Structural constraints (cardinalities ratios, participation)

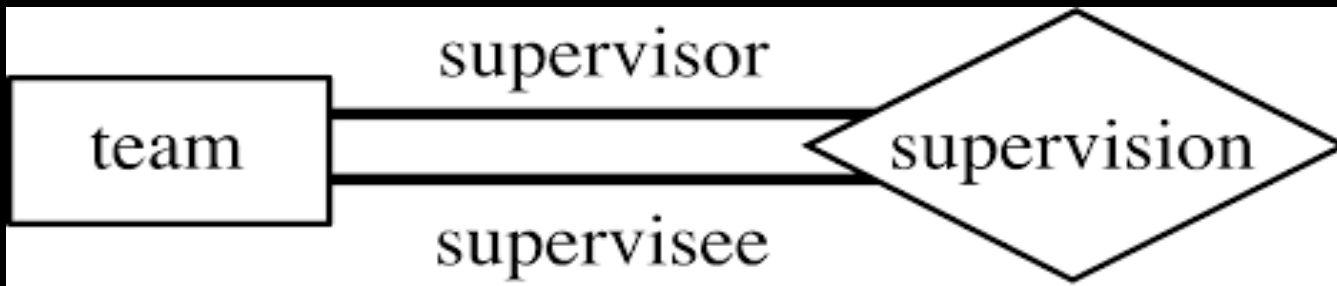
- **NAME**- every relation must have a unique name.

Degree of a relationship/relationship set

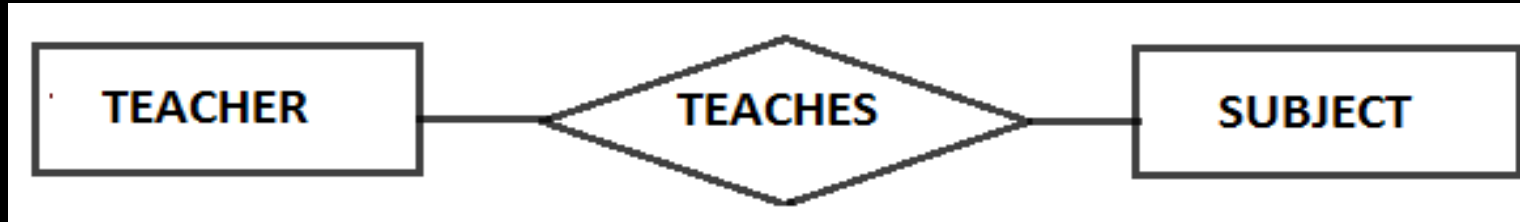
- Means number of entities set(relations/tables) associated(participate) in the relationship set.
- Most of the relationship sets in a data base system are binary.
- Occasionally however relationship sets involve more than two entity sets.
- Logically, we can associate any number of entity set in a relationship called N-ary Relationship.



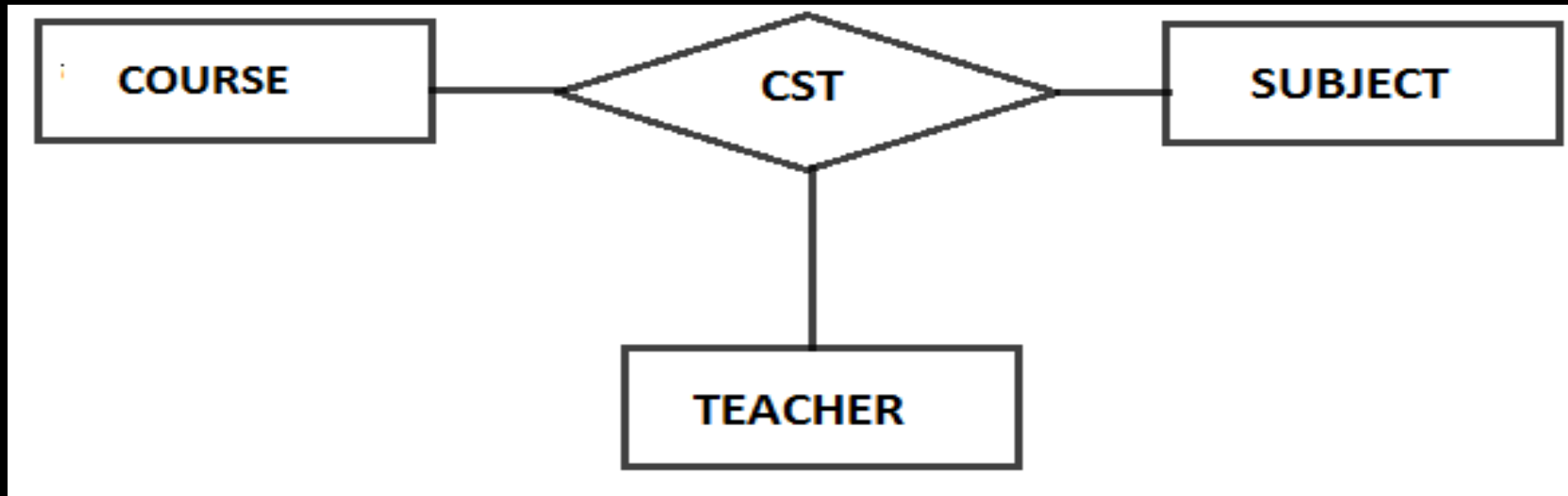
- **Unary Relationship** - One single entity set participate in a Relationship, means two entities of the same entity set are related to each other.
- These are also called as self -referential Relationship set.
- E.g.- A member in a team maybe supervisor of another member in team.



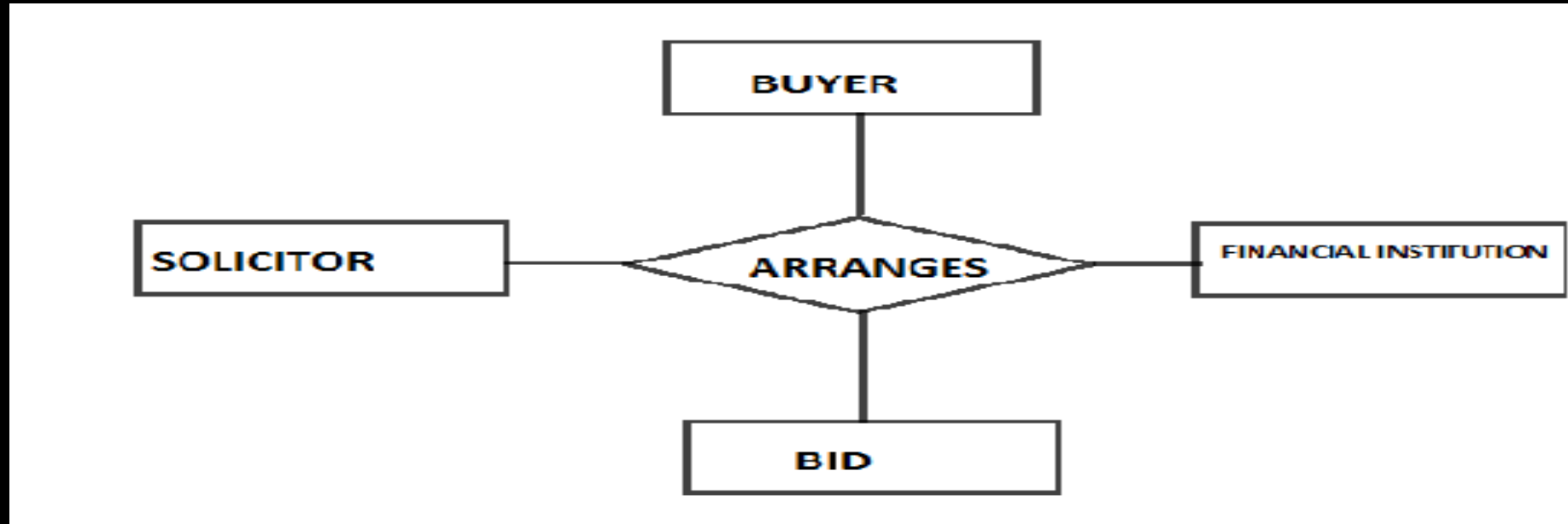
- **Binary Relationship** - Two entity sets participate in a Relationship. It is most common Relationship.



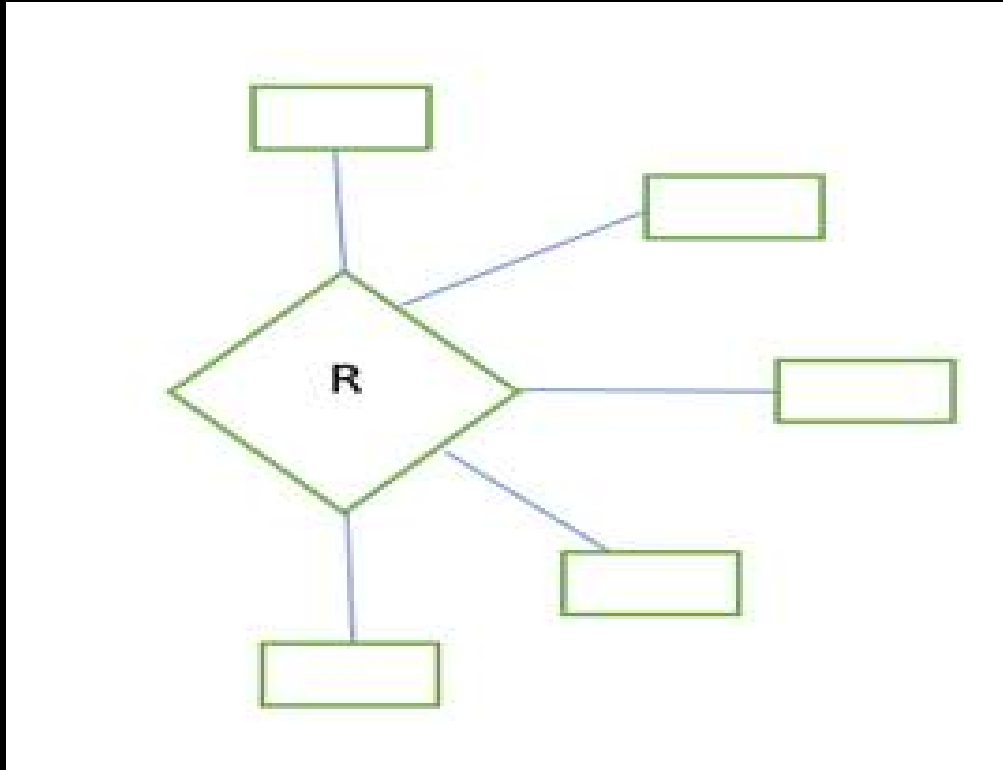
- **Ternary Relationship** - When three entities participate in a Relationship. E.g. The University might need to record which teachers taught which subjects in which courses.



- **Quaternary Relationship** - When four entities participate in a Relationship.



- **N-ary relationship** – where n number of entity set are associated



- But the most common relationships in ER models are ***Binary***.

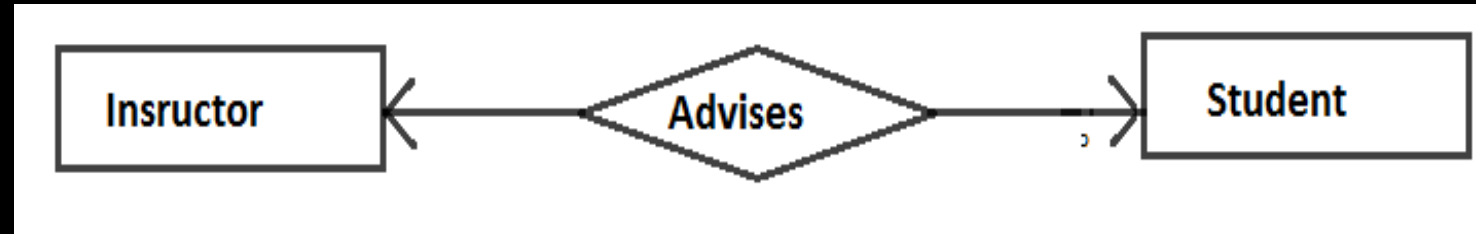
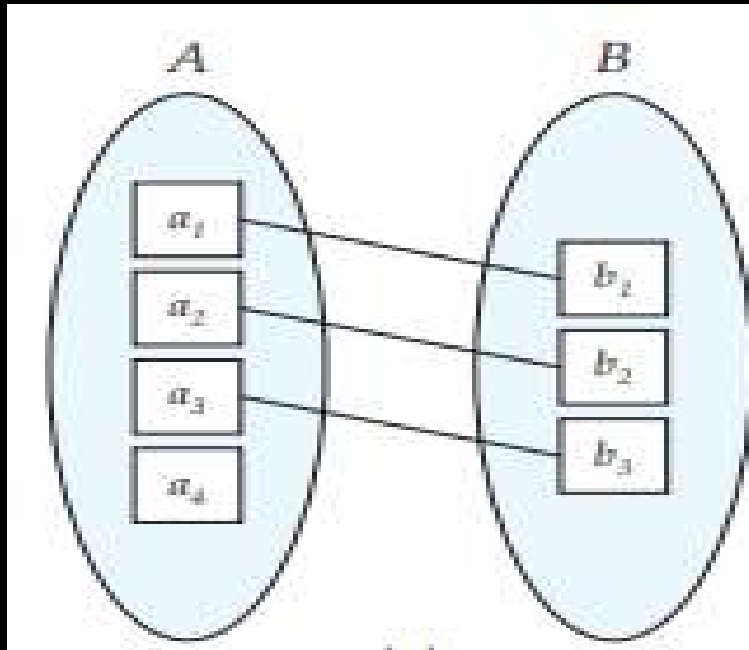
Structural constraints (Cardinalities Ratios, Participation)

- An E-R enterprise schema may define certain constraints to which the contents of a database must conform.

MAPPING CARDINALITIES / CARNINALITY RATIOS

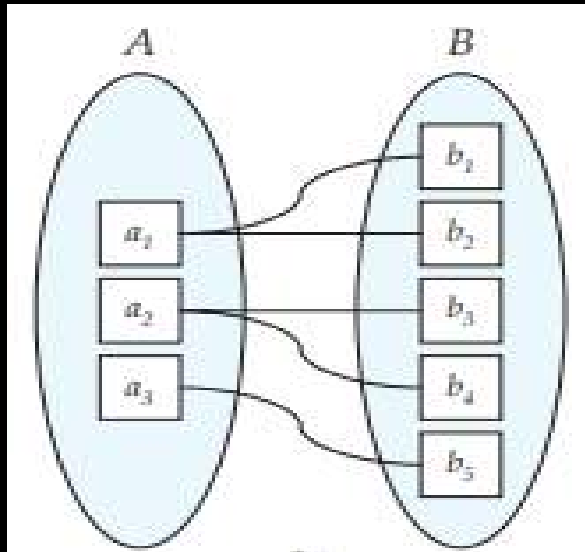
- Express the number of entities to which another entity can be associated via a relationship set. Four possible categories are-
 - One to One (1:1) Relationship.
 - One to Many (1: M) Relationship.
 - Many to One (M: 1) Relationship.
 - Many to Many (M: N) Relationship.

One to One (1:1) Relationship - An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.



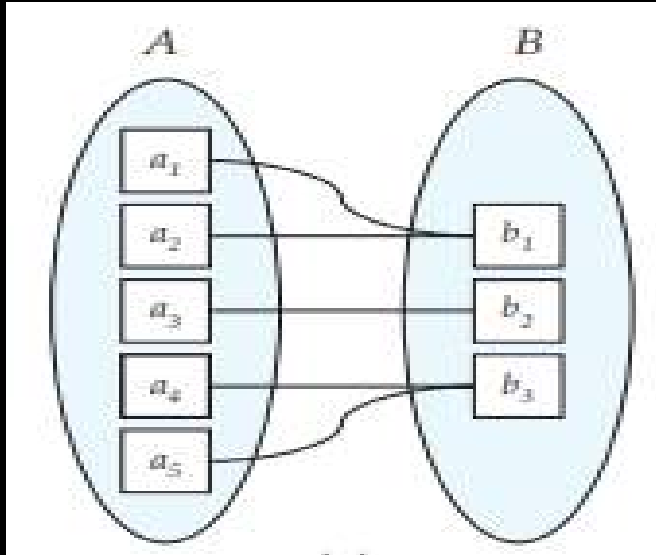
E.g.- The directed line from relationship set advisor to both entities set indicates that 'an instructor may advise at most one student, and a student may have at most one advisor'.

One to Many (1: M) Relationship - An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.



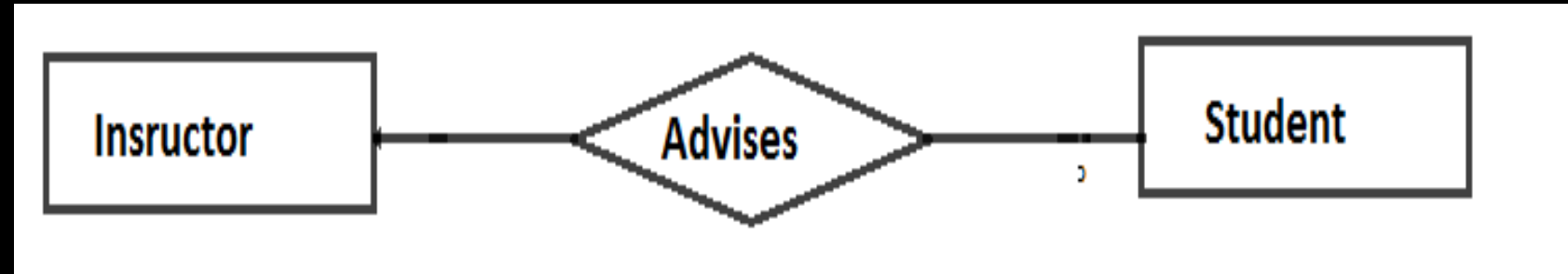
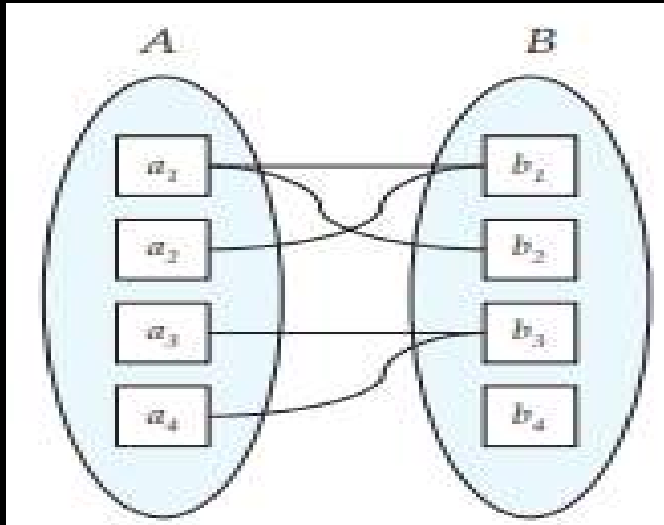
E.g.- This indicates that an instructor may advise many students, but a student may have at most one advisor.

Many to One (M: 1) Relationship - An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.



E.g.- This indicates that student may have many instructors but an instructor can advise at most one student.

Many to Many(M:N) Relationship - An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



E.g.- This indicates a student may have many advisors and an instructor may advise many students.

- **PARTICIPATION CONSTRAINTS**- it defines participations of entities of an entity type in a relationship.

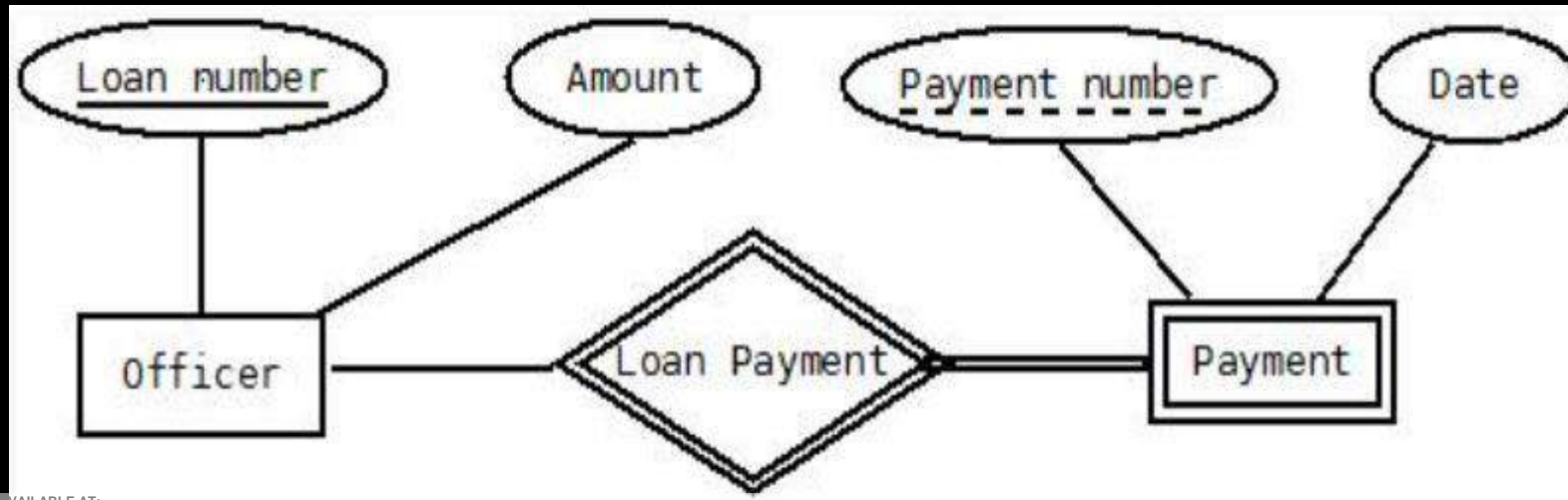
- **Partial participation**

- **Total Participation**



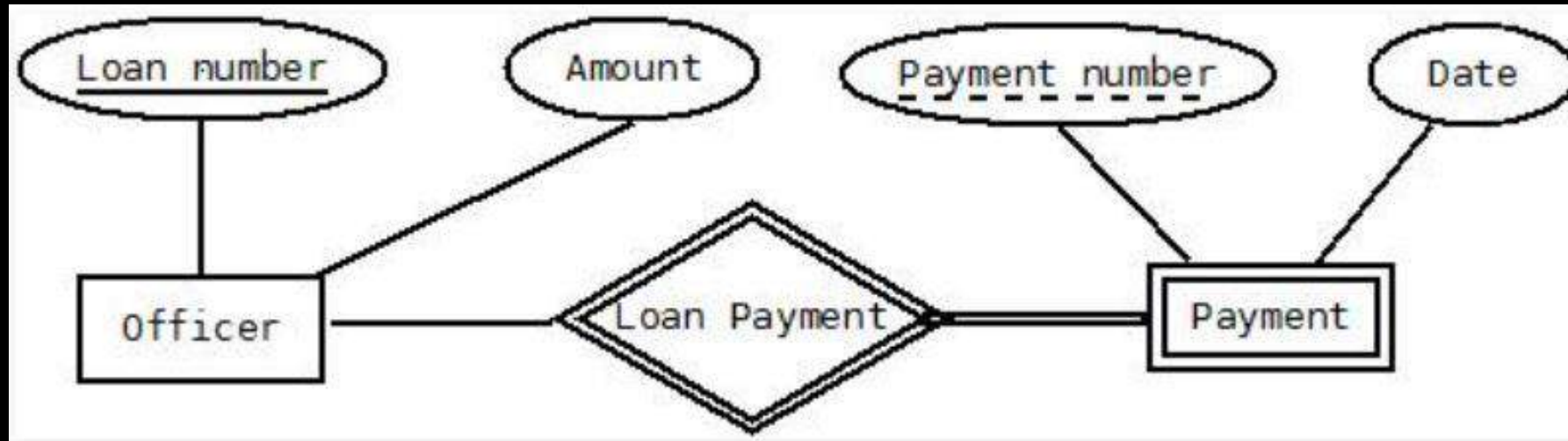
STRONG AND WEAK ENTITY SET

- An entity set is called strong entity set, if it has a primary key, all the tuples in the set are distinguishable by that key.
- An entity set that does not process sufficient attributes to form a primary key is called a weak entity set.
- It contains discriminator attributes (partial key) which contain partial information about the entity set, but it is not sufficient enough to identify each tuple uniquely. Represented by double rectangle.

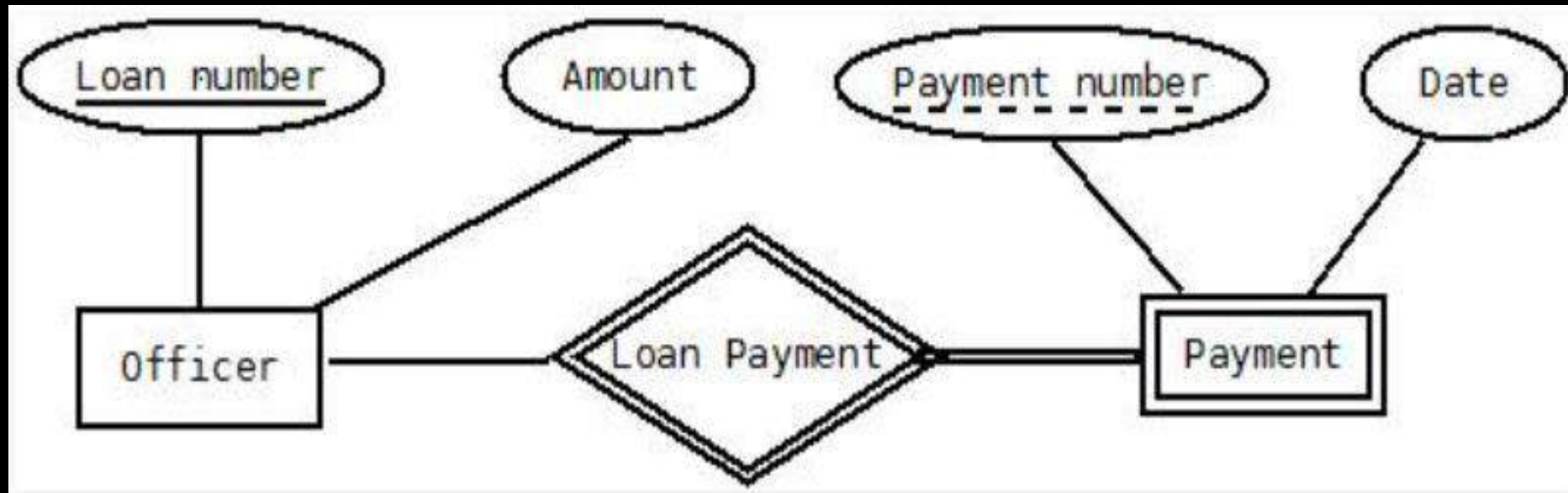


AVAILABLE AT:

- For a weak entity set to be meaningful and converted into strong entity set, it must be associated with another strong entity set called the **identifying or owner entity set** i.e. weak entity set is said to be **existence dependent** on the identity set.
- The identifying entity set is said to own weak entity set that it identifies.
- A weak entity set may participate as owner in an identifying relationship with another weak entity set.



- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship (double diamonds)**.
- The identifying relationship is many to one from the weak entity set to identifying entity set, and the participation of the weak entity set in relationship is always total.
- The primary key of weak entity set will be the union of primary key and discriminator attributes.



REASONS TO HAVE WEAK ENTITY SET

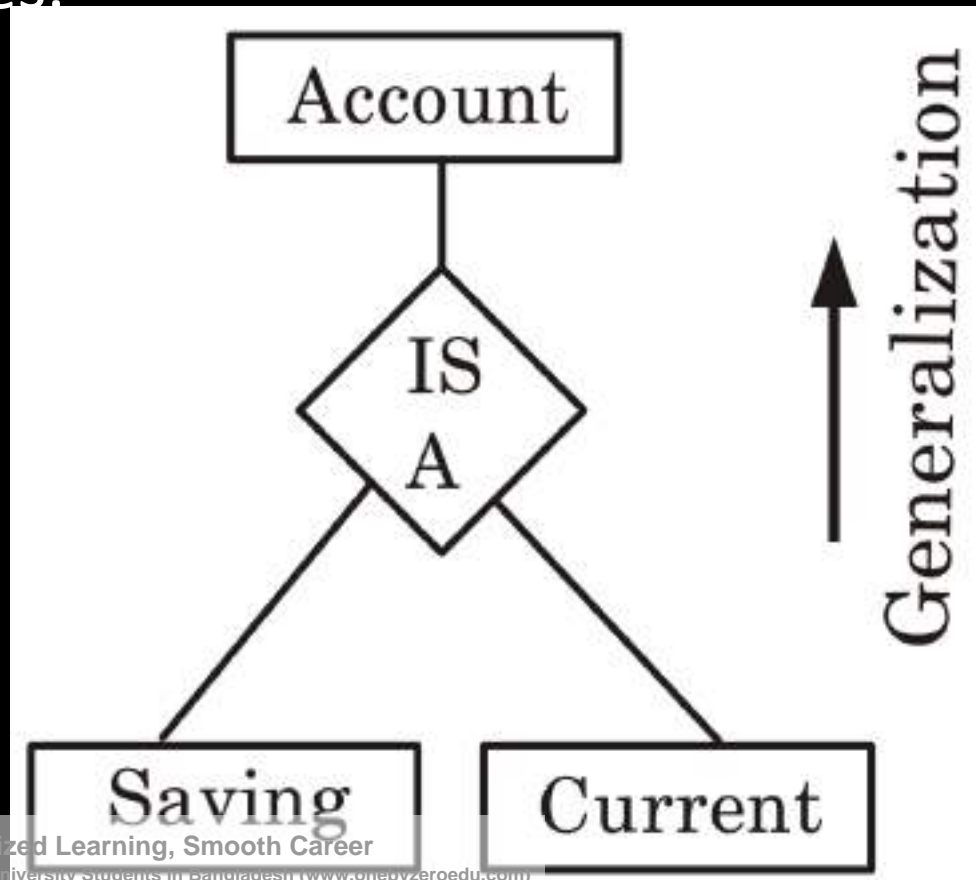
- Weak entities reflect the logical structure of an entity being dependent on another.
- Weak entity can be deleted automatically when their strong entity is deleted.
- Without weak entity set it will lead to duplication and consequent possible inconsistencies.

Conversion From ER Diagram To Relational Model

- Entity Set
 - Convert every strong, weak entity set into a separate table. In weak entity set we make it dependent onto one strong entity set (**identifying or owner entity set**).
- Relationship
 - If Unary: No separate table is required, add a new column as fk which refer the pk of the same table.
 - if 1:1 No separate table is required, take pk of one side and put it as fk on other side, priority must be given to the side having total participation.
 - if 1:n or n:1 No separate table is required, modify n side by taking pk of 1 side a foreign key on n side.
 - If m-n Separate table is required take pk of both table and declare their combination as a pk of new table
 - (3 or More) Take the pk of all participating entity sets as fk and declare their combinations as pk in the new table.
- Attributes
 - Multivalued-A separate table must be taken for all multivalued attributes, where we take pk of the main table as fk and declare combination of fk and multivalued attribute are pk in the new table.
 - Composite Attributes-A separate column must be taken for all simple attributes of the composite attribute.

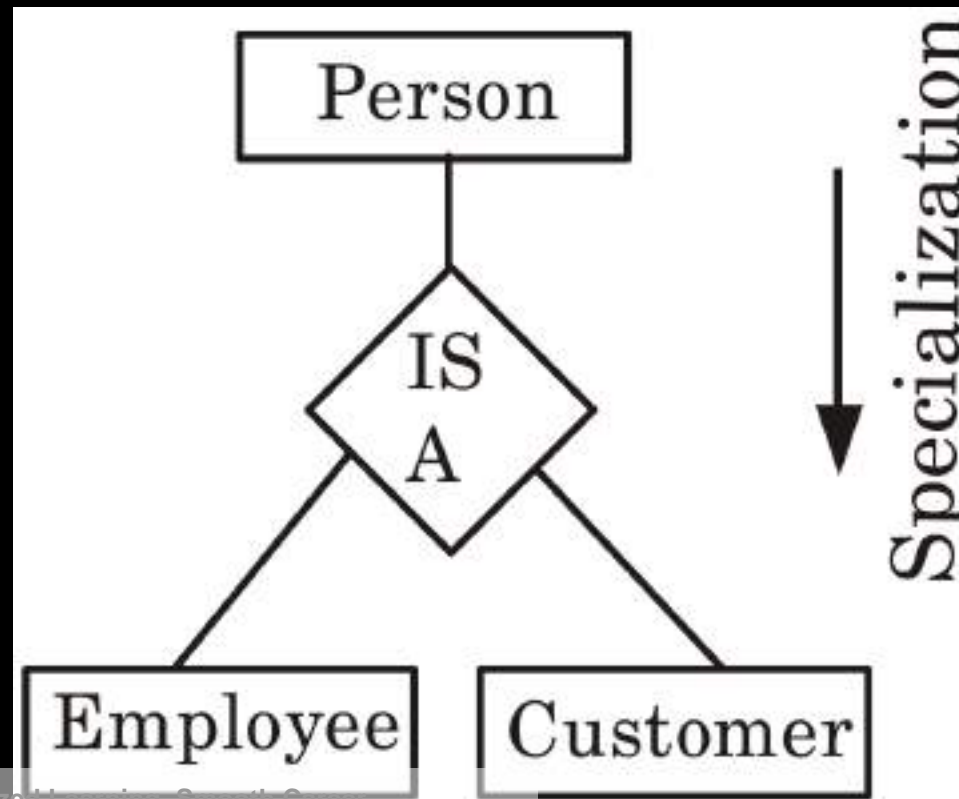
Generalization

- Involves merging two lower-level entities to create a higher-level entity.
- A bottom-up approach that builds complexity from simpler components.
- Highlights similarities among lower-level entity sets while hiding differences.
- Leads to a simplified, structured data representation, aiding in database design and querying processes.



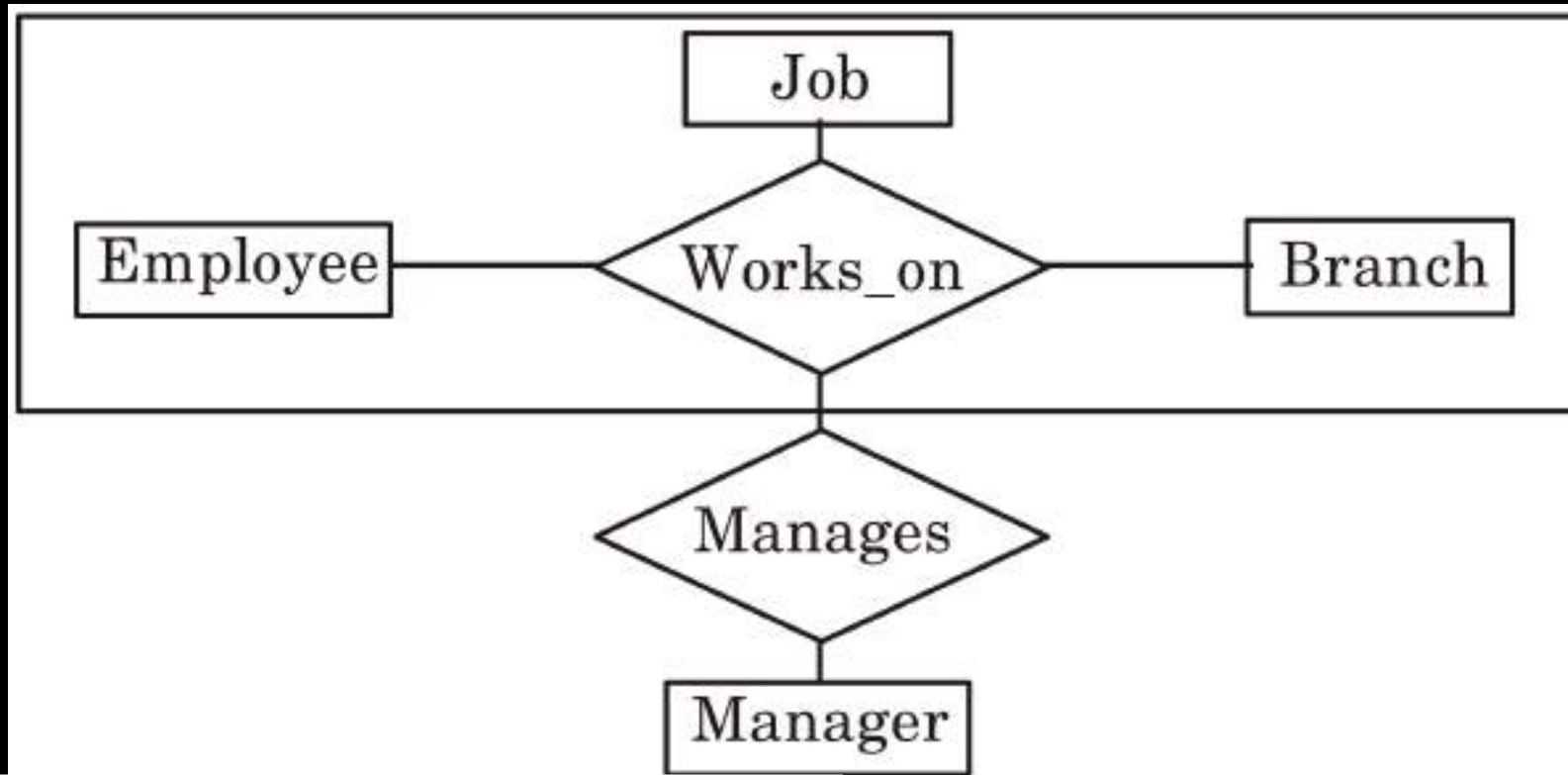
Specialization

- A process where a higher-level entity is broken down into more specific, lower-level entities.
- This top-down approach delineates complexity into simpler components.
- Acts as the converse of the generalization process, focusing on differentiating properties rather than similarities.



Aggregation

- A concept wherein relationships are abstracted to form higher-level entities, enabling a more organized representation of complex relationships.



ADVANTAGES OF E-R DIGRAM

- Constructs used in the ER diagram can easily be transformed into relational tables.
- It is simple and easy to understand with minimum training.

DISADVANTAGE OF E-R DIGRAM

- Loss of information content
- Limited constraints representation
- It is overly complex for small projects

RELATIONAL DATABASE MANAGEMENT SYSTEM

- A relational database management system (RDBMS), conceptualized by Edgar F. Codd in 1970, serves as the foundation for most contemporary commercial and open-source database applications.
- Central to its design is the utilization of tables for data storage, where it maintains and enforces specific data relationships, marking a significant evolution in database design.

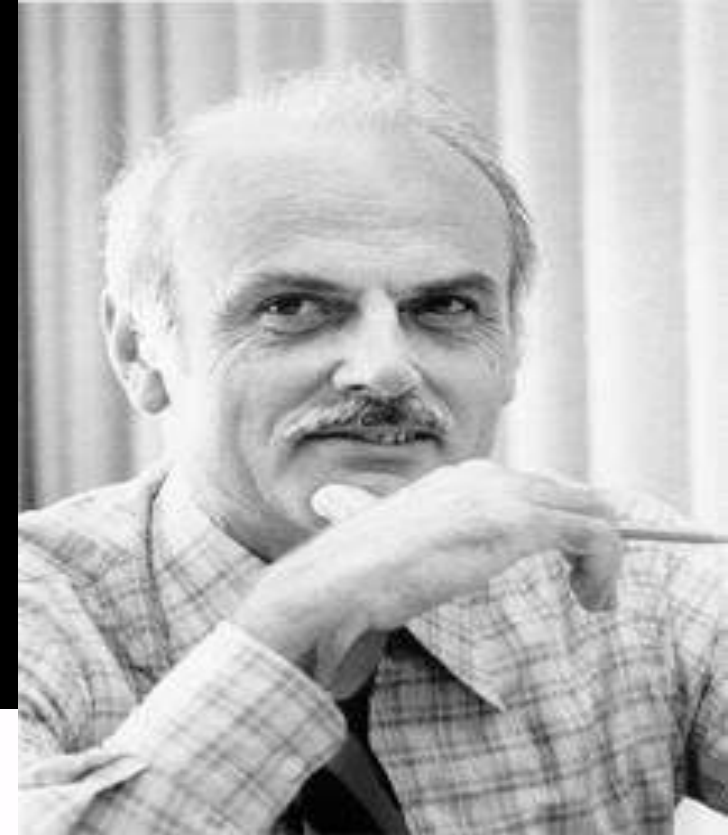


Diagram illustrating a Relational Database Table structure with annotations:


- Column (attribute)**: Points to the header row.
- Table (relation)**: Points to the entire table structure.
- Row (tuple)**: Points to a single row of data.
- Primary key**: Points to the **CustomerID** column.

CustomerID	FirstName	LastName	Birthdate
XY001	John	Doe	April 18, 1929
BR092	Mary	Green	March 4, 1980
PD500	Francesca	de la Gillebert	September 12, 1959
WI308	John	Green	March 4, 1980

BASICS OF RDBMS

- **Domain (set of permissible value in particular column)** is a set of atomic values.
- By **atomic** we mean that each value in the domain is indivisible as far as the formal relational model is concerned.
- A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
 - E.g. Names: The set of character strings that represent names of persons.

**Domain/
Field/
Column/
Arity/
Degree**



NAME	ID	CITY	COUNTRY	HOBBY
NISHA	1	AGRA	INDIA	PLAYING
NIKITA	2	DELHI	INDIA	DANCING
AJAY	3	AGRA	INDIA	CHESS
ARPIT	4	PATNA	INDIA	READING

- **Table (Relation)** - A Relation is a set of tuples/rows/entities/records.
- **Tuple** - Each row of a Relation/Table is called Tuple.
- **Arity/Degree** - No. of columns/attributes of a Relation. E.g. - Arity is 5 in Table Student.
- **Cardinality** - No of rows/tuples/record of a Relational instance. E.g. - Cardinality is 4 in table Student.

Rows/Tuples/Record/ →
Cardinality

NAME	ID	CITY	COUNTRY	HOBBY
NISHA	1	AGRA	INDIA	PLAYING
NIKITA	2	DELHI	INDIA	DANCING
AJAY	3	AGRA	INDIA	CHESS
ARPIT	4	PATNA	INDIA	READING

Properties of Relational tables

1. Cells contains atomic values
2. Values in a column are of the same kind
3. Each row is unique
4. No two tables can have the same name in a relational schema.
5. Each column has a unique name
6. The sequence of rows is insignificant
7. The sequence of columns is insignificant.

Problems in relational database

- **Update Anomalies**- Anomalies that cause redundant work to be done during insertion into and Modification of a relation and that may cause accidental loss of information during a deletion from a relation
 - **Insertion Anomalies**
 - **Modification Anomalies**
 - **Deletion Anomalies**

- **Insertion anomalies:** An independent piece of information cannot be recorded into a relation unless an irrelevant information must be inserted together at the same time.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

- **Modification anomalies:** The update of a piece of information must occur at multiple locations.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

- **Deletion Anomalies:** The deletion of a piece of information unintentionally removes other information.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

PK
↓

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

FK
↓

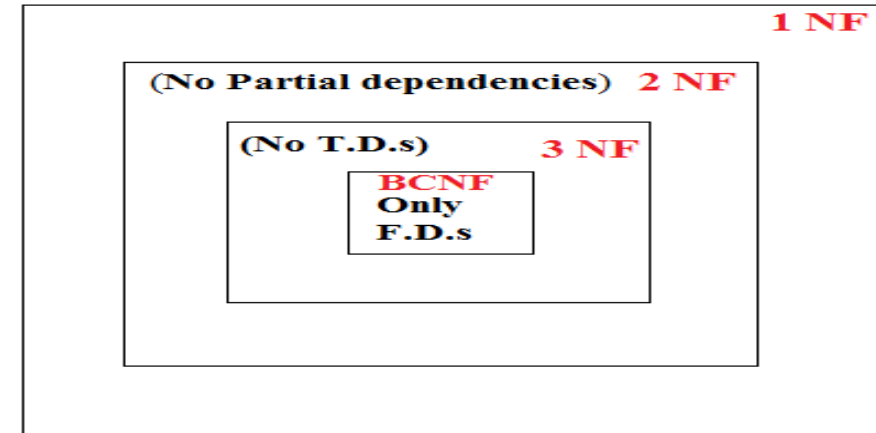


PK
↓

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	ec	Pqr

Purpose of Normalization

- Normalization may be simply defined as refinement process. Which includes creating tables and establishing relationships between those tables according to rules designed both to protect data and make the database more flexible by eliminating two factors;
 - Redundancy
 - Inconsistent Dependency
- With out normalization data base system may be inaccurate, slow and inefficient and they might not produce the data we expect. A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree.
- 1NF>>>2NF>>3NF>>BCNF



Hierarchy of Normal forms

Conclusion

1. Like every paragraph must have a single idea similarly every table must have a single idea and if a table contains more than one idea then that table must be decomposed until each table contains only one idea.
2. We need some tools to approach this decomposition or normalization on large database which contains a number of table, and that tool is functional dependencies.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	ec	Pqr

Functional Dependency



Br_code



Br_hod_name

Br_code



Br_hod_name

Functional Dependency कोई बताता नहीं, इसकी feel आ जाती है



FUNCTIONAL DEPENDENCY

- A formal tool for analysis of relational schemas.
- In a Relation R, if ' α ' \subseteq R AND ' β ' \subseteq R, then attribute or a Set of attribute ' α ' Functionally derives an attribute or set of attributes ' β ', iff each ' α ' value is associated with precisely one ' β ' value.
- For all pairs of tuples t_1 and t_2 in R such that
 - If $T_1[\alpha] = T_2[\alpha]$
 - Then, $T_1[\beta] = T_2[\beta]$

X	Y	Z
1	4	2
1	4	3
2	6	3
3	2	2

Q Consider the following relation instance, which of the following dependency doesn't hold

A) $A \rightarrow B$

B) $BC \rightarrow A$

C) $B \rightarrow C$

D) $AC \rightarrow B$

A	B	C
1	2	3
4	2	3
5	3	3

- Trivial Functional dependency - If β is a subset of α , then the functional dependency $\alpha \rightarrow \beta$ will always hold.

जिसका होना न होना बराबर हो



X	Y	Z
1	4	2
1	4	3
2	6	3
3	2	2

ATTRIBUTES CLOSURE/CLOSURE ON ATTRIBUTE SET/ CLOSURE SET OF ATTRIBUTES

- Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from F.
 - DENOTED BY F^+

R (A, B, C, D)

$A \rightarrow B$

$B \rightarrow C$

$AB \rightarrow D$

$A^+ =$

R(ABCDEFGFG)

$A \rightarrow B$

$BC \rightarrow DE$

$AEG \rightarrow G$

$(AC)^+ = ?$

Q R(ABCDE)

$A \rightarrow BC$

$CD \rightarrow E$

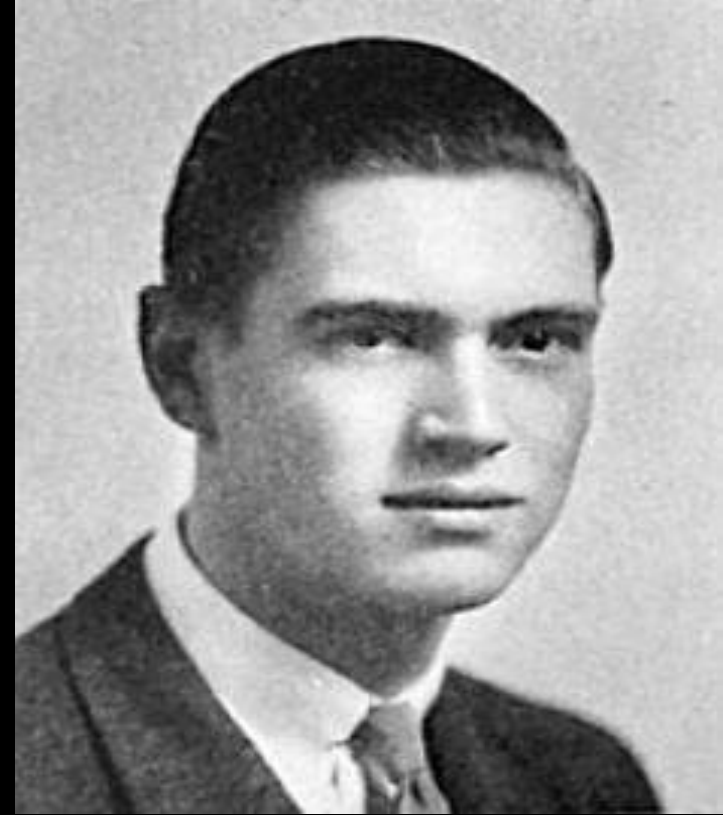
$B \rightarrow D$

$E \rightarrow A$

$(B)^+ =$

ARMSTRONG'S AXIOMS

1. An **axiom** or **postulate** is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments.
2. **Armstrong's axioms** are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong in his 1974 paper.
3. The axioms are sound in generating only functional dependencies in the closure of a set of functional dependencies (denoted as F^+) when applied to that set (denoted as F).



Armstrong Axioms

- **Reflexivity:** If Y is a subset of X , then $X \rightarrow Y$
- **Augmentation:** If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity:** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

From these rules, we can derive these secondary rules-

- **Union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- **Decomposition:** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Pseudo transitivity:** If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
- **Composition:** If $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$

Why Armstrong axioms refers to the Sound and Complete

- By sound, we mean that given a set of functional dependencies F specified on a relation schema R , any dependency that we can infer from F by using the primary rules of Armstrong axioms holds in every relation state r of R that satisfies the dependencies in F .
- By complete, we mean that using primary rules of Armstrong axioms repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F .

Equivalence of Two FD sets-

Two FD sets F_1 and F_2 are equivalent if –

$$F_1^+ = F_2^+$$

Or

$$F_1 \subseteq F_2^+ \text{ and } F_2 \subseteq F_1^+$$

Q Consider the following set of fd R(ACDEH)

F	G
$A \rightarrow C$	$A \rightarrow CD$
$AC \rightarrow D$	
$E \rightarrow AD$	$E \rightarrow AH$
$E \rightarrow H$	

To find the MINIMAL COVER / CANONICAL COVER / IRREDUCIBLE SET

- A **canonical cover** (also known as a minimal cover) for a set of functional dependencies in a database is a minimal set of functional dependencies that is equivalent to the original set, but with redundant dependencies and extraneous attributes removed. It is used in the normalization process of database design to simplify the set of functional dependencies and to find a good set of relations.
- There may be any following type of redundancy in the set of functional dependencies: -
 - Complete production may be Redundant.
 - One or more than one attributes may be redundant on right hand side of a production.
 - One or more than one attributes may be redundant on Left hand side of a production.

$Q \vee R(ABCD)$

$A \rightarrow B$

$C \rightarrow B$

$D \rightarrow ABC$

$AC \rightarrow D$

$R(A,B,C)$

$A \rightarrow B$

$B \rightarrow C$

$A \rightarrow C$

$AB \rightarrow B$

$AB \rightarrow C$

$AC \rightarrow B$

Key



Super key

- Set of attributes using which we can identify each tuple uniquely is called Super key.
- Let X be a set of attributes in a Relation R , if X^+ (Closure of X) determines all attributes of R then X is said to be Super key of R .
- There should be at least one Super key in every relation.

Candidate key

- Minimal set of attributes using which we can identify each tuple uniquely is called Candidate key. A super key is called candidate key if it's No proper subset is a super key. Also called as **MINIMAL SUPER KEY**.
- There should be at least one candidate key.

Prime attribute - Attributes that are member of at least one candidate Keys are called Prime attributes.

Primary key

- One of the candidate keys is selected by database administrator as a Primary means to identify tuple is called primary Key. Primary Key attribute are not allowed to have Null values. Exactly one Primary Key per table in RDMS.
- Candidate key which are not chosen as primary key is alternate key.

Foreign Keys

- A foreign key is a column or group of columns in a relational database table that refers the primary key of the same table or some other table to represent relationship.
- The ***concept of referential integrity*** is derived from foreign key theory.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

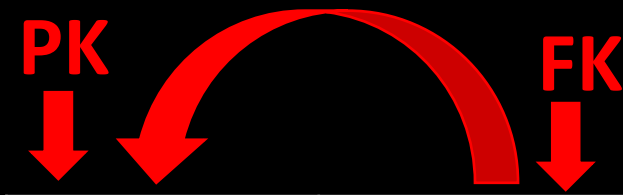
PK
↓

FK
↓

PK
↓

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	ec	Pqr



Roll no	CR
1	1
2	2
3	1
4	2
5	1
6	2
7	1
8	2
9	1
10	2
11	1
12	2
13	1
14	2
15	null

- **Composite key** – Composite key is a key composed of more than one column sometimes it is also known as concatenated key.
- **Secondary key** – Secondary key is a key used to speed up the search and retrieval contrary to primary key, a secondary key does not necessary contain unique values.

NORMAL FORM

FIRST NORMAL FORM

- 1NF is the initial step of database normalization.
- Implications of first normal form
 - **Atomic Values**: Each cell in a table contains indivisible, atomic values. Means a Relation should not contain any multivalued or composite attributes.
 - **Unique Columns**: Each column must have a distinct name to identify the data it contains.
 - **Primary Key**: A table in 1NF should have a primary key that uniquely identifies each record.
 - **Eliminating Duplicates**: Duplicate rows are removed to prevent data redundancy.

Prime attribute: - A attribute is said to be prime if it is part of any of the candidate key

Non-Prime attribute: - A attribute is said to be non-prime if it is not part of any of the candidate key

Eg R(ABCD)

AB \rightarrow CD

Here candidate key is AB so, A and B are prime attribute, C and D are non-prime attributes.

PARTIAL DEPENDENCY- When a non – prime attribute is dependent only on a part (Proper subset) of candidate key then it is called partial dependency. (PRIME > NON-PRIME)

Full DEPENDENCY- When a non – prime attribute is dependent on the entire candidate key then it is called Full dependency.

e.g. R(ABCD)

$AB \rightarrow D$

$A \rightarrow C$

SECOND NORMAL FORM

- Relation R is in 2NF if,
 - R should be in 1 NF.
 - R should not contain any Partial dependency. (that is every non-prime attribute should be fully dependent upon candidate key)

Q R(A, B, C) $B \rightarrow C$

A	B	C
a	1	X
b	2	Y
a	3	Z
C	3	Z
D	3	Z
E	3	Z

A	B
A	1
B	2
A	3
C	3
D	3
E	3

B	C
1	X
2	Y
3	z



TRANSITIVE DEPENDENCY – A functional dependency from non-Prime attribute to non-Prime attribute is called transitive

E.g.- R(A, B, C, D) with A as a candidate key

$A \rightarrow B$

$B \rightarrow C$ [transitive dependency]

$C \rightarrow D$ [transitive dependency]

THIRD NORMAL FORM

- Let R be the relational schema, it is said to be in 3 NF
 - R should be in 2NF
 - It must not contain any transitive dependency

THIRD NORMAL FORM DIRECT DEFINATION

- A relational schema R is said to be 3 NF if every functional dependency in R from $\alpha \rightarrow \beta$, either α is super key or β is the prime attribute

$R(A, B, C)$

$A \rightarrow B$

$B \rightarrow C$

A	B	C
A	1	P
B	2	Q
C	2	Q
D	2	Q
E	3	R
F	3	R
G	4	S

A	B
A	1
B	2
C	2
D	2
E	3
F	3
G	4

B	C
1	P
2	Q
3	R
4	S

BCNF (BOYCE CODD NORMAL FORM)

- A relational schema R is said to be BCNF if every functional dependency in R from
 - $\alpha \rightarrow \beta$
 - α must be a super key

$R(A, B, C)$

$AB \rightarrow C$

$C \rightarrow B$

A	B	C
A	C	B
B	B	C
B	A	D
A	A	E
C	C	B
D	C	B
E	C	B
F	C	B

A	B
A	B
B	B
B	A
A	A
C	C
D	C
e	C
f	c

C	B
B	C
C	B
D	A
E	A

Some important note points on Normalization:

- A Relation with two attributes is always in BCNF.
- A Relation schema R consist of only prime attributes then R is always in 3NF, but may or may not be in BCNF.

Q Consider the universal relational schema R (A, B, C, D, E, F, G, H, I, J) and a set of following functional dependencies.

$F = \{AB \rightarrow C, A \rightarrow DE, B \rightarrow F, F \rightarrow GH, D \rightarrow IJ\}$ determine the keys for R ?

Decompose R into 2nd normal form.

Q Find the normal form of relation $R(A, B, C, D, E)$ having FD set $F = \{A \rightarrow B, BC \rightarrow E, ED \rightarrow A\}$.

Multivalued Dependency

- Denoted by, $A \twoheadrightarrow B$, Means, for every value of A, there may exist more than one value of B.
- E.g. $S_name \twoheadrightarrow Club_name$

S_Name	Club_name
Kamesh	Dance
Kamesh	Guitar

- A **trivial multivalued dependency** $X \twoheadrightarrow Y$ is one where either Y is a subset of X , or X and Y together form the whole set of attributes of the relation.

$S_name \twoheadrightarrow Club_name$

S_Name	Club_name
Kamesh	Dance
Kamesh	Guitar

$S_name \twoheadrightarrow Club_name$
 $S_name \twoheadrightarrow P_no$

S_Name	Club_name	P_no
Kamesh	Dance	123
Kamesh	Guitar	123
Kamesh	Dance	789
Kamesh	Guitar	789

E.g. let the constraint specified by MVD in relation **Student** as

$S_name \twoheadrightarrow Club_name$

$S_name \twoheadrightarrow P_no$

S_Name	Club_name
Kamesh	Dance
Kamesh	Guitar

S_Name	P_no
Kamesh	123
Kamesh	789

S_Name	Club_name	P_no
Kamesh	Dance	123
Kamesh	Guitar	123
Kamesh	Dance	789
Kamesh	Guitar	789

NOTE: The above Student schema is in BCNF as no functional dependency holds on EMP, but still redundancy due to MVD.

- Each row indicates that a given restaurant can deliver a given variety. The table has no non-key attributes because its only key is {Restaurant, Variety, Delivery Area}. Therefore, it meets all normal forms up to BCNF.
- If we assume, however, that Variety offered by a restaurant are not affected by delivery area (i.e. a restaurant offers all Variety it makes to all areas it supplies), then it does not meet 4NF. The problem is that the table features two non-trivial multivalued dependencies on the {Restaurant} attribute (which is not a super key). The dependencies are:

- {Restaurant} {Variety}
- {Restaurant} {Delivery Area}

Restaurant Delivery Permutations		
Restaurant	Variety	Delivery Area
Chatora Sweets	Samosa	Hatibagan Market
Chatora Sweets	Samosa	Chandni Chowk
Chatora Sweets	Samosa	Koramangala
Chatora Sweets	Dosa	Hatibagan Market
Chatora Sweets	Dosa	Chandni Chowk
Chatora Sweets	Dosa	Koramangala
Moolchand	Ladoo	Koramangala
Moolchand	Dosa	Koramangala
Thaggu	Samosa	Hatibagan Market
Thaggu	Samosa	Chandni Chowk
Thaggu	Ladoo	Hatibagan Market
Thaggu	Ladoo	Chandni Chowk

- If we have two or more multivalued *independent* attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

Delivery Areas By Restaurant	
Restaurant	Delivery Area
Chatora Sweets	Hatibagan Market
Chatora Sweets	Chandni Chowk
Chatora Sweets	Koramangala
Moolchand	Koramangala
Thaggu	Hatibagan Market
Thaggu	Chandni Chowk

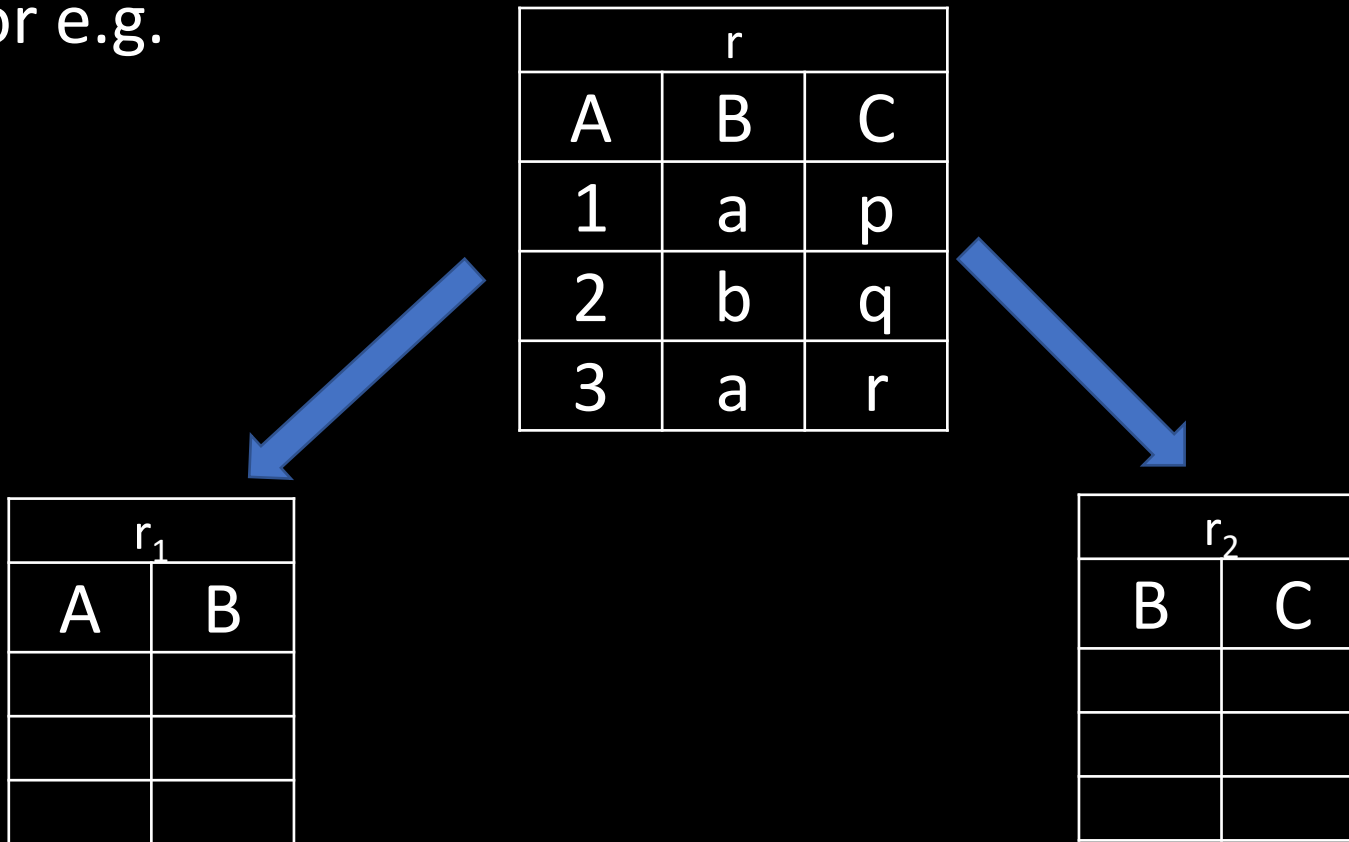
Varieties By Restaurant	
Restaurant	Pizza Variety
Chatora Sweets	Samosa
Chatora Sweets	Dosa
Moolchand	Ladoo
Moolchand	Dosa
Thaggu	Samosa
Thaggu	Ladoo

- A relation is in 4NF iff
 - It is in BCNF
 - There must not exist any non-trivial multivalued dependency.
- Each MVD is decomposed in separate table, where it becomes trivial MVD.

Lossy/Lossless-Dependency Preserving Decomposition

- Because of a normalization a table is Decomposed into two or more tables, but during this decomposition we must ensure satisfaction of some properties out of which the most important is lossless join property / decomposition.

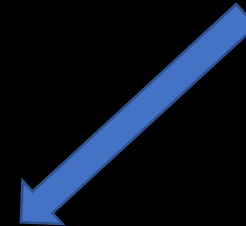
- if we decompose a table r into two tables r_1 and r_2 because of normalization then at some later stage if we want to join (combine) (natural join) these tables r_1 and r_2 , then we must get back the original table r , without any extra or less tuple. But some information may be lost during retrieval of original relation or table. For e.g.



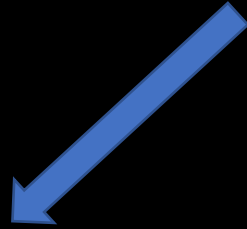
r		
A	B	C
1	a	p
2	b	q
3	a	r



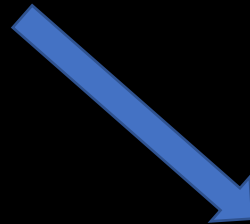
r ₂	
B	C
a	p
b	q
a	r



A	B	C



r ₁	
A	B
1	a
2	b
3	a



- Decomposition is lossy if $R_1 \bowtie R_2 \supset R$
- Decomposition is lossy if $R \supset R_1 \bowtie R_2$

- Decomposition is lossless if $R_1 \bowtie R_2 = R$ "The decomposition of relation R into R_1 and R_2 is lossless when the join of R_1 and R_2 yield the same relation as in R." which guarantees that the spurious (extra or less) tuple generation problem does not occur with respect to the relation schemas created after decomposition.
- This property is extremely critical and must be achieved at any cost.
- lossless Decomposition / NonAdditive Join Decomposition

A	B	C	D	E
A	122	1	W	A
E	236	4	X	B
A	199	1	Y	C
B	213	2	Z	D

How to check for lossless join decomposition using FD set, following conditions must hold:

- Union of Attributes of R_1 and R_2 must be equal to attribute of R . Each attribute of R must be either in R_1 or in R_2 . $\text{Att}(R_1) \cup \text{Att}(R_2) = \text{Att}(R)$
- Intersection of Attributes of R_1 and R_2 must not be NULL. $\text{Att}(R_1) \cap \text{Att}(R_2) \neq \Phi$
- Common attribute must be a key for at least one relation (R_1 or R_2)
 - $\text{Att}(R_1) \cap \text{Att}(R_2) \rightarrow (R_1)$ or $\text{Att}(R_1) \cap \text{Att}(R_2) \rightarrow (R_2)$

Q R (A, B, C, D)

$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A$

$R_1(A, B), R_2(B, C)$ AND $R_3(C, D)$

Q R(ABCDE)(NF) $R_1(AB)$ $R_2(BC)$ $R_3(ABCD)$ $R_4(EG)$

$A \rightarrow BC$

$C \rightarrow DE$

$D \rightarrow E$

5 NF/Project-Join Normal Form

- A Relational table R is said to be in 5th normal form if
 - it is in 4 NF
 - it cannot be further non-loss decomposed

Dependency Preserving Decomposition

- Let relation R be decomposed into Relations $R_1, R_2, R_3, \dots, R_N$ with their respective functional Dependencies set as $F_1, F_2, F_3, \dots, F_N$, then the Decomposition is Dependency Preserving iff
 - $\{F_1 \cup F_2 \cup F_3 \cup F_4 \dots \cup F_N\}^+ = F^+$
- Dependency preservation property, although desirable, is sometimes sacrificed.

$X = (PQRS)$

$F = \{QR \rightarrow S, R \rightarrow P, S \rightarrow Q\}$

$Y = (PR)$ and $Z = (QRS)$

Indexing

Relational databases are based on set theory.

- In set theory, the order of elements is unimportant, similarly in database tables.
- However, in practical implementation, element order in tables is often specified.
- Various operations such as search, insertion, and deletion are influenced by the order of elements in the tables.
- Elements in a table can be stored in two ways: sorted (ordered) or unsorted (unordered).

File organization/ organization of records in a file

Ordered file organization

- All the records in the file are ordered on some search key field.
- Here binary search is possible. (give example of book page searching)
- Maintenance (insertion & deletion) is costly, as it requires reorganization of entire file.
- Notes that we will get binary search only if we are using that key for searching on which indexing is done, otherwise it will behave as unsorted file.
- if file is unordered then no of block accesses required to reach correct block which contain the desired record is $O(\log_2 n)$, where n is the number of blocks.



Permissions	Owner	Group	Size	File Name	
-rw-r-----	1	mysql	mysql	10M	node__field_meta_tags.ibd
-rw-r-----	1	mysql	mysql	10M	comment__field_data.ibd
-rw-r-----	1	mysql	mysql	11M	search_total.ibd
-rw-r-----	1	mysql	mysql	11M	cache_discovery.ibd
-rw-r-----	1	mysql	mysql	12M	node_field_data.ibd
-rw-r-----	1	mysql	mysql	14M	url_alias.ibd
-rw-r-----	1	mysql	mysql	14M	taxonomy_index.ibd
-rw-r-----	1	mysql	mysql	15M	node__tags.ibd
-rw-r-----	1	mysql	mysql	15M	comment__comment_body.ibd
-rw-r-----	1	mysql	mysql	16M	node_revision__tags.ibd
-rw-r-----	1	mysql	mysql	19M	sessions.ibd
-rw-r-----	1	mysql	mysql	22M	search_dataset.ibd
-rw-r-----	1	mysql	mysql	31M	node__body.ibd
-rw-r-----	1	mysql	mysql	32M	watchdog.ibd
-rw-r-----	1	mysql	mysql	36M	node_revision__body.ibd
-rw-r-----	1	mysql	mysql	108M	search_index.ibd
-rw-r-----	1	mysql	mysql	120M	cache_entity.ibd
-rw-r-----	1	mysql	mysql	268M	cache_data.ibd
-rw-r-----	1	mysql	mysql	1.6G	cache_dynamic_page_cache.ibd
-rw-r-----	1	mysql	mysql	3.4G	cache_render.ibd

Unordered file organization

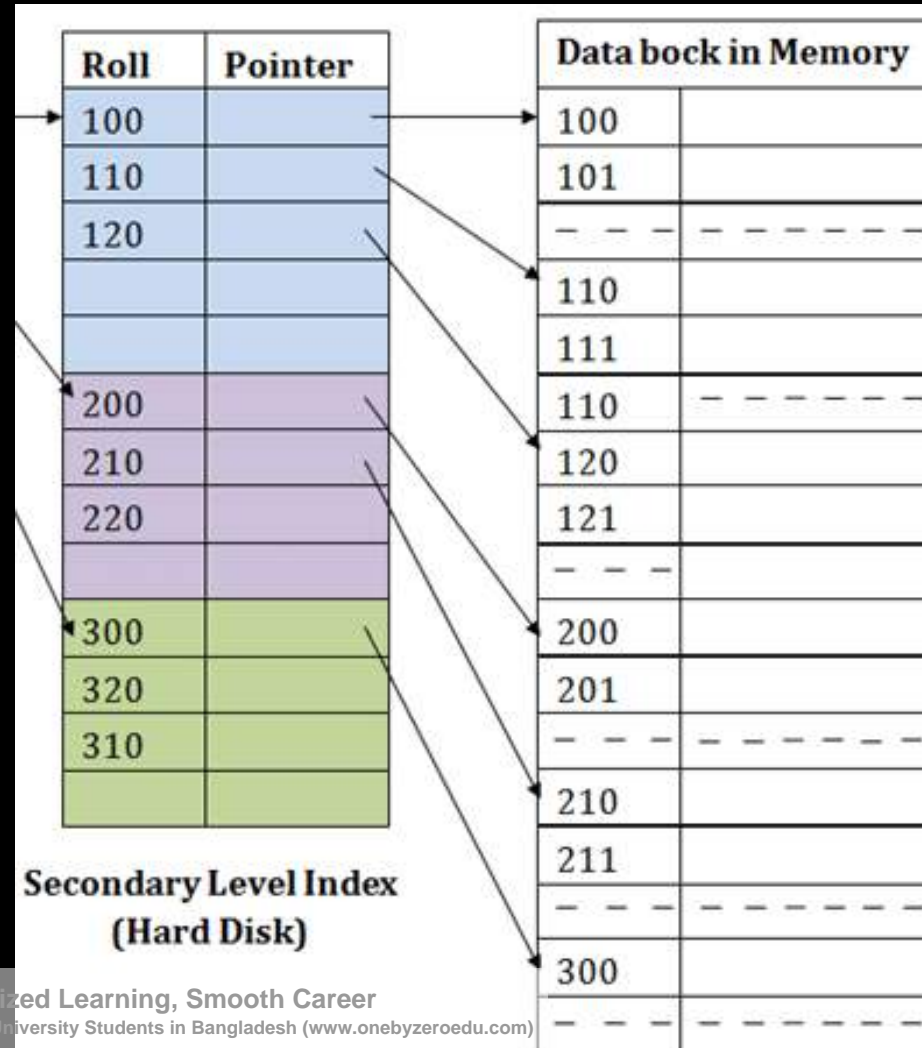
- Records are typically added at the end of the file, without following any specific order.
- This insertion method allows only linear search, resulting in slower search times.
- Despite slow searches, maintenance including insertion and deletion is simpler.
- No reorganization of the entire file is needed, making maintenance easier.
- If file is unordered then no of block accesses required to reach correct block which contain the desired record is $O(n)$, where n is the number of blocks.

record 0	A-102	Perryridge	400
record 1	A-305	Round Hill	350
record 2	A-215	Mianus	700
record 3	A-101	Downtown	500
record 4	A-222	Redwood	700
record 5	A-201	Perryridge	900
record 6	A-217	Brighton	750
record 7	A-110	Downtown	600
record 8	A-218	Perryridge	700

- Indexes are supplementary structures in databases, aiding in swift record retrieval.
- They enable quick data access based on particular attributes identified for indexing.
- This technique is similar to the index sections seen in books.
- Indexes provide secondary pathways to access records without changing their physical position in the main file.



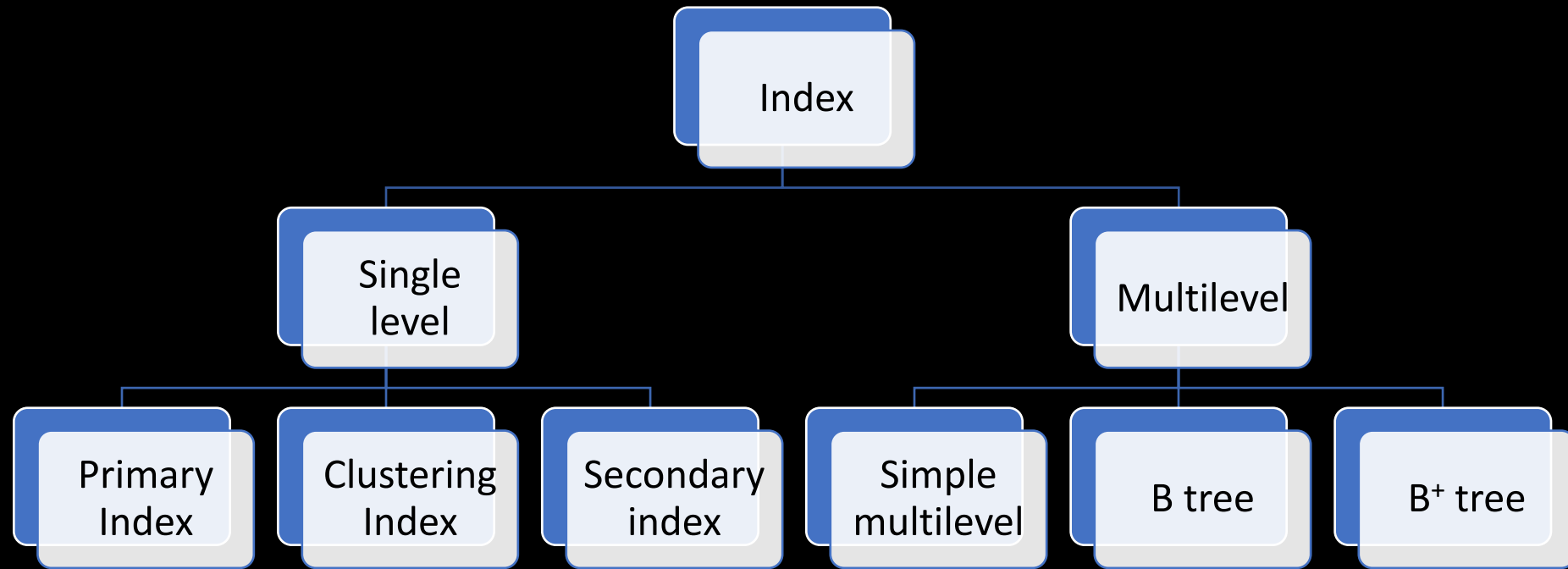
- The size of index file is way smaller than that of the main file, as index file record contain only two columns key (attribute in which searching is done) and block pointer (base address of the block of main file which contains the record holding the key), while main file contains all the columns.



Q Suppose we have ordered file with records stored $r = 30,000$ on a disk with Block Size $B = 1024$ B. File records are of fixed size and are unspanned with record length $R = 100$ B. Suppose that ordering key field of file is 9 B long and a block pointer is 6 B long, Implement primary indexing?

1. Indexes can be established on any relation field, be it primary key or non-key.
2. Each attribute can have a dedicated index file, meaning multiple index files may exist for one main file.
3. Index files are always organized, allowing for the utilization of binary search advantages, irrespective of the main file's order.
4. Indexing accelerates data retrieval time but also introduces space overhead for storing the index file.
5. The correct block in the main file can be located with $\log_2(\text{number of blocks in index file}) + 1$ accesses.

TYPES OF INDEXING



- In single-level indexing, an index file is created for the main file, marking the end of the indexing process.
- Multiple-level indexing, on the other hand, involves creating an index for the index file and continually repeating this procedure until only a single block remains.

PRIMARY INDEXING

- Main file is always sorted according to primary key.
- Indexing is done on Primary Key, therefore called as primary indexing
- Index file have two columns, first primary key and second anchor pointer (base address of block)
- It is an example of Sparse Indexing.
- Here first record (anchor record) of every block gets an entry in the index file
- No. of entries in the index file = No of blocks acquired by the main file.

CLUSTERED INDEXING

- Main file will be ordered on some non-key attributes
- No of entries in the index file = no of unique values of the attribute on which indexing is done.
- It is the example of Sparse as well as dense indexing

Q Suppose we have ordered file with records stored $r = 30,000$ on a disk with Block Size $B = 1024$ B. File records are of fixed size and are unspanned with record length $R = 100$ B. Suppose that ordering key field of file is 9 B long and a block pointer is 6 B long, Implement Secondary indexing?

SECONDARY INDEXING

- Most common scenarios, suppose that we already have a primary indexing on primary key, but there is frequent query on some other attributes, so we may decide to have one more index file with some other attribute.
- Main file is ordered according to the attribute on which indexing is done(unordered).
- Secondary indexing can be done on key or non-key attribute.
- No of entries in the index file is same as the number of entries in the main file.
- It is an example of dense indexing.

Dense Vs Sparse

- **Dense Index** In dense index, there is an entry in the index file for every search key value in the main file. This makes searching faster but requires more space to store index records itself. Note that it is not for every record, it is for every search key value. Sometime number of records in the main file $>$ number of search keys in the main file, for example if search key is repeated.
- **Sparse Index**-If an index entry is created only for some records of the main file, then it is called sparse index. No. of index entries in the index file $<$ No. of records in the main file. Note: - dense and sparse are not complementary to each other, sometimes it is possible that a record is both dense and sparse.

B tree

- A B-tree of order m if non-empty is an m -way search tree in which.
 - The root has at least zero child nodes and at most m child nodes.
 - The internal nodes except the root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes.
 - The number of keys in each internal node is one less than the number of child nodes and these keys partition the subtrees of the nodes in a manner similar to that of m -way search tree.
 - All leaf nodes are on the same level(perfectly balanced).

Root

Rules	MAX	MIN
CHILD	m	0
DATA	$m-1$	1

Internal except Root

Rules	MAX	MIN
CHILD	m	$\lceil m/2 \rceil$
DATA	$m-1$	$\lceil m/2 \rceil - 1$

Leaf

Rules	MAX	MIN
CHILD	0	0
DATA	$m-1$	$\lceil m/2 \rceil - 1$

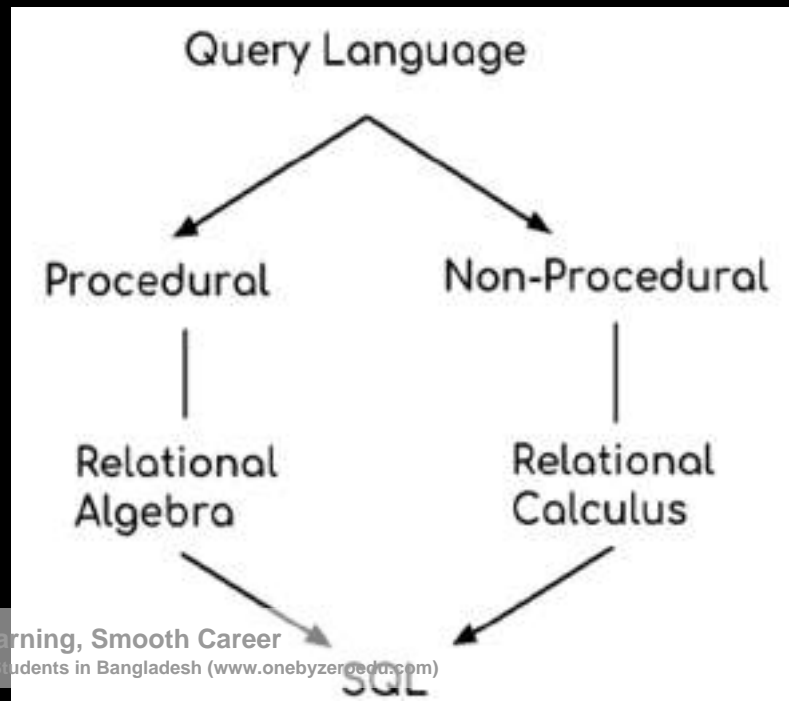
Insertion in B-TREE

- A B-tree starts with a single root node (which is also a leaf node) at level 0 (zero). Once the root node is full with $m - 1$ search key values and we attempt to insert another entry in the tree, the root node splits into two nodes at level 1.
- Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes. When a non-root node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes.
- If the parent node is full, it is also split. Splitting can propagate all the way to the root node, creating a new level if the root is split.

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 3, 4
insert them into an empty b-tree of order = 3.

Query Language

- After designing a data base, that is ER diagram followed by conversion in relational model followed by normalization and indexing, now next task is how to store, retrieve and modify data in the data base.
- So here we will be concentrating more on the retrieval part. Query languages are used for this purpose. **Query languages, data query languages or database query languages (DQLs)** are computer languages using which user request some information from the database. A well known example is the **Structured Query Language (SQL)**.



Procedural Query Language

- Here users instruct the system to performs a sequence of operations on the data base in order to compute the desired result.
- Means user provides both what data to be retrieved and how data to be retrieved. e.g. Relational Algebra.

Non-Procedural Query Language

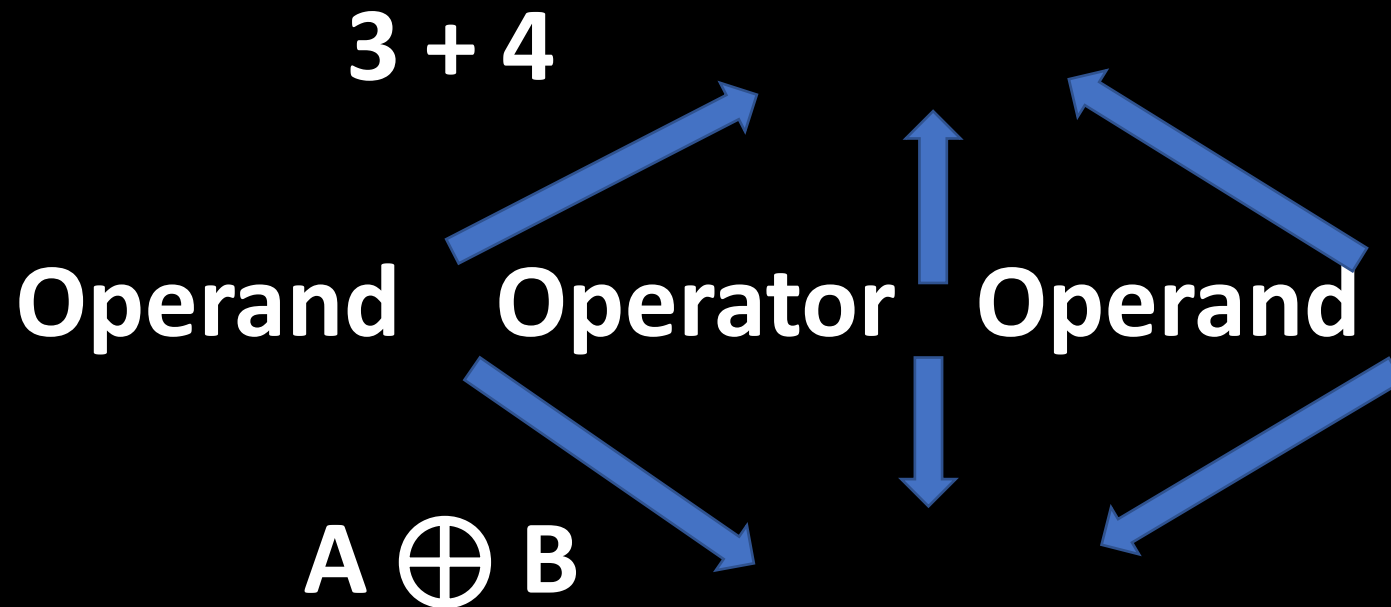
- In nonprocedural language, the user describes the desired information without giving a specific procedure for obtaining that information.
- What data to be retrieved e.g. Relational Calculus. **Tuple relational calculus, Domain relational calculus** are declarative query languages based on mathematical logic

- Relational Algebra (Procedural) and Relational Calculus (non-procedural) are mathematical system/ query languages which are used for query on relational model.
- RA and RC are not executed in any computer they provide the fundamental mathematics on which SQL is based.
- SQL (structured query language) works on RDBMS, and it includes elements of both procedural or non-procedural query language.

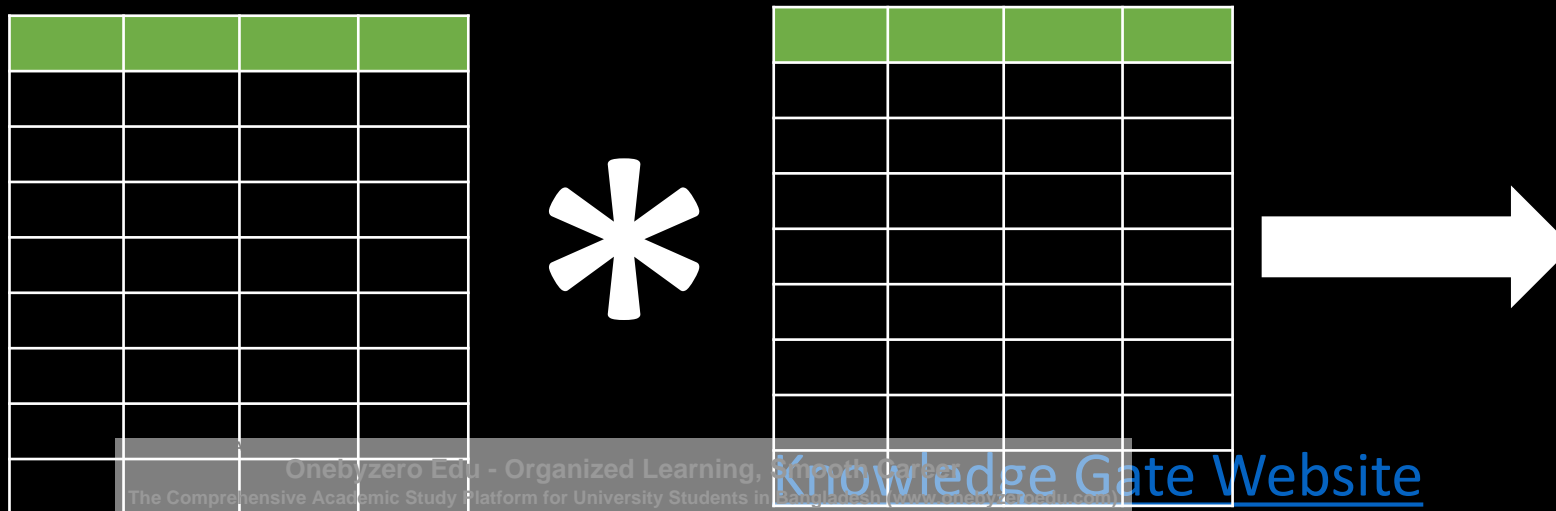
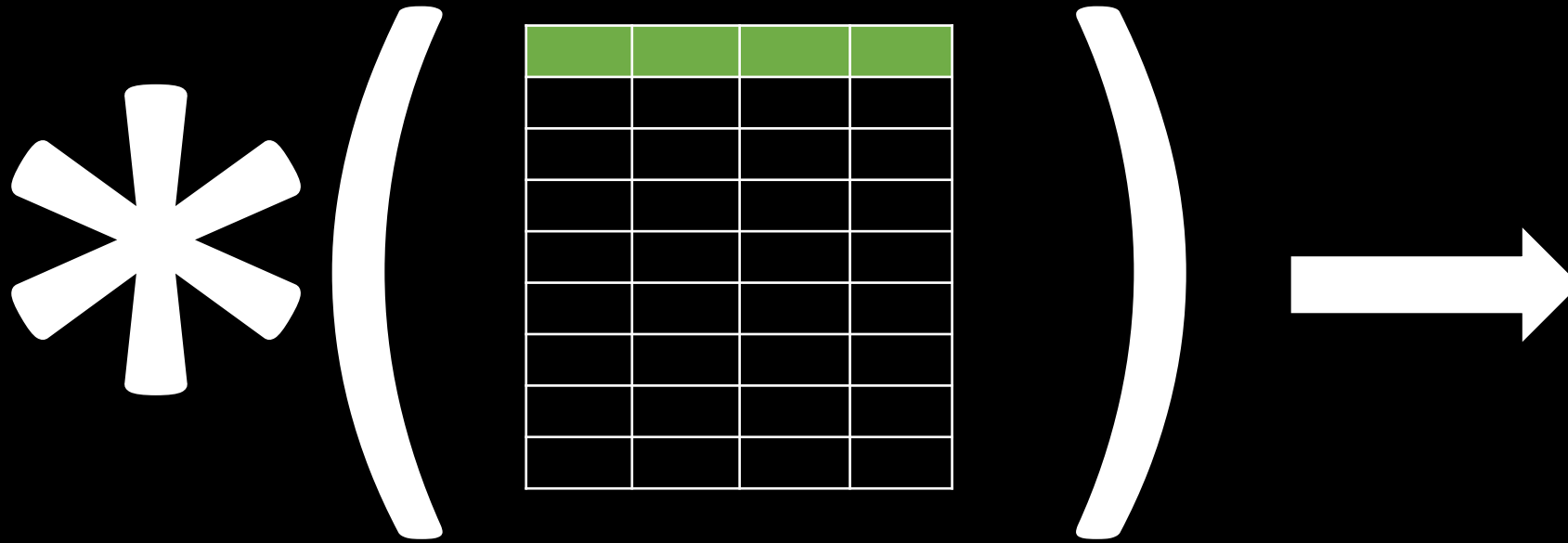
<u>Relational model</u>	<u>RDBMS</u>
RA, RC	SQL
Algo	Code
Conceptual	Reality
Theoretical	Practical

RELATIONAL ALGEBRA

- RA like any other mathematical system provides a number of operators and use relations (tables) as operands and produce a new relation as their result.



- Every operator in the RA accepts (one or two) relation/table as input arguments and returns always a single relation instance as the result without a name.



- It also does not consider duplicity by default as it is based on set theory. Same query is written in RA and SQL the result may be different as SQL considers duplication.
- As it is pure mathematics no use of English keywords. Operators are represented using symbols.

BASIC / FUNDAMENTAL OPERATORS

- The fundamental operations in the relational algebra are **select**, **project**, **union**, **set difference**, **Cartesian product**, and **Rename**.

Name	Symbol
Select	(σ)
Project	(Π)
Union	(\cup)
Set difference	$(-)$
Cross product	(\times)
Rename	(ρ)

DERIVED OPERATORS

- There are several other operations namely: **set intersection, natural join, and assignment.**

Name	Symbol	DERIVED FROM
Join	(\bowtie)	(X)
Intersection	(\cap)	$(-)$ $A \cap B = A - (A - B)$
Division	(\div)	$(X, -, \Pi)$
Assignment	$(=)$	

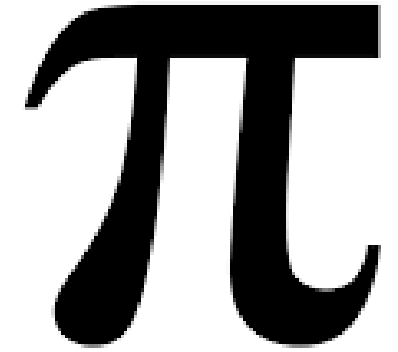
- The **select, project, and rename** operations are called **unary operations**, because they operate on one relation.
- **Union, Cartesian product and set difference** operate on pairs of relations and are, therefore, called **binary operations**.
- Relational algebra also provides the framework for query optimization.

Relational schema - A relation schema R , denoted by $R (A_1, A_2, \dots, A_n)$, is made up of a relation name R and a list of attributes, A_1, A_2, \dots, A_n . Each **attribute** A_i is the name of a role played by some domain D in the relation schema R . It is use to describe a Relation.

E.g. Schema representation of Table **Student** is as –
STUDENT (NAME, ID, CITY, COUNTRY, HOBBY).

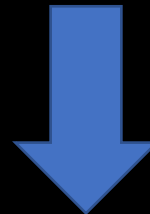
Relational Instance - Relations with its data at particular instant of time.

The Project Operation (Vertical Selection)



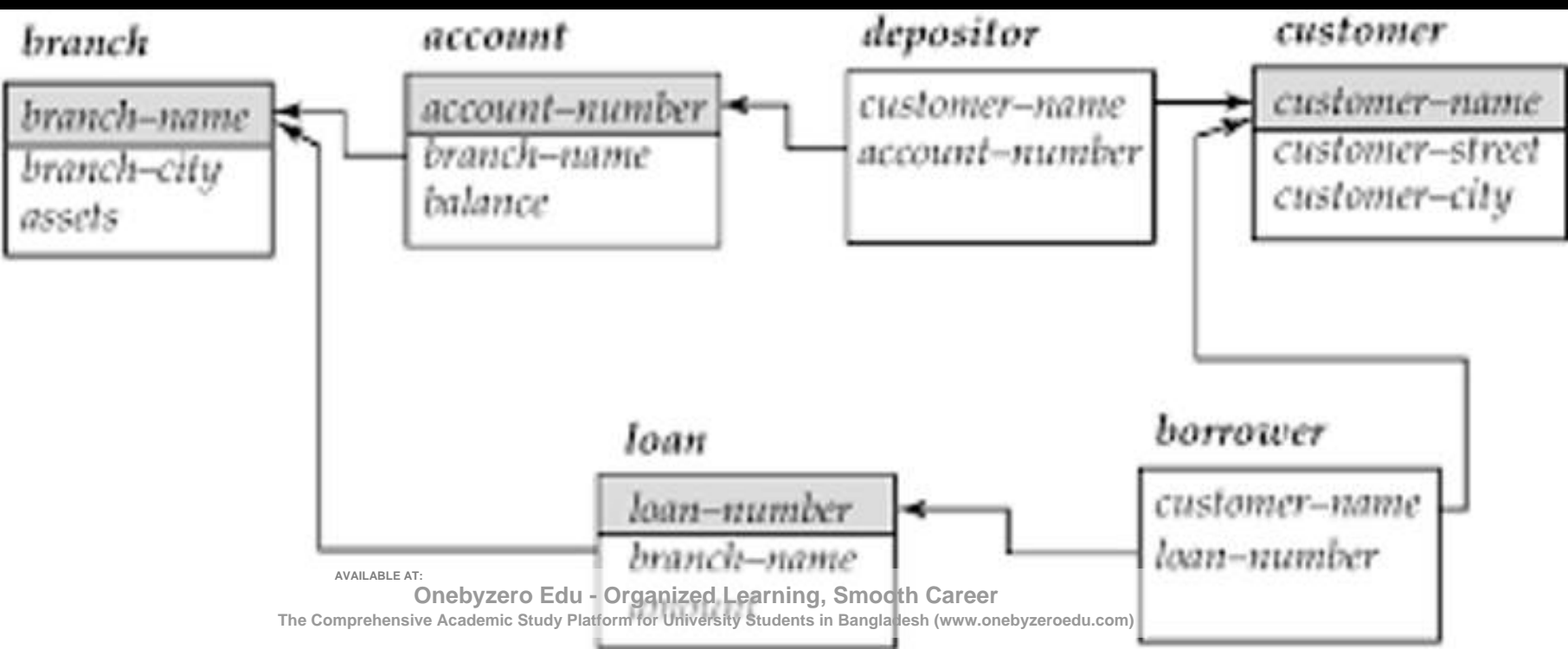
- Main idea behind project operator is to select desired columns
- The project operation is a unary operation that returns its argument relation, with certain attributes left out.
- Projection is denoted by the uppercase Greek letter pi (Π). Maximum columns selected can be 1, Maximum selected Columns can be $n - 1$.

- $\Pi_{\text{column_name}}(\text{table_name})$



Q Write a RELATIONAL ALGEBRA query to find the name of all customer without duplication having bank account?

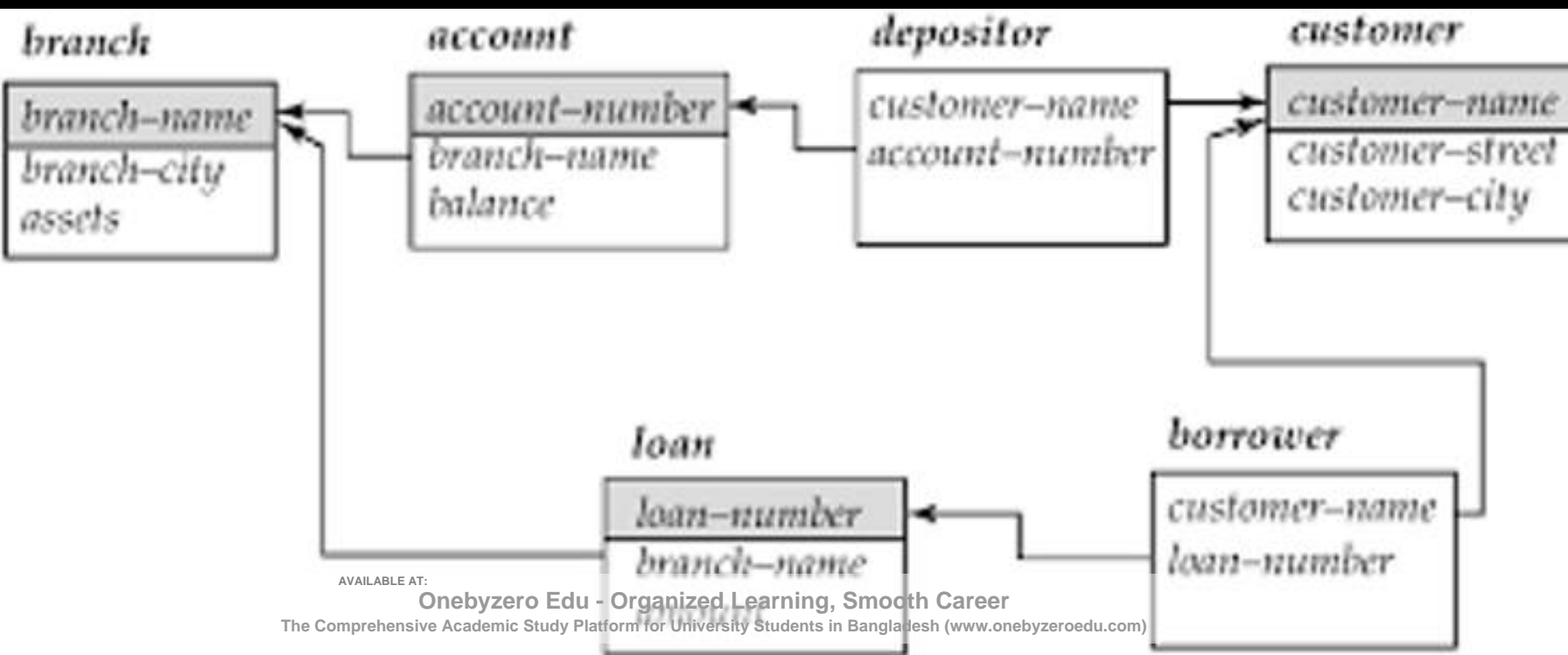
Q Write a RELATIONAL ALGEBRA query to find all the details of bank branches?



Q Write a RELATIONAL ALGEBRA query to find the name of all customer without duplication having bank account?

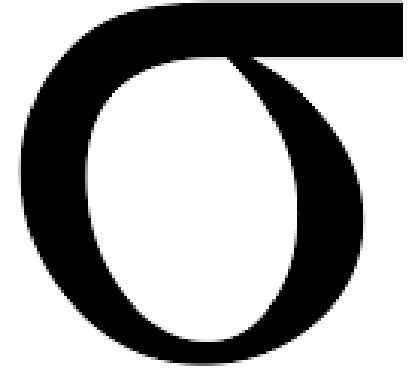
$\Pi_{\text{customer_name}}$ (depositor)

Q Write a RELATIONAL ALGEBRA query to find all the details of bank branches?
(branch)



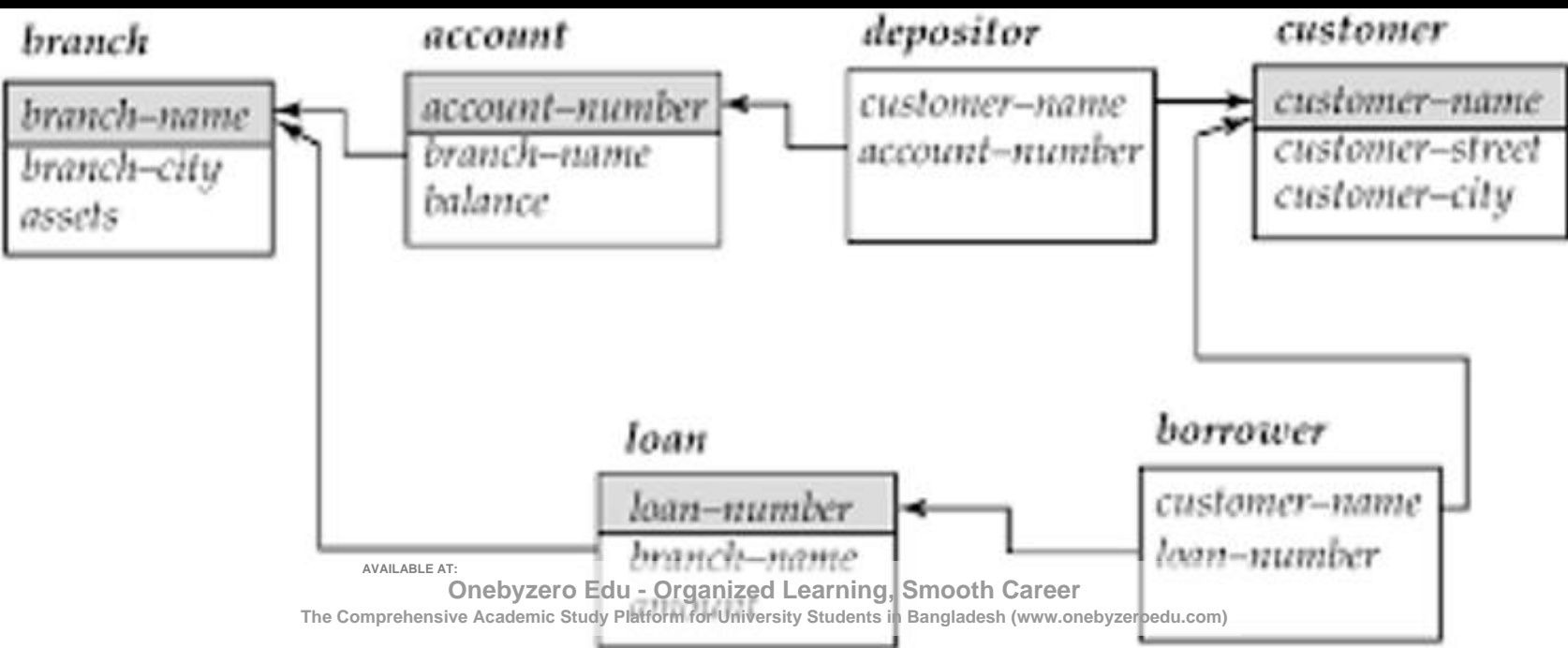
The Select Operation (Horizontal Selection)

- The select operation selects tuples that satisfy a given predicate/Condition p.
- Lowercase Greek letter sigma (σ) is used to denote selection.
- It is a unary operator.
- Eliminates only tuples/rows.
- $\sigma_{\text{condition}}(\text{table_name})$



Q Write a RELATIONAL ALGEBRA query to find all account_no where balance is less the 1000?

Q Write a RELATIONAL ALGEBRA query to find branch name which is situated in Delhi and having assets less than 1,00,000?

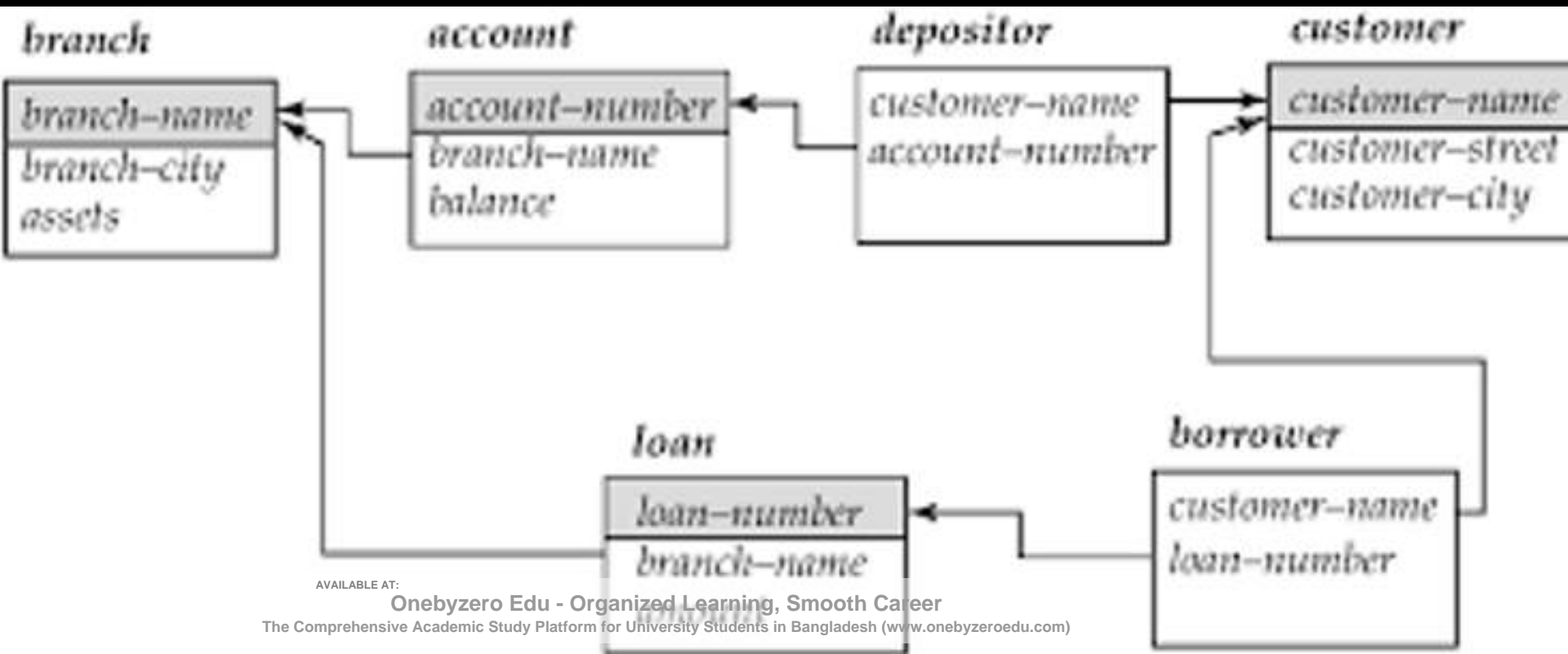


Q Write a RELATIONAL ALGEBRA query to find all account_no where balance is less the 1000?

$\Pi_{\text{account_number}}(\sigma_{\text{balance} < 1000}(\text{account}))$

Q Write a RELATIONAL ALGEBRA query to find branch name which is situated in Delhi and having assets less than 1,00,000?

$\Pi_{\text{branch_name}}(\sigma_{(\text{branch_city} = \text{delhi}) \wedge (\text{assets} < 1000)}(\text{branch}))$

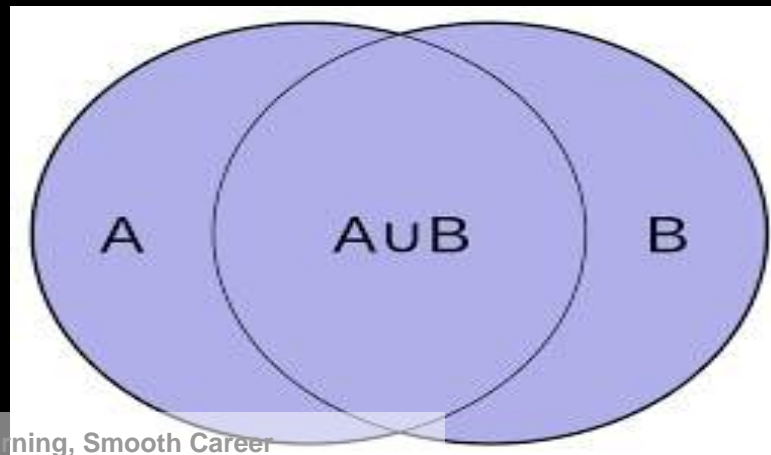
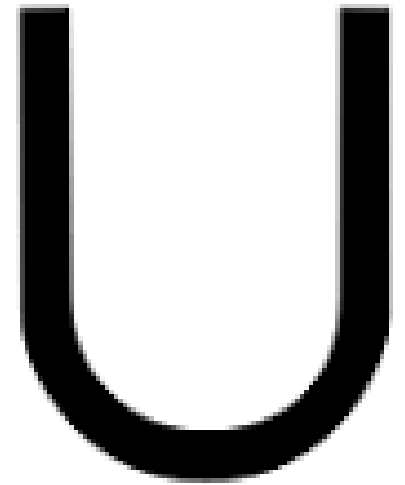


- Commutative in Nature, $\sigma_{p1 \wedge p2}(r) = \sigma_{p2 \wedge p1}(r) = \sigma_{p1}(\sigma_{p2}(r)) = \sigma_{p2}(\sigma_{p1}(r))$
- We allow comparisons using $=$, \neq , $<$, $>$, \leq and \geq in the selection predicate.
- Using the connectives and (\wedge), or (\vee), and not (\neg), we can combine several predicates into a larger predicate.
- Minimum number of tuples selected can be 0, Maximum selected tuples can be all.



The Union Operation

- It is a binary operation, denoted, as in set theory, by \cup .
- Written as, $\text{Expression}_1 \cup \text{Expression}_2$, $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- For a union operation $r \cup s$ to be valid, we require that two conditions are satisfied:
 - The relations r and s must be of the same arity. That is, they must have the same number of attributes.
 - The domains of the i^{th} attribute of r and the i^{th} attribute of s must be the same, for all i .

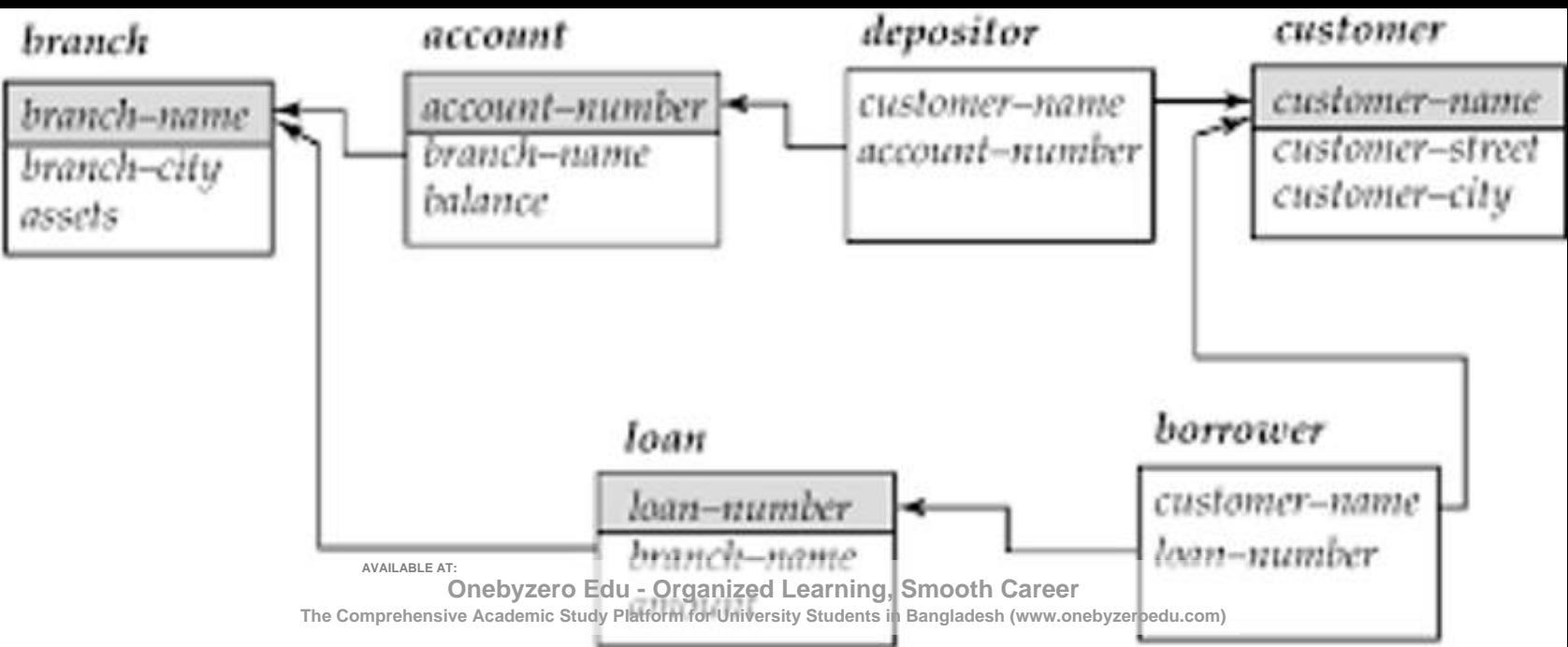


- Some points to remember

- $\text{Deg}(R \cup S) = \text{Deg}(R) = \text{Deg}(S)$
- $\text{Max}(IRI, ISI) \leq IRUSI \leq (IRI + ISI)$

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan or an account or both?

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan but do not have an account?

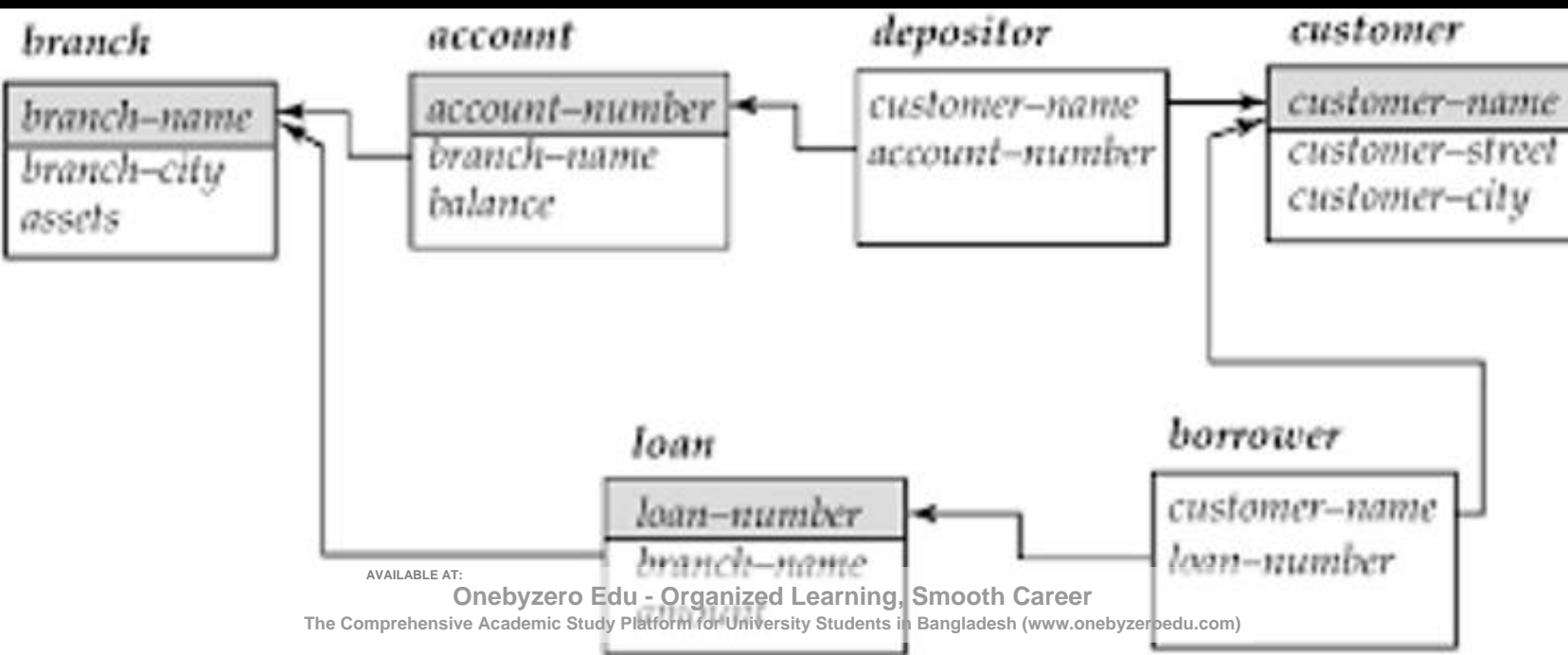


Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan or an account or both?

$\Pi_{\text{customer_name}}(\text{depositor}) \cup \Pi_{\text{customer_name}}(\text{borrower})$

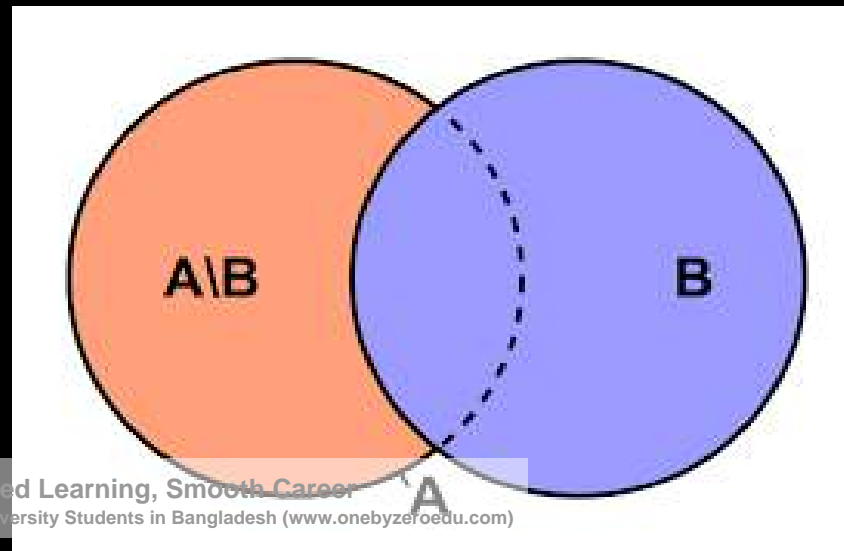
Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan but do not have an account?

$\Pi_{\text{customer_name}}(\text{borrower}) - (\Pi_{\text{customer_name}}(\text{depositor}))$



The Set-Difference Operation

- The set-difference operation, denoted by $-$, allows us to find tuples that are in one relation but are not in another. It is a binary operator.
- The expression $r - s$ produces a relation containing those tuples in r but not in s .
- We must ensure that set differences are taken between compatible relations.
- For a set-difference operation $r - s$ to be valid, we require that the relations r and s be of the same arity, and that the domains of the i th attribute of r and the i th attribute of s be the same, for all i .
 - $0 \leq |R - S| \leq |R|$



The Cartesian-Product Operation

- The Cartesian-product operation, denoted by a cross (\times), allows us to combine information from any two relations.
- It is a binary operator; we write the Cartesian product of relations R_1 and R_2 as $R_1 \times R_2$.
- Cartesian-product operation associates every tuple of R_1 with every tuple of R_2 .
 - $R_1 \times R_2 = \{rs \mid r \in R_1 \text{ and } s \in R_2\}$, contains one tuple $\langle r, s \rangle$ (concatenation of tuples r and s) for each pair of tuples $r \in R_1, s \in R_2$.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \times R_2$			
A	$R_1.B$	$R_2.B$	C

The Cartesian-Product Operation

- The Cartesian-product operation, denoted by a cross (\times), allows us to combine information from any two relations.
- It is a binary operator; we write the Cartesian product of relations R_1 and R_2 as $R_1 \times R_2$.
- Cartesian-product operation associates every tuple of R_1 with every tuple of R_2 .
 - $R_1 \times R_2 = \{rs \mid r \in R_1 \text{ and } s \in R_2\}$, contains one tuple $\langle r, s \rangle$ (concatenation of tuples r and s) for each pair of tuples $r \in R_1, s \in R_2$.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \times R_2$			
A	$R_1.B$	$R_2.B$	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

The Cartesian-Product Operation

- The Cartesian-product operation, denoted by a cross (\times), allows us to combine information from any two relations.
- It is a binary operator; we write the Cartesian product of relations R_1 and R_2 as $R_1 \times R_2$.
- Cartesian-product operation associates every tuple of R_1 with every tuple of R_2 .
 - $R_1 \times R_2 = \{rs \mid r \in R_1 \text{ and } s \in R_2\}$, contains one tuple $\langle r, s \rangle$ (concatenation of tuples r and s) for each pair of tuples $r \in R_1, s \in R_2$.

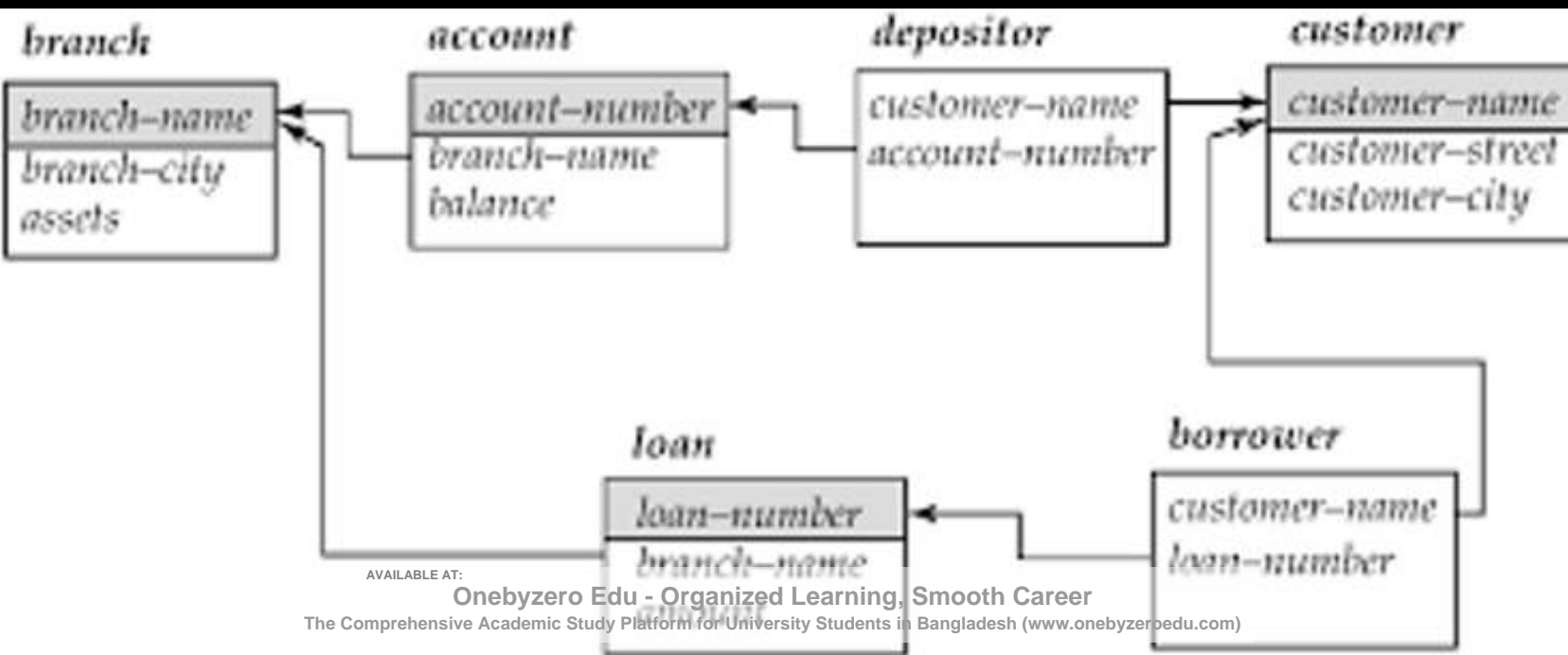
R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \times R_2$			
A	$R_1.B$	$R_2.B$	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

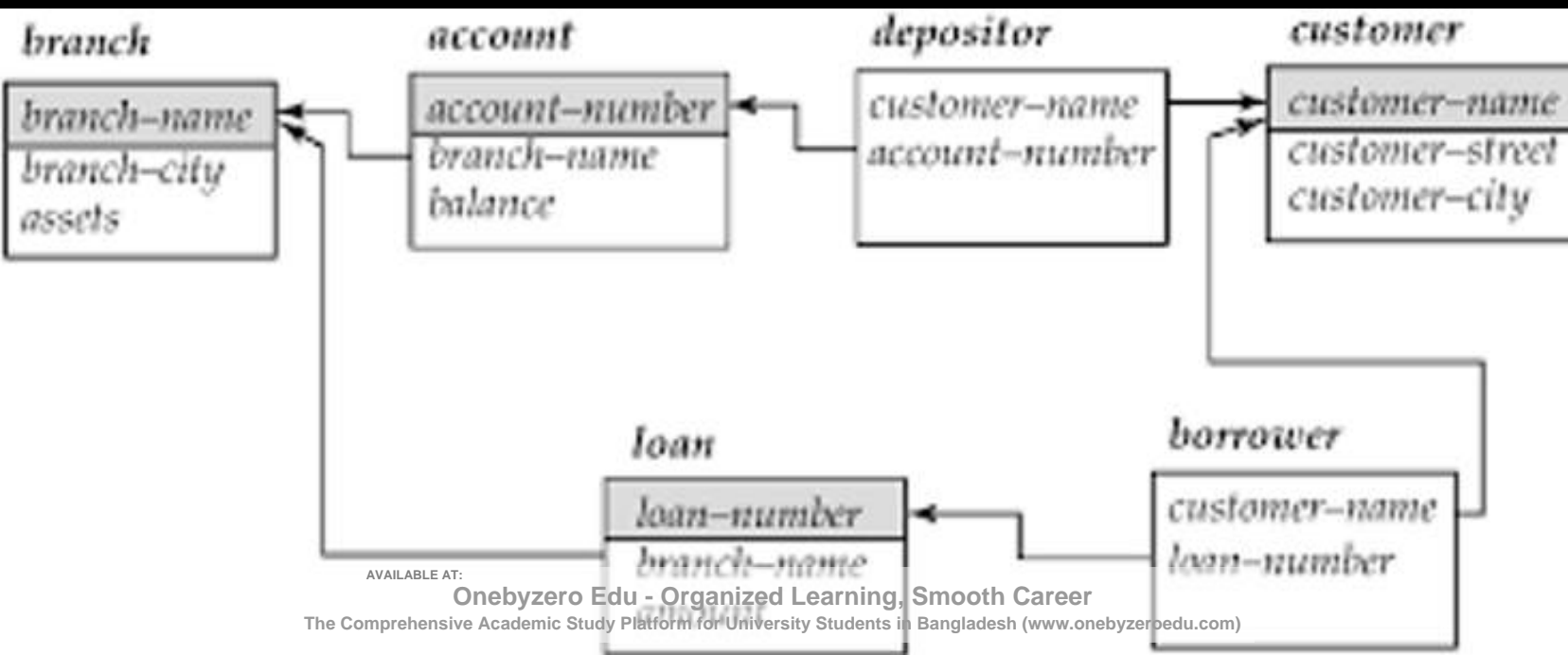
- $R_1 \times R_2$ returns a relational instance whose schema contains all the fields of R_1 (in order as they appear in R_1) and all fields of R_2 (in order as they appear in R_2).
- If R_1 has m tuples and R_2 has n tuples the result will be having $= m * n$ tuples.
- Same attribute name may appear in both R_1 and R_2 , we need to devise a naming schema to distinguish between these attributes.

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?



Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?

$\Pi_{\text{customer_name, balance}} (\sigma_{\text{account.account_number}=\text{depositor.account_number}} (\text{account X depositor}))$



Rename Operation

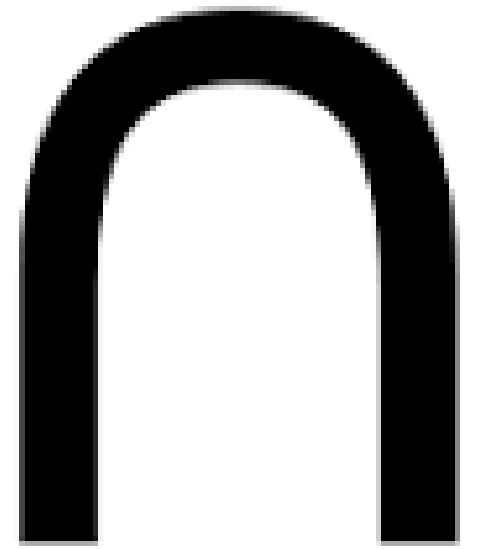
- The results of relational algebra are also relations but without any name. This Query do not change the name of the table in the original data base, but create a new copy of the table.
- The rename operation allows us to rename the output relation. It is denoted with small Greek letter **rho** ρ . Where the result of expression **E** is saved with name of **x**.
- $\rho_{x(A1, A2, A3, A4, \dots, AN)}(E)$
- $\rho_{\text{Learner}}(\text{Student})$
- $\rho_{\text{Learner}(\text{Stu_ID, User_Name, Age})}(\text{Student}(\text{Roll_No, Name, Age}))$

Additional/Derived Relational-Algebra Operations

- If we restrict ourselves to just the fundamental operations, certain common queries are lengthy to express. Therefore, we use additional operations.
- These additional operations do not add any power to the algebra.
- They are used to simplify the queries.

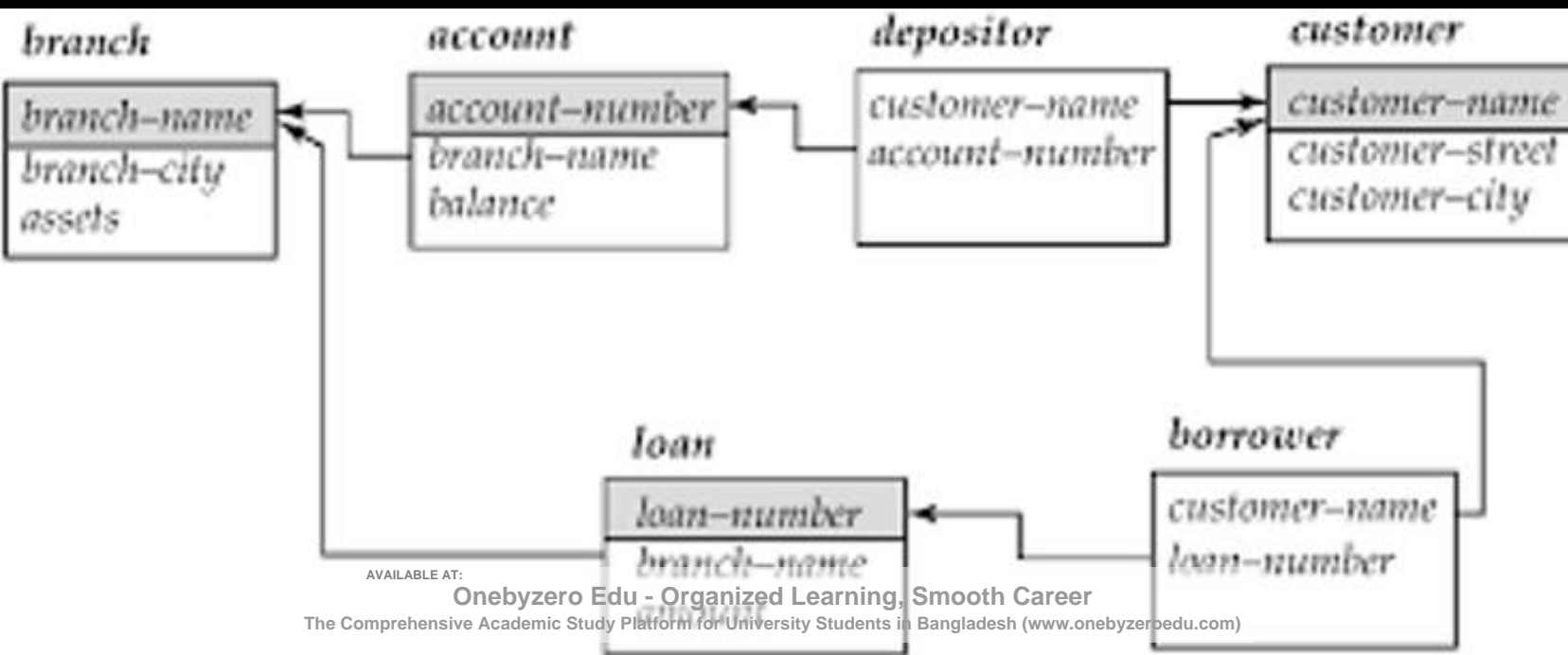
Set-Intersection Operat

- We will be using \cap symbol to denote set intersection.
- $r \cap s = r - (r - s)$
- Set intersection is not a fundamental operation in relational algebra.
- $r \cap s = \{t \mid t \in r \text{ and } t \in s\}$
- $0 \leq |R \cap S| \leq \min(|R|, |S|)$



Q Write a RELATIONAL ALGEBRA query to find all the customer name who have both a loan and an account?

$\Pi_{\text{customer_name}}(\text{depositor}) \cup \Pi_{\text{customer_name}}(\text{borrower})$



The Natural-Join Operation *

- The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.

he natural join is a Lossy operator.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \bowtie R_2$		
A	B	C
1	P	X
2	Q	X
3	R	Y

- In general, the DIVISION operation is applied when we have query like student who have completed both database1 and data base2 tasks.

Completed		DBProject
Student	Task	Task
Fred	Database1	Database1
Fred	Database2	Database2
Fred	Compiler1	
Eugene	Database1	
Eugene	Compiler1	
Sarah	Database1	
Sarah	Database2	

$$\pi_{\text{Student}}(R) - \{\pi_{\text{Student}}[(\pi_{\text{Student}}(R) \times S) - \pi_{\text{Student,Task}}(R)]\}$$

Completed

Student	Task
Fred	Database1
Fred	Database2
Fred	Compiler1
Eugene	Database1
Eugene	Compiler1
Sarah	Database1
Sarah	Database2

DBProject

Task
Database1
Database2

Completed

÷

DBProject

Student
Fred
Sarah

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

S
A
a1
a2
a3

T
B
b1
b4

AVAILABLE AT:

Introduction to SQL

- Structured Query Language is a domain-specific language (not general purpose) used in programming and design for managing data held in a relational database management system (RDBMS).
- Although we refer to the SQL language as a “query language,” it can do much more than just query a database. It can define the structure of the data base, modify data in the database, specify security constraints and number of other tasks.
- Originally based upon relational algebra(procedural) and tuple relational calculus (Non-procedural) mathematical model.

Overview of the SQL Query Language

1. IBM developed the original version of SQL, originally called Sequel (*Structured English Query Language*), as part of the System R project in the early 1970s.
2. The Sequel language has evolved since then, and its name has changed to SQL (Structured Query Language) (some other company has trademark on the word sequel). SQL has clearly established itself as *the* standard relational database language.
3. In 1986, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) published an SQL standard, called SQL-86.
4. The next version of the standard was SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL: 2016, SQL: 2019 and most recently SQL:2023.

Classification of database languages

1. Data Definition Language (DDL) :

1. a. DDL is set of SQL commands used to create, modify and delete database structures but not data.
2. b. They are used by the DBA to a limited extent, a database designer, or application developer.
3. c. Create, drop, alter, truncate are commonly used DDL command. CREATE, ALTER, DROP, TRUNCATE, COMMENT, GRANT, REVOKE statement

2. Data Manipulation Language (DML) :

1. a. A DML is a language that enables users to access or manipulates data as organized by the appropriate data model.
2. b. There are two types of DMLs :
 1. i. Procedural DMLs : It requires a user to specify what data are needed and how to get those data.
 2. ii. Declarative DMLs (Non-procedural DMLs) : It requires a user to specify what data are needed without specifying how to get those data.
3. c. Insert, update, delete, query are commonly used DML commands. INSERT, UPDATE, DELETE statement

3. Data Control Language (DCL) :

1. a. It is the component of SQL statement that control access to data and to the database.
2. b. Commit, rollback command are used in DCL. GRANT and REVOKE statement

4. Data Query Language (DQL) :

1. a. It is the component of SQL statement that allows getting data from the database and imposing ordering upon it.
2. b. It includes select statement. SELECT statement

5. View Definition Language (VDL) :

1. 1. VDL is used to specify user views and their mapping to conceptual schema.
2. 2. It defines the subset of records available to classes of users.
3. 3. It creates virtual tables and the view appears to users like conceptual level.
4. 4. It specifies user interfaces. SQL is a DML language.

```
CREATE TABLE table_name (  
    column1 data_type [constraints],  
    column2 data_type [constraints],  
    column3 data_type [constraints],  
    ...  
);
```

```
CREATE TABLE Students (  
    StudentID INT PRIMARY KEY,  
    FirstName VARCHAR(50),  
    LastName VARCHAR(50),  
    Age INT,  
    Email VARCHAR(100)  
);
```

- list of some common data types supported by SQL along with a brief description of each:
- Numeric Data Types:
 - 1. `INT`: For storing integer values.
 - 2. `SMALLINT`: A smaller range of integers compared to INT.
 - 3. `BIGINT`: For storing larger integers.
 - 4. `DECIMAL(p, s)`: For storing exact numerical values, where `p` is the precision and `s` is the scale.
 - 5. `FLOAT`: For storing floating-point numbers.
 - 6. `REAL`: A data type that can store floating-point numbers, generally with less precision compared to FLOAT.
- String Data Types:
 - 7. `VARCHAR(n)`: Variable-length character string, where `n` is the maximum length.
 - 8. `CHAR(n)`: Fixed-length character string, where `n` is the length.
 - 9. `TEXT`: For storing long text strings.

Adding a New Column

```
ALTER TABLE Employees  
ADD PhoneNumber VARCHAR(15);
```

Dropping a Column

```
ALTER TABLE Employees  
DROP COLUMN PhoneNumber;
```

Modifying an Existing Column

```
ALTER TABLE Employees  
MODIFY COLUMN PhoneNumber VARCHAR(20);
```

```
ALTER TABLE Employees  
ALTER COLUMN PhoneNumber VARCHAR(20);
```

Renaming a Column

```
ALTER TABLE Employees  
RENAME COLUMN PhoneNumber TO ContactNumber;
```

Renaming a table

```
ALTER TABLE Employees  
RENAME TO Staff;
```

```
DROP TABLE table_name;
```

```
CREATE TABLE Orders (  
    OrderID INT PRIMARY KEY,  
    CustomerID INT,  
    OrderDate DATE,  
    FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)  
);
```

```
ALTER TABLE Orders  
ADD FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);
```

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
INSERT INTO Students (StudentID, FirstName, LastName, Age)
VALUES
(1, 'Amit', 'Sharma', 20),
(2, 'Priya', 'Gupta', 22),
(3, 'Ravi', 'Kumar', 19);
```

```
DELETE FROM table_name  
WHERE condition;
```

```
DELETE FROM Students  
WHERE StudentID = 1;
```

```
DELETE FROM table_name;
```

Basic Structure of SQL Queries

- For any SQL as query, input and output both are relations. Number of relations inputs to a query will be at least one, but output will always be a single relation without any name unless specified, but columns will have names from input tables.
- The basic structure of an SQL query consists of three clauses: select, from, and where. The query takes it's input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result without any name unless specified.
- A typical SQL query has the form.

Select A_1, A_2, \dots, A_n	(Column name)
from r_1, r_2, \dots, r_m	(Relation/table name)
Where P;	(Condition)

Select A_1, A_2, \dots, A_n	(Column name)
from r_1, r_2, \dots, r_m	(Relation/table name)
Where P ;	(Condition)

1. It is to be noted only select and from are mandatory clauses, and if not required then it is not essential to write where. If the **where** clause is omitted, the predicate P is **true**.
2. SQL in general is not case sensitive i.e. it doesn't matter whether we write query in upper or lower case.
3. In practice, duplicate elimination is time-consuming. Therefore, SQL allows duplicates in relations as well as in the results of SQL expressions. In those cases where we want to force the elimination of duplicates, we insert the keyword **distinct** after **select**, will discuss in detail later. SQL allows us to use the keyword **all** to specify explicitly that duplicates are not removed, Since duplicate retention is the default, we shall not use **all** in our examples.

Select Clause

- The function of Select clause in SQL is more or less same as that of ' Π ' projection in the relational algebra. It is used to pick the column required in result of the query out of all the columns in relation/table. (Vertical filtering)
 - **Select A_1, A_2, \dots, A_n (Column name)**
- We can use '*' to specify that we need all columns
 - **Select ***
- The **select** clause may also contain arithmetic expressions involving the operators +, -, / and * operating on constants or attributes of tuples. however, that it does not result in any change to the relation/table.

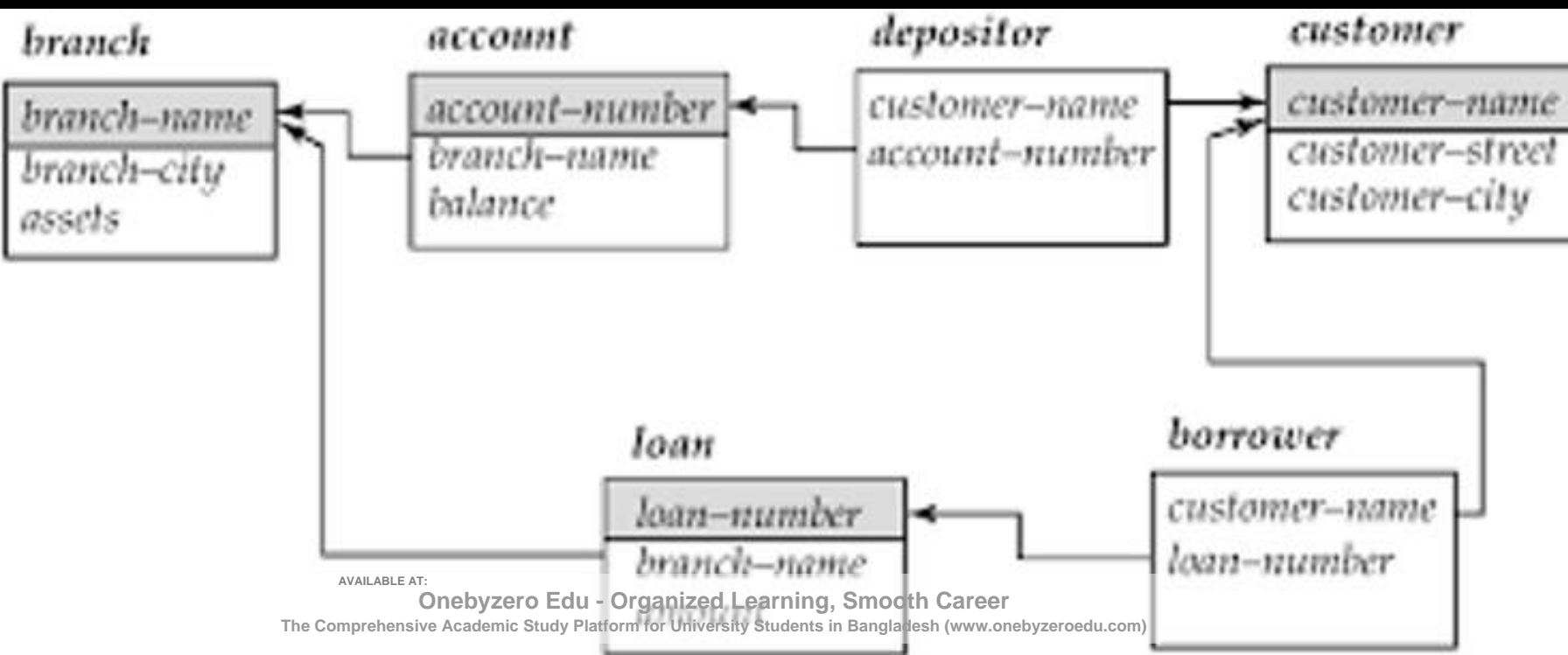


Q Write a SQL query to find all the details of bank branches?

Q Write a SQL query to find each loan number along with loan amount?

Q Write a SQL query to find the name of all customer without duplication having bank account?

Q Write a SQL query to find all account_no and balance with 6% yearly interest added to it?



Q find all the details of bank branches?

Select *
from branch

Q find each loan number along with loan amount?

Select loan_number, amount
from loan

Q find the name of all customer without duplication having bank account?

Select distinct customer_name
from depositor

Q find all account_no and balance with 6% yearly interest added to it?

Select account_number, balance*1.06
from account

Select Clause with where clause

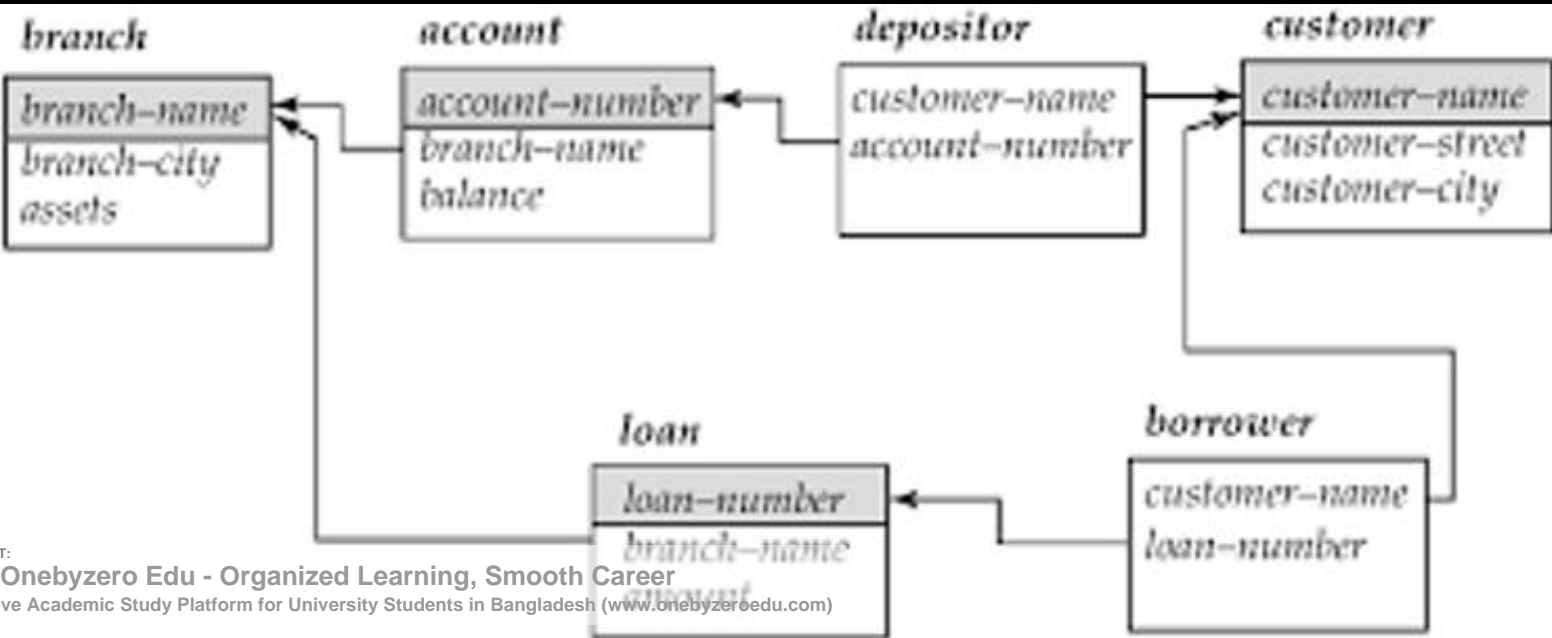
1. Where clause in SQL is same as ' σ ' sigma of relational algebra where we specify the conditions/Predicate (horizontal filtering).
2. Where clause can have expressions involving the comparison operators $<$, $<=$, $>$, $>=$, $=$ and \neq . SQL allows us to use the comparison operators to compare strings and arithmetic expressions.
3. SQL allows the use of the logical connectives **and**, **or**, and **not** in the **where** clause.
4. SQL includes a **between** comparison operator to simplify **where** clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.
5. Similarly, we can use the **not between** comparison operator.



Q Write a SQL query to find all account_no where balance is less the 1000?

Q Write a SQL query to find branch name which is situated in Delhi and having assets less than 1,00,000?

Q Write a SQL query to find branch name and account_no which has balance greater than equal to 1,000 but less than equal to 10,000?



Q find all account_no where balance is less the 1000?

```
Select account_number  
from account  
Where balance < 1000
```

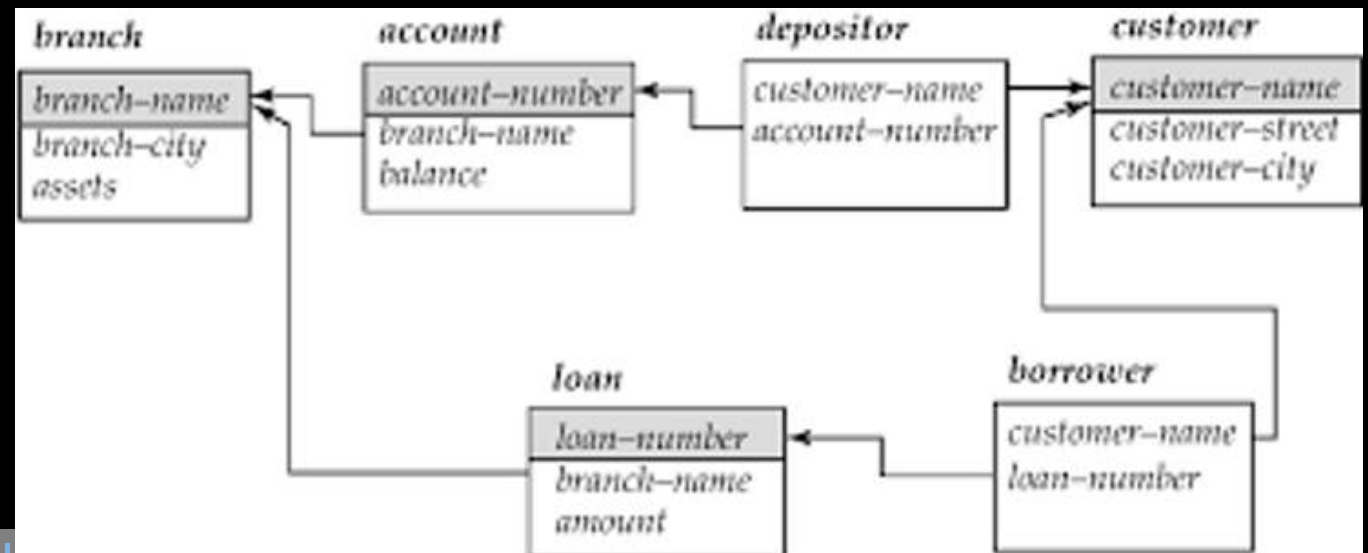
Q Write a SQL query to find branch name which is situated in Delhi and having assets less than 1,00,000?

```
Select branch_name  
from branch  
Where branch_city = 'delhi' and assets < 1,00,000
```

Q Write a SQL query to find branch name and account_no which has balance greater than equal to 1,000 but less than equal to 10,000?

```
Select branch_name, account_number  
from account  
Where balance between 1000 and 10000
```

```
Select branch_name, account_number  
from account  
Where balance >= 1000 and balance<=10000
```



Set Operation

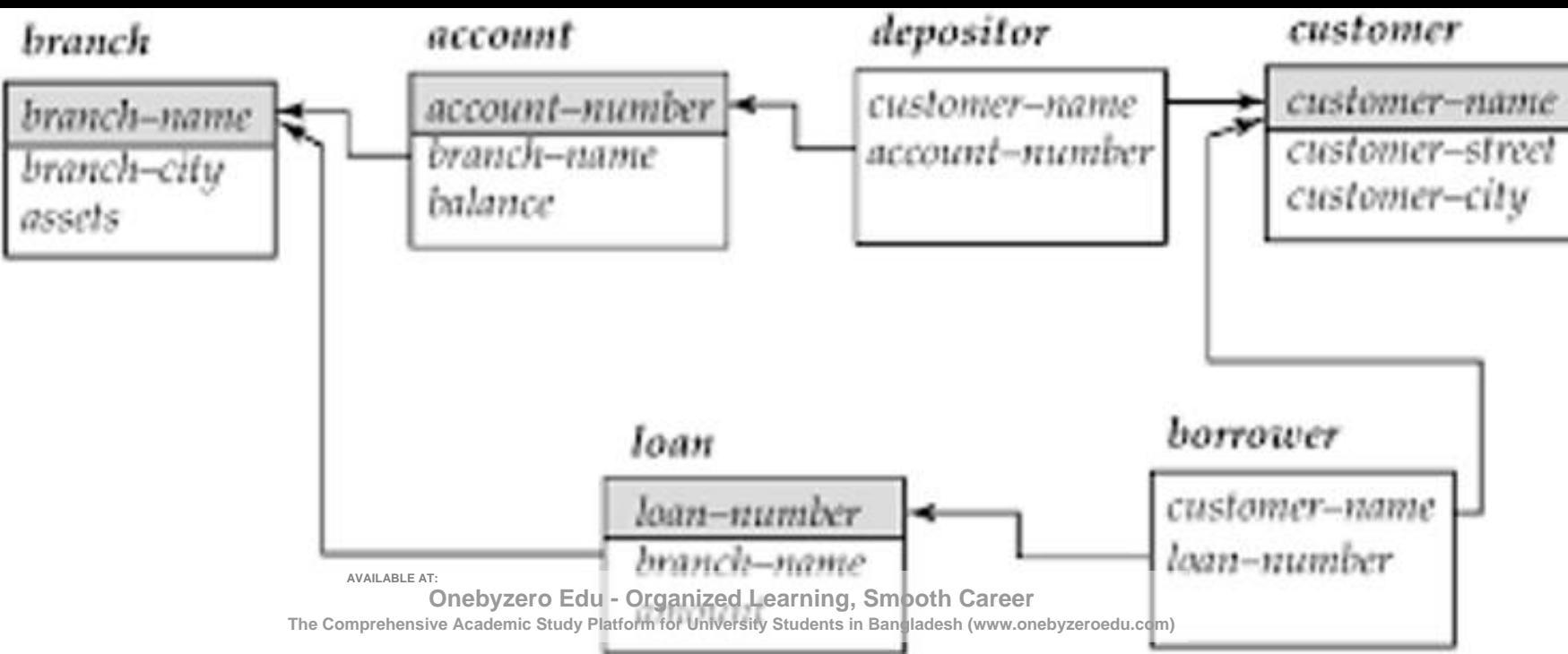
1. The SQL operations **union**, **intersect**, and **except/minus** operate on relations and corresponds to the mathematical set-theory operations \cup , \cap and $-$ respectively.
2. The **union** operation automatically eliminates duplicates, unlike the **select** clause, If we want to retain all duplicates, we must write **union all** in place of **union**.
3. The **intersect** operation automatically eliminates duplicates. If we want to retain all duplicates, we must write **intersect all** in place of intersect.
4. If we want to retain duplicates, we must write **except all** in place of **except**.

Q Write a SQL query to find all the customer name

a) who have a loan or an account or both ?

b) who have both a loan and an account?

c) who have a loan but do not have an account?



Q Write a SQL query to find all the customer name
a) who have a loan or an account or both ?

Select customer_name
From depositor

Union

Select customer_name
From borrower

b) who have both a loan and an account?

Select customer_name
From depositor

intersect

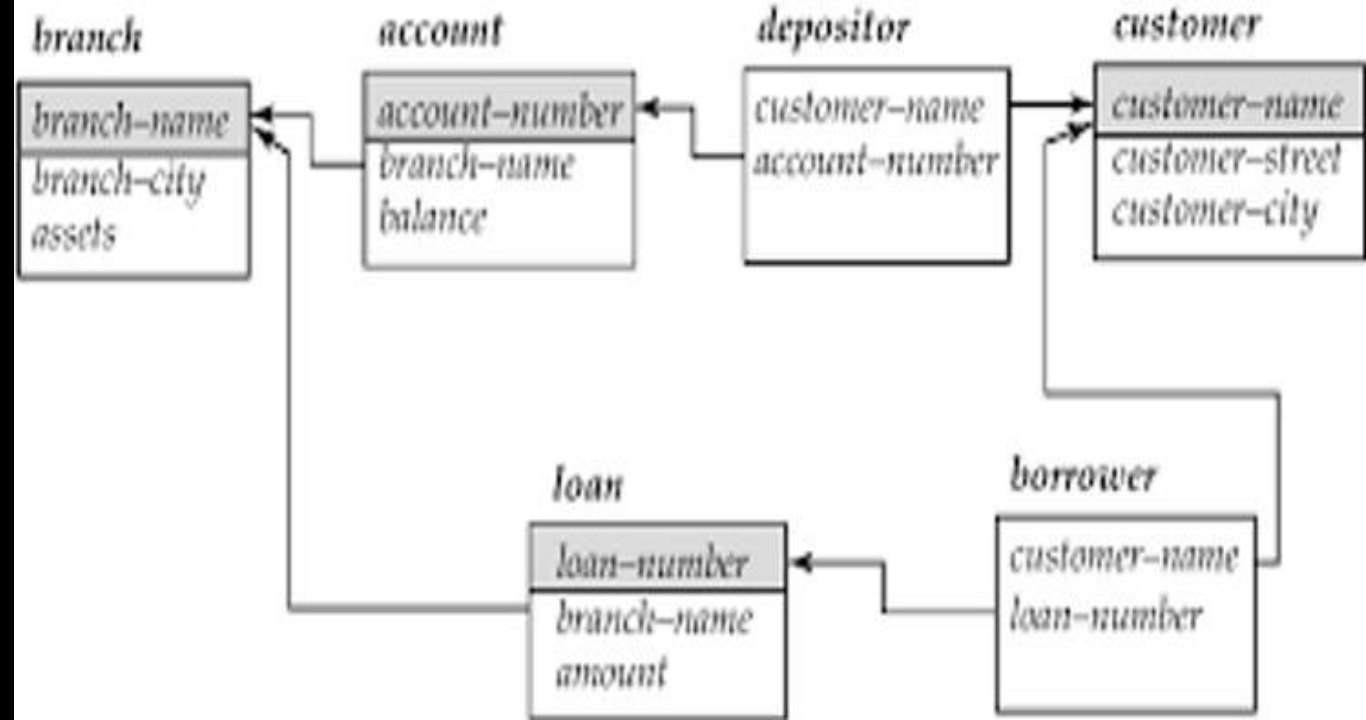
Select customer_name
From borrower

c) who have a loan but do not have an account?

Select customer_name
From borrower

Except

Select customer_name
From depositor



Queries on Multiple Relations

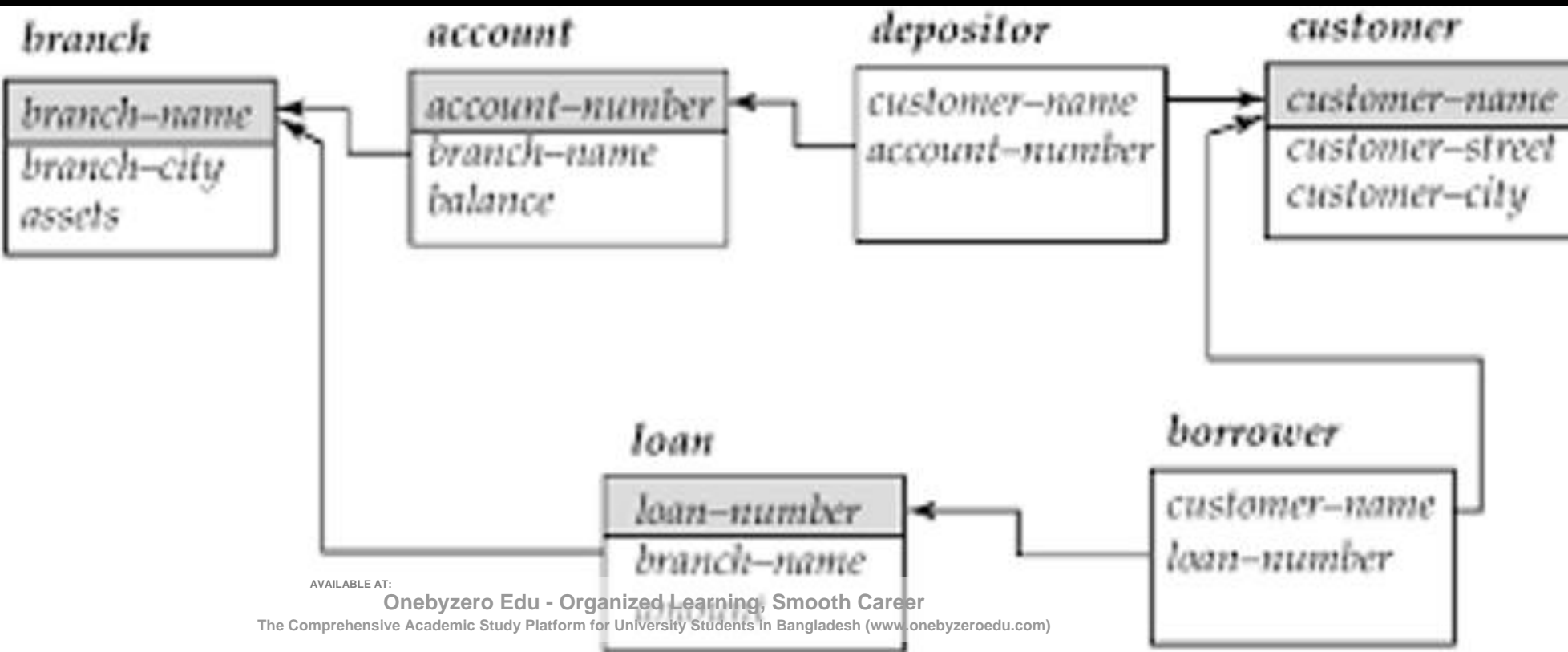
- The **from** clause by itself defines a Cartesian product of the relations listed in the clause. Cartesian product of two relations, which concatenates each tuple of the first relation with every tuple of the second
- Since the same attribute name may appear in both r_1 and r_2 , we prefix the name of the relation from which the attribute originally came, before the attribute name. For those attributes that appear in only one of the two schemas, we shall usually drop the relation-name prefix. This simplification does not lead to any ambiguity.
- Cartesian Product is commutative in nature

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \times R_2$			
A	$R_1.B$	$R_2.B$	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?



AVAILABLE AT:

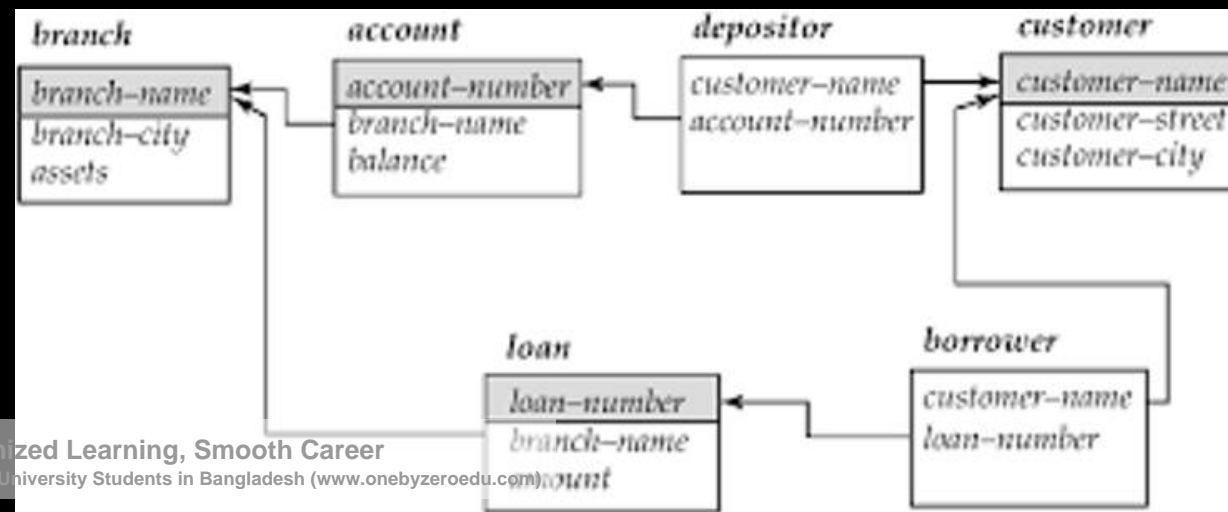
Onebyzero Edu - Organized Learning, Smooth Career
The Comprehensive Academic Study Platform for University Students in Bangladesh (www.onebyzeroedu.com)

Q find the name of all the customers with account balance, who have an account in the bank?

Select customer_name, balance

From account, depositor

Where account.account_number = depositor.account_number



Natural Join

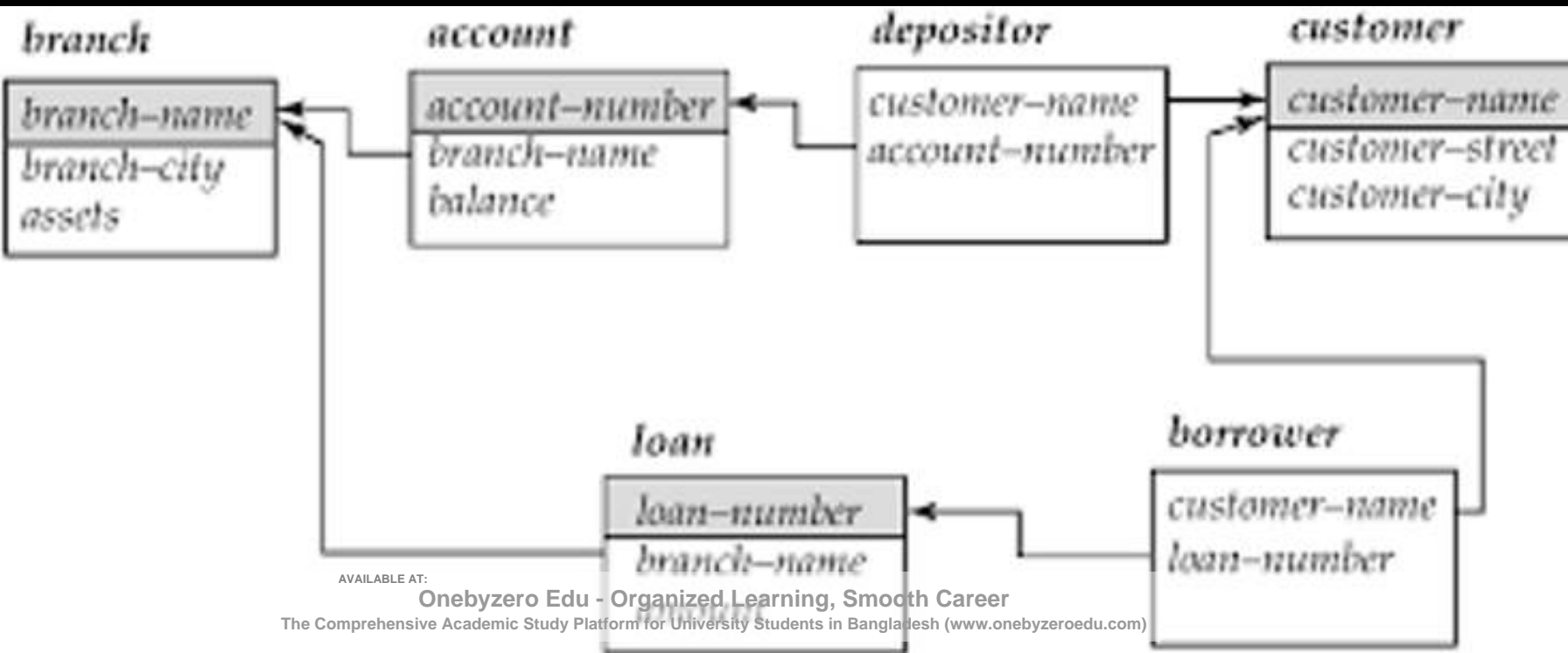
- To make the life of an SQL programmer easier for this common case, SQL supports an operation called the *natural join*. The **natural join** operation like cartesian product operates on two relations and produces a relation as the result.
- Natural join considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations. Notice that we do not repeat those attributes that appear in the schemas of both relations; rather they appear only once.
- Notice also the order in which the attributes are listed: first the attributes common to the schemas of both relations, second those attributes unique to the schema of the first relation, and finally, those attributes unique to the schema of the second relation. Commutative in nature.

R ₁	
A	B
1	P
2	Q
3	R

R ₂	
B	C
Q	X
R	Y
S	Z

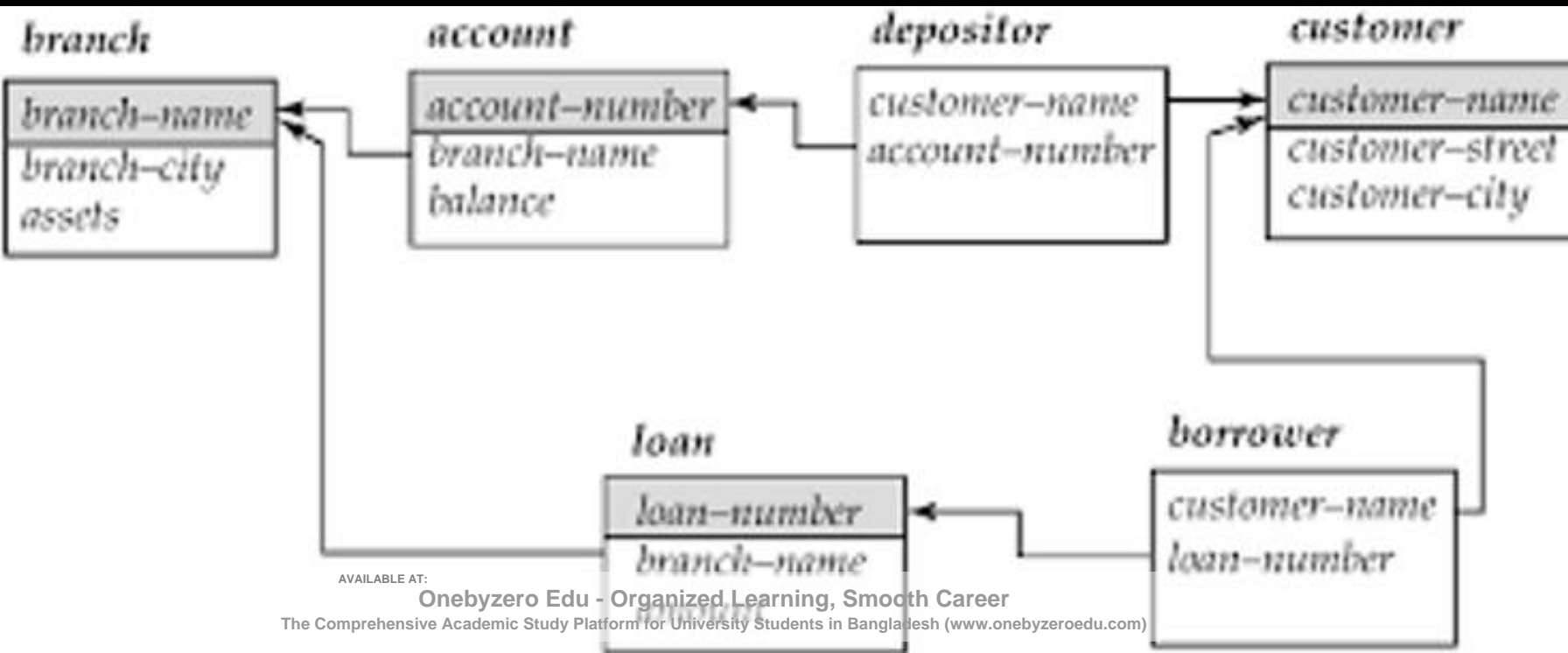
R ₁ ⋈ R ₂		
A	B	C
1	P	
2	Q	X
3	R	Y

Q Write a SQL query to find the name of all the customers along with account balance, who have an account in the bank?



Q Write a SQL query to find the name of all the customers along with account balance, who have an account in the bank?

Select customer_name, balance
From account natural join depositor



Outer Join

- The problem with natural join or join or inner join is only those values that appears in both relations will manage to reach final table, but if some value is explicitly in table one or in second table then that information will be lost, and that will be loss of information.
- The outer join operation works in a manner similar to the join operations we have already studied, but preserve those tuples that would be lost in a join, by creating tuples in the result containing null values.
- There are in fact three forms of outer join:
 - The **left outer join** (left join) preserves tuples only in the relation named before (to the left of) the left outer join operation.
 - The **right outer join** (right join) preserves tuples only in the relation named after (to the right of) the right outer join operation.
 - The **full outer join** preserves tuples in both relations.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 * R_2$			
A	$R_1.B$	$R_2.B$	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

$R_1 \bowtie R_2$		
A	B	C
2	Q	X
3	R	Y

$R_1 \bowtie R_2$		
A	B	C
1	P	null
2	Q	X
3	R	Y

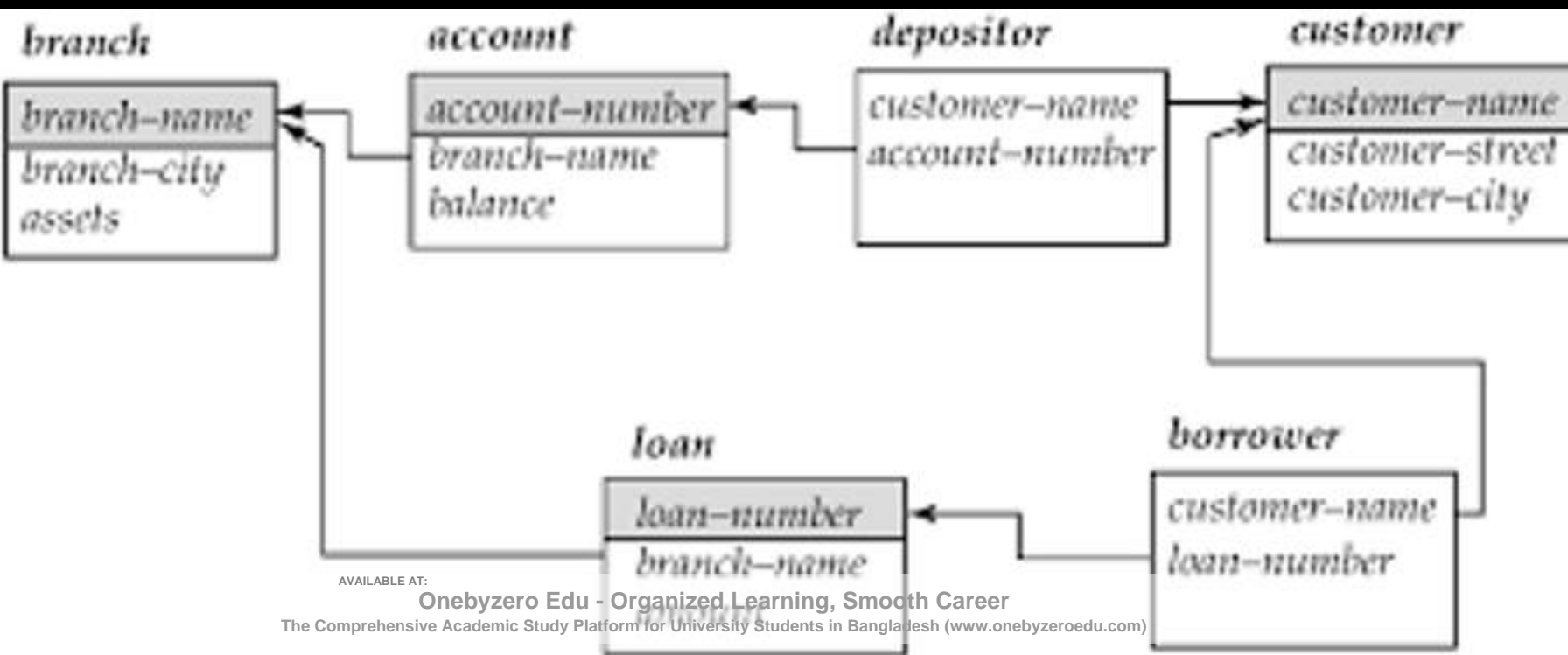
$R_1 \bowtie R_2$		
A	B	C
2	Q	X
3	R	Y
null	S	Z

$R_1 \bowtie R_2$		
A	B	C
1	P	null
2	Q	X
3	R	Y
null	S	Z

Alias Operation/rename

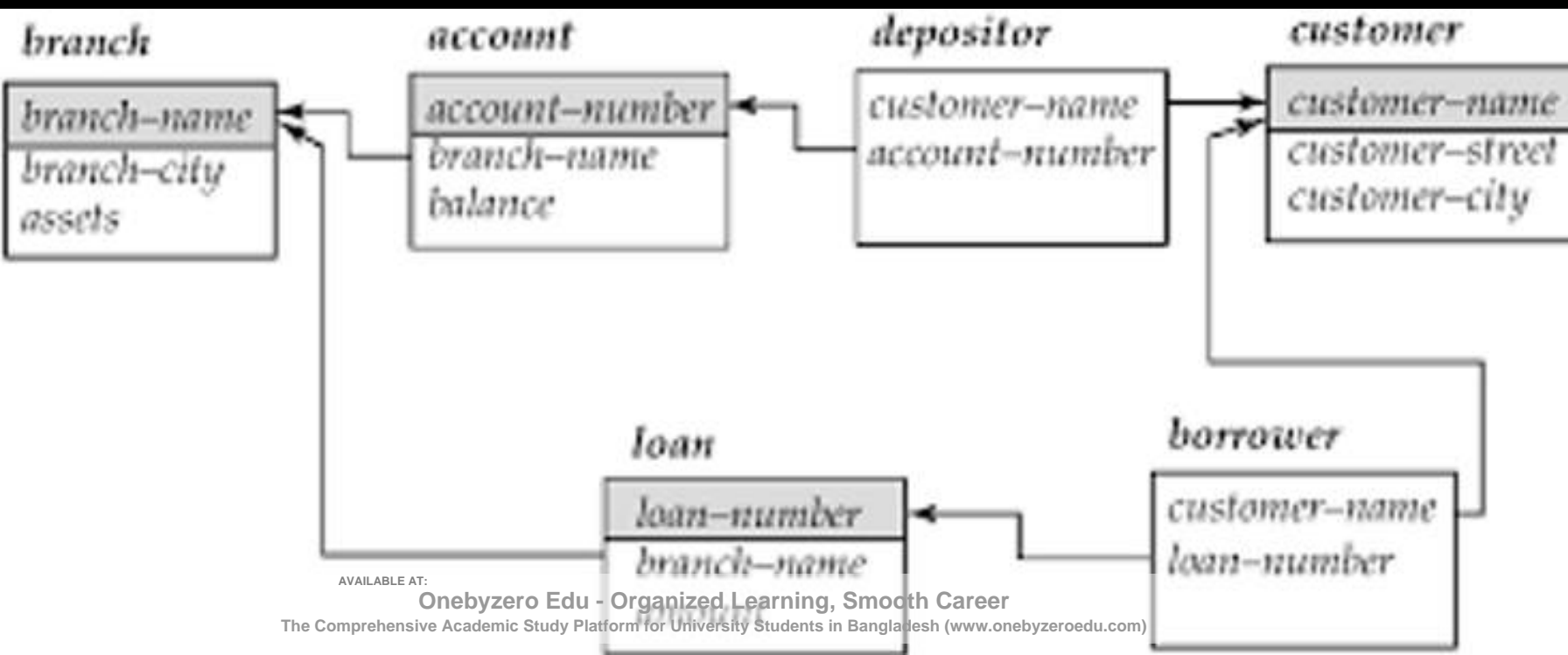
- SQL aliases are used to give a table, or a column in a table, a temporary name. Just create a new copy but do not change anything in the data base. An alias only exists for the duration of the query.
- Aliases are often used to make column names more readable.
- It uses the as clause, taking the form: old-name as new-name. The as clause can appear in both the select and from clauses.

Q Write a SQL query to find the account_no along and balance with 8% interest, as Account, total_balance?



Q Write a SQL query to find the account_no along and balance with 8% interest, as Account, total_balance?

Select account_number, balance*1.06 as total_balance
From account



Q Write a SQL query to find the loan_no with maximum loan amount?

Select balance

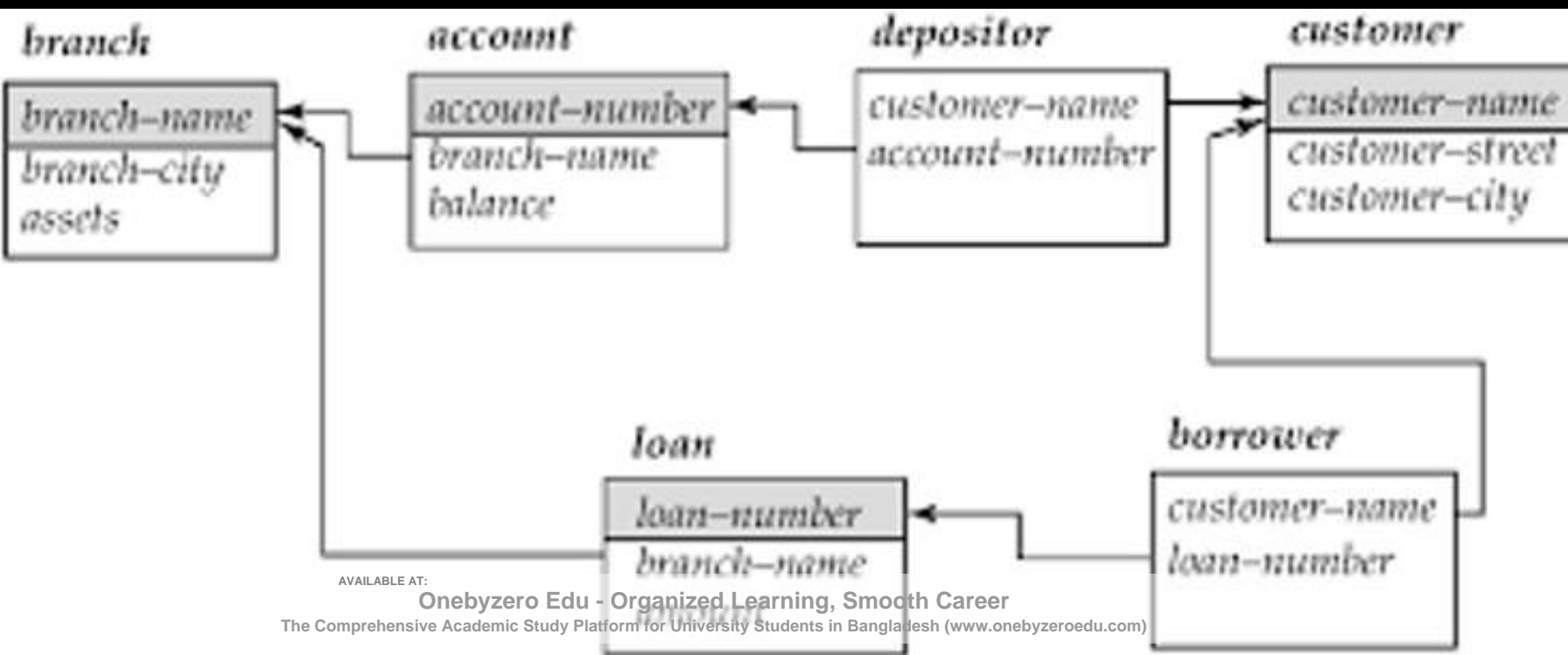
From account

Except

Select A.balance

From account as A, account as B

Where A.balance < B.balance



Aggregate Functions

- *Aggregate functions* are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:
 - Average: **avg**
 - Minimum: **min**
 - Maximum: **max**
 - Total: **sum**
 - Count: **count**
- The input to **sum** and **avg** must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well. Count is the only aggregate function which can work with null, all other aggregate functions simply ignore null.
- We use the aggregate function **count** frequently to count the number of tuples in a relation. The notation for this function in SQL is **count (*)**.

Q find the number of accounts in the bank?

Select count(*)

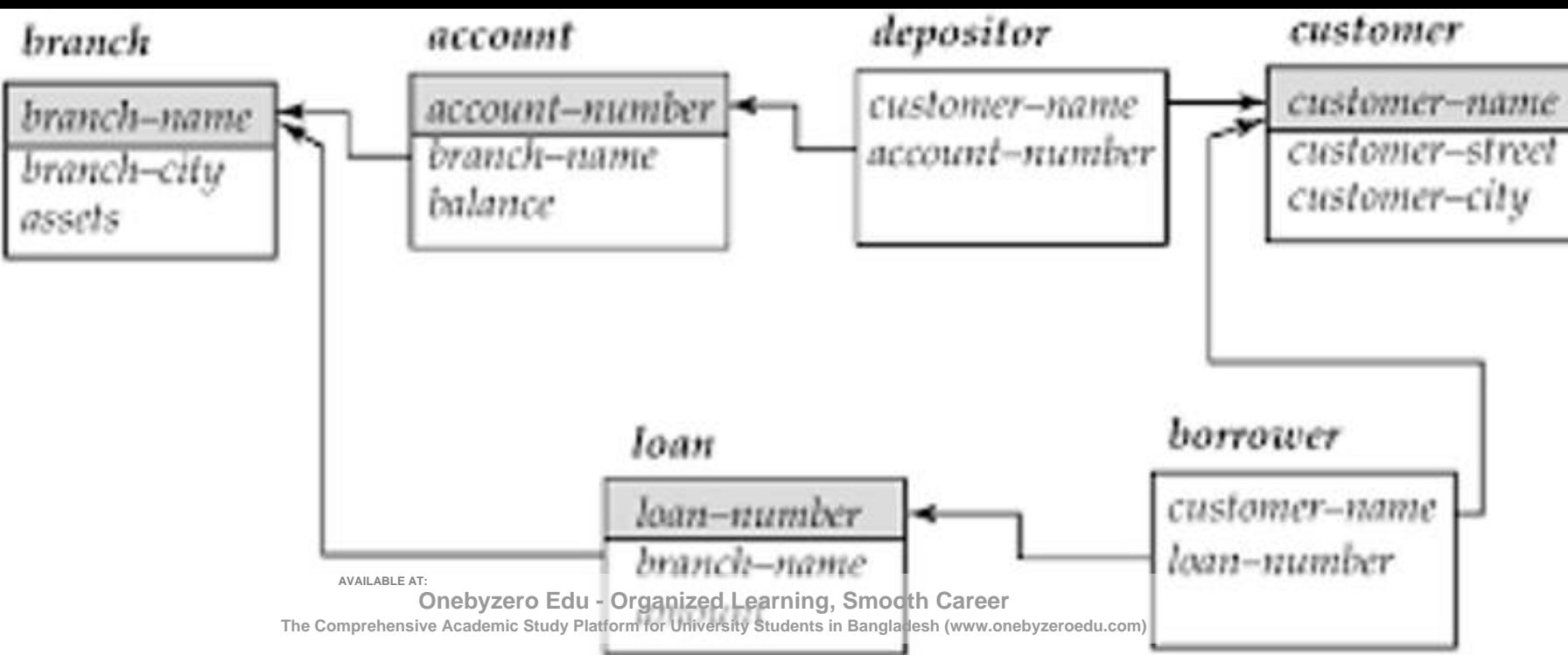
from account

Q find the average balance of every account in the banks from south_delhi branch?

Select avg(balance)

from account

Where branch_name = 'south_delhi'



Q Consider a table along with two query?

Select avg (balance)
from account

Account_no	balance	Branch_name
Abc123	100	N_delhi
Pqr123	500	S_mumbai
Wyz123	null	S_delhi

Select sum(balance)/count(balance)
from account

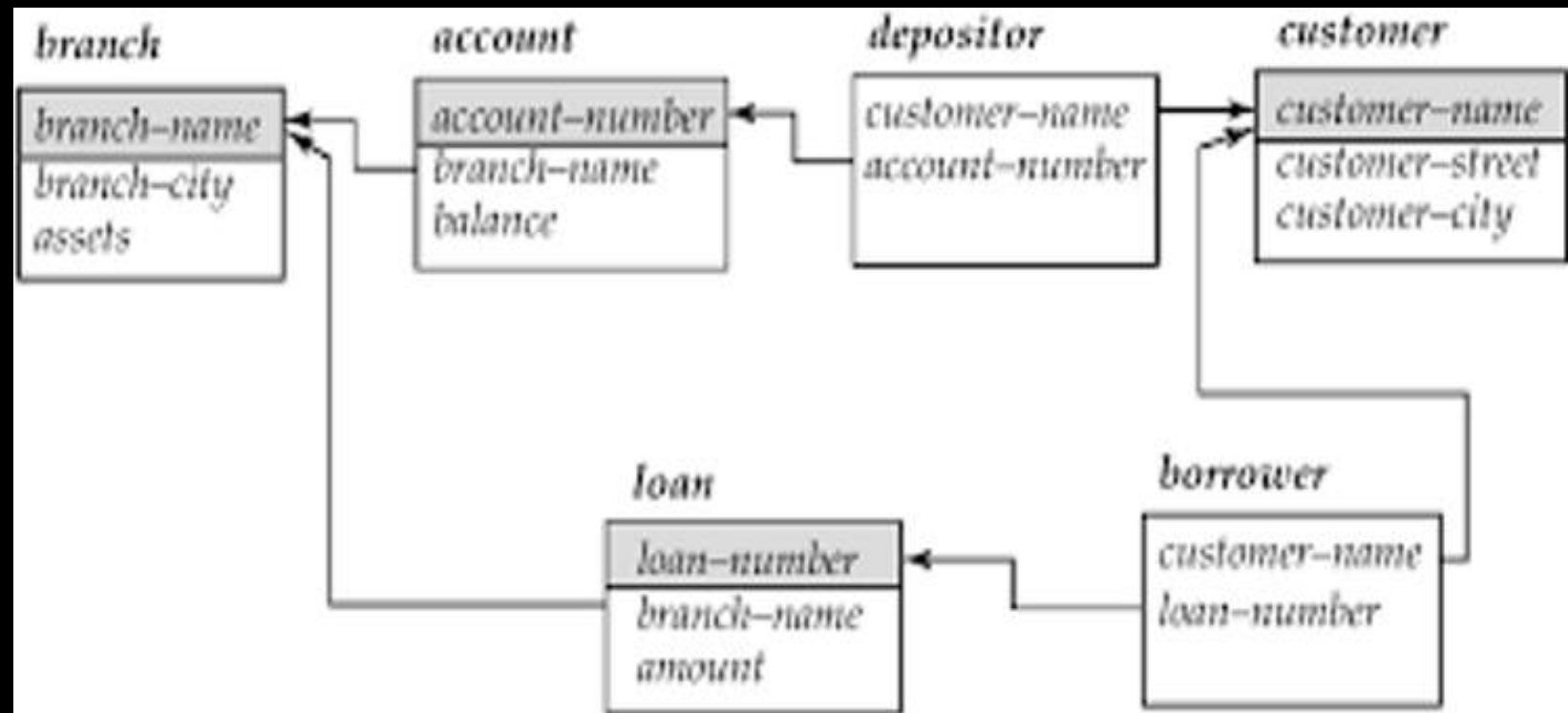
Ordering the Display of Tuples

- SQL offers the user some control over the order in which tuples in a relation are displayed. The **order by** clause causes the tuples in the result of a query to appear in sorted order.

Q find all the **branch_name** which are situated in Delhi in alphabetic order?

Select distinct branch_name
from branch
where branch_city = 'Delhi'
Order by branch_name aesc;

Select distinct branch_name
from branch
where branch_city = 'Delhi'
Order by branch_name desc;



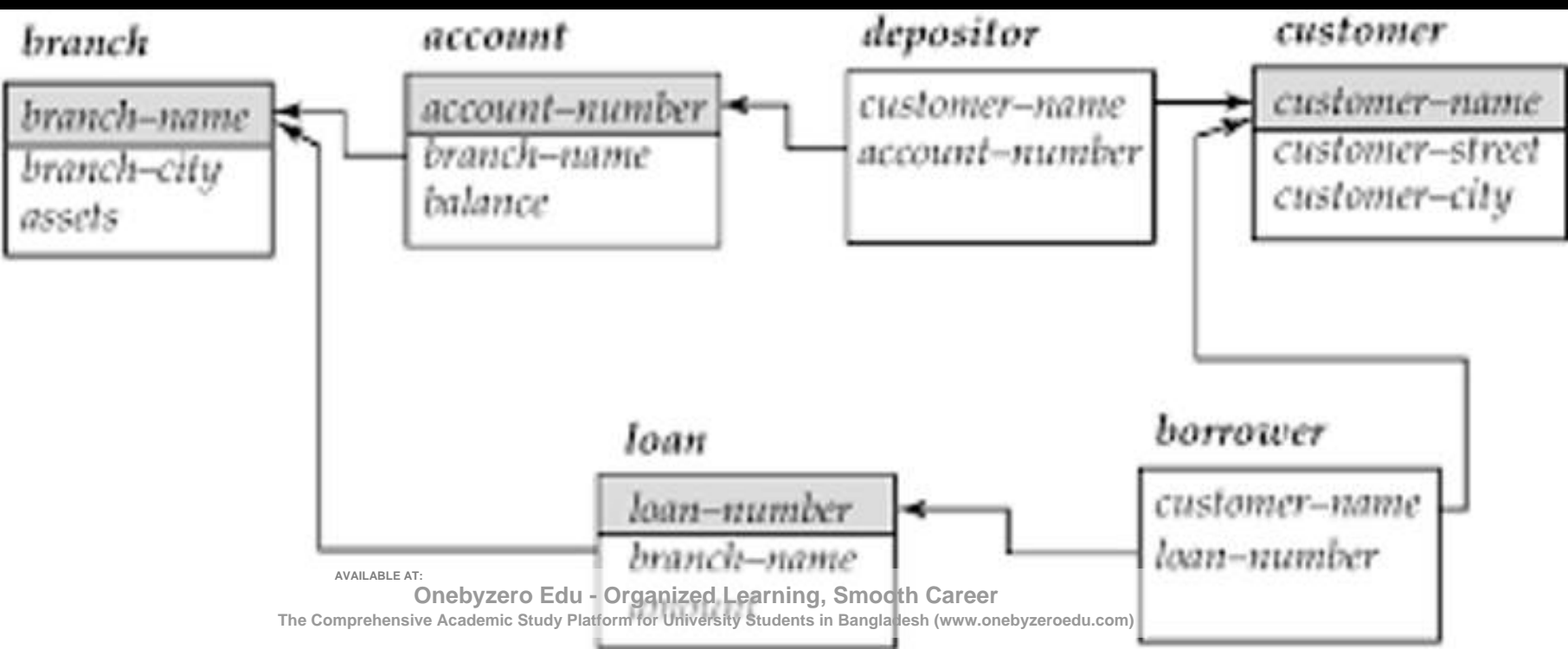
String Operations

- SQL specifies strings by enclosing them in single quotes, for example, 'Computer'. The SQL standard specifies that the equality operation on strings is case sensitive; as a result the expression 'Computer' = 'computer' evaluates to false.
- However, some database systems, such as MySQL and SQL Server, do not distinguish uppercase from lowercase when matching strings; as a result, would evaluate to true on these databases. This default behavior can, however, be changed, either at the database level or at the level of specific attributes.
- SQL also permits a variety of functions on character strings, such as concatenating, extracting substrings, finding the length of strings, converting strings to uppercase and lowercase, removing spaces at the end of the string and so on. There are variations on the exact set of string functions supported by different database systems.

- Pattern matching can be performed on strings, using the operator **like**. We describe patterns by using two special characters:
 - Percent (%): The % character matches any substring.
 - Underscore (_): The _ character matches any character.
- '%Comp%' matches any string containing "Comp" as a substring, for example, 'Intro to Computer Science', and 'Computational Biology'.
 - '___' matches any string of exactly three characters.
 - '___%' matches any string of at least three characters.

Q find all the branch name who have exactly 5 character in their name ?

Q find all the customer name who have 'kumar' in their name ?

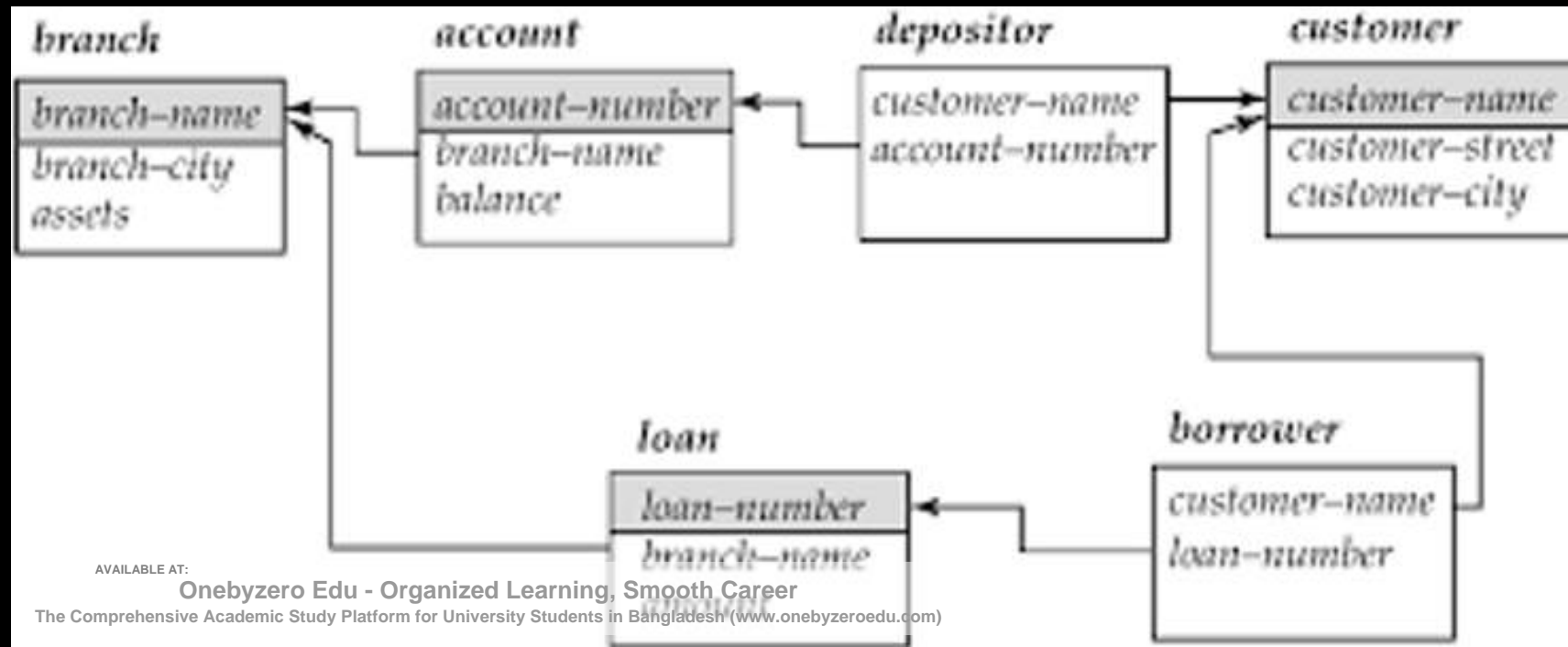


Q find all the branch name who have exactly 5 character in their name ?

Select branch_name
from branch
where branch_name like '_____'

Q find all the customer name who have 'kumar' in their name ?

Select customer_name
from customer
where customer_name like '%kumar%'



- We define the escape character for a **like** comparison using the **escape** keyword. To illustrate, consider the following patterns, which use a backslash (\) as the escape character:
- **like** 'ab\%cd%' **escape** '\' matches all strings beginning with “ab%cd”.
- **like** 'ab\\cd%' **escape** '\' matches all strings beginning with “ab\cd”.

Group by clause

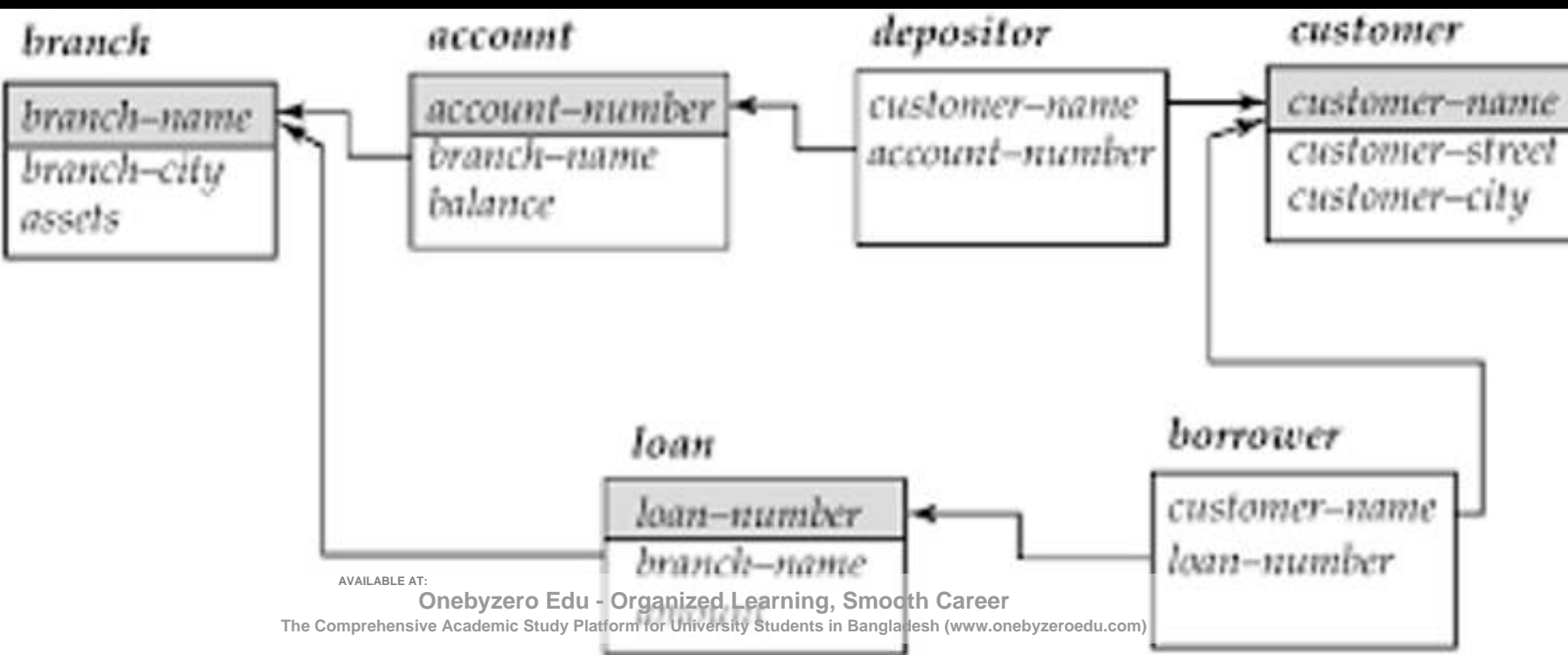
1. There are circumstances where we would like to work on a set of tuples in a relation rather than working on the whole table as one unit.
2. The attribute or attributes given in the **group by** clause are used to form groups. Tuples with the same value on all attributes in the **group by** clause are placed in one group.

Q find the average account balance of each branch?

Select branch_name, avg(balance)

from account

Group by branch_name



Q find the branch name of Gwalior city with average balance more than 1500?

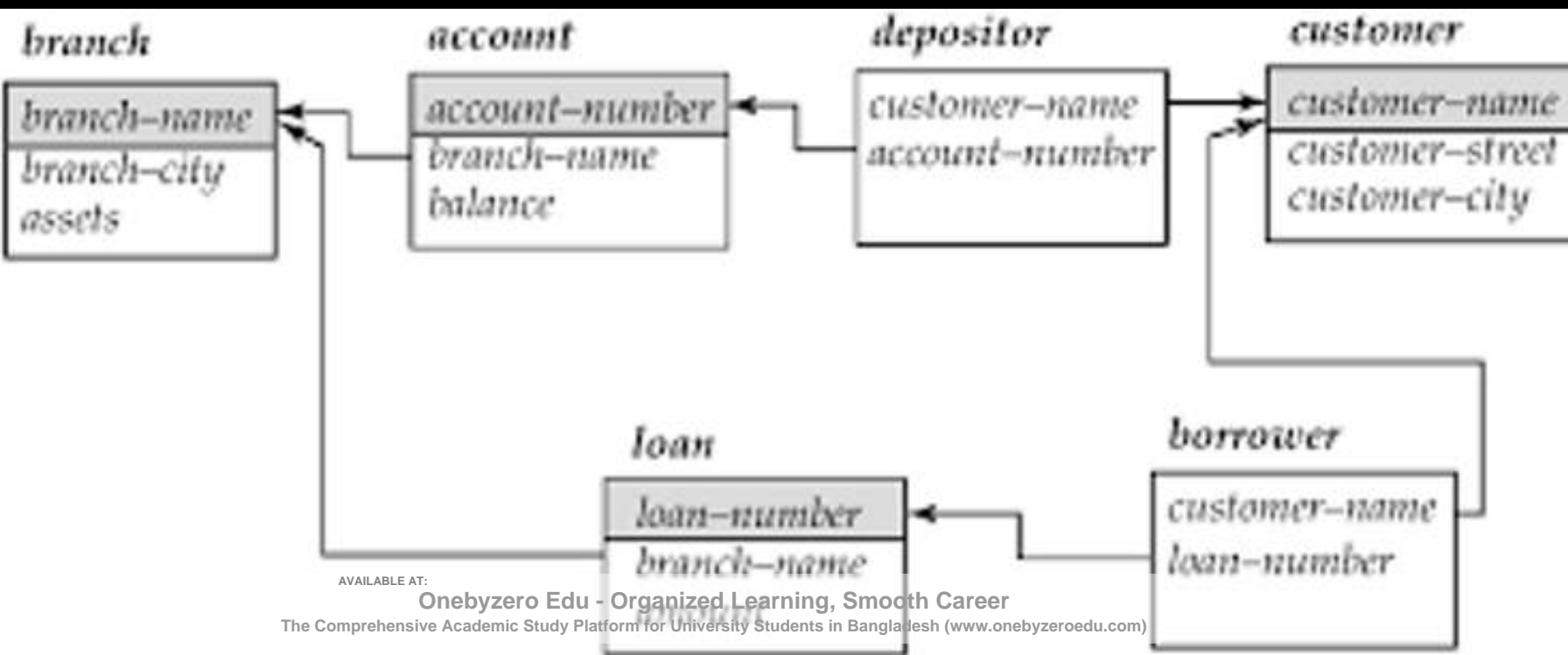
Select branch.branch_name, avg(balance)

from branch, account

Where branch.branch_name = account.branch_name and branch_city = 'gwalior'

Group by branch_name

Having avg(balance) > 1500

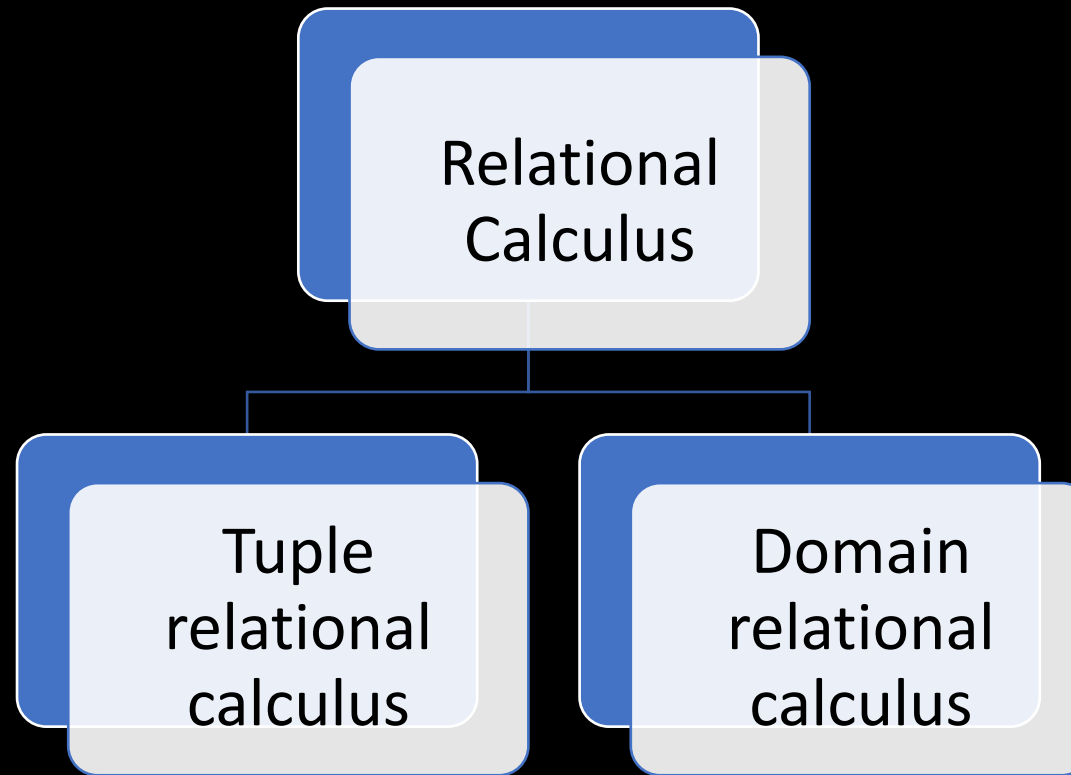


- **Trigger Definition**
 - A trigger is a special procedure that automatically activates in response to modifications on a table or view in a database, aiding in maintaining data integrity.
- **Usage of Triggers**
 - Triggers help maintain database integrity by automatically enforcing conditions or modifying data during database operations. They are crucial for applying business rules, auditing database alterations, and supporting data replication, ensuring compliance before any data insertions, updates, or deletions.

- **Embedded SQL**
 - Embedded SQL is a method where SQL statements are incorporated directly into a procedural programming language, such as C or Java. It allows the programmers to integrate SQL queries within their code, facilitating the interaction between the database and the application. Embedded SQL statements are static and defined at compile time.
- **Dynamic SQL**
 - Dynamic SQL, on the other hand, enables the construction of SQL statements dynamically at runtime. It allows the creation of more flexible and adaptable applications, where SQL statements can be generated and executed based on changing conditions or user inputs. This makes it possible to create more complex and adaptable database operations, though it might be more susceptible to SQL injection attacks if not handled carefully.

Relational Calculus

- Relational calculus is non-procedural query language, where we have to define what to get and not how to get it



Tuple Relational Calculus

- TRC is based on specifying a number of tuple variables. Each tuple variable usually range over a particular database relation, meaning that the variable may takes as its value from any individual tuple of a relation.
- A simple tuple relational calculus query is of the form.
 - $\{t \mid \text{Condition}(t)\}$
- Where t is a tuple variable and $\text{condition}(t)$ is a conditional expression involving. The result of such a query is the set of all tuple t that satisfy condition (t) .

Student(Roll No, Name, Branch)

Q Find the details of all computer science students?

SQL: select * from student where branch = CSE

RA: $\{\sigma_{\text{branch} = \text{CSE}}(\text{Student})\}$

TRC: $\{t \mid \text{Student}(t) \wedge t.\text{branch} = \text{CSE}\}$

DRC:

Student(Roll No, Name, Branch)

Q Find the Roll No of all computer science students?

SQL: select Roll No from student where branch = CSE

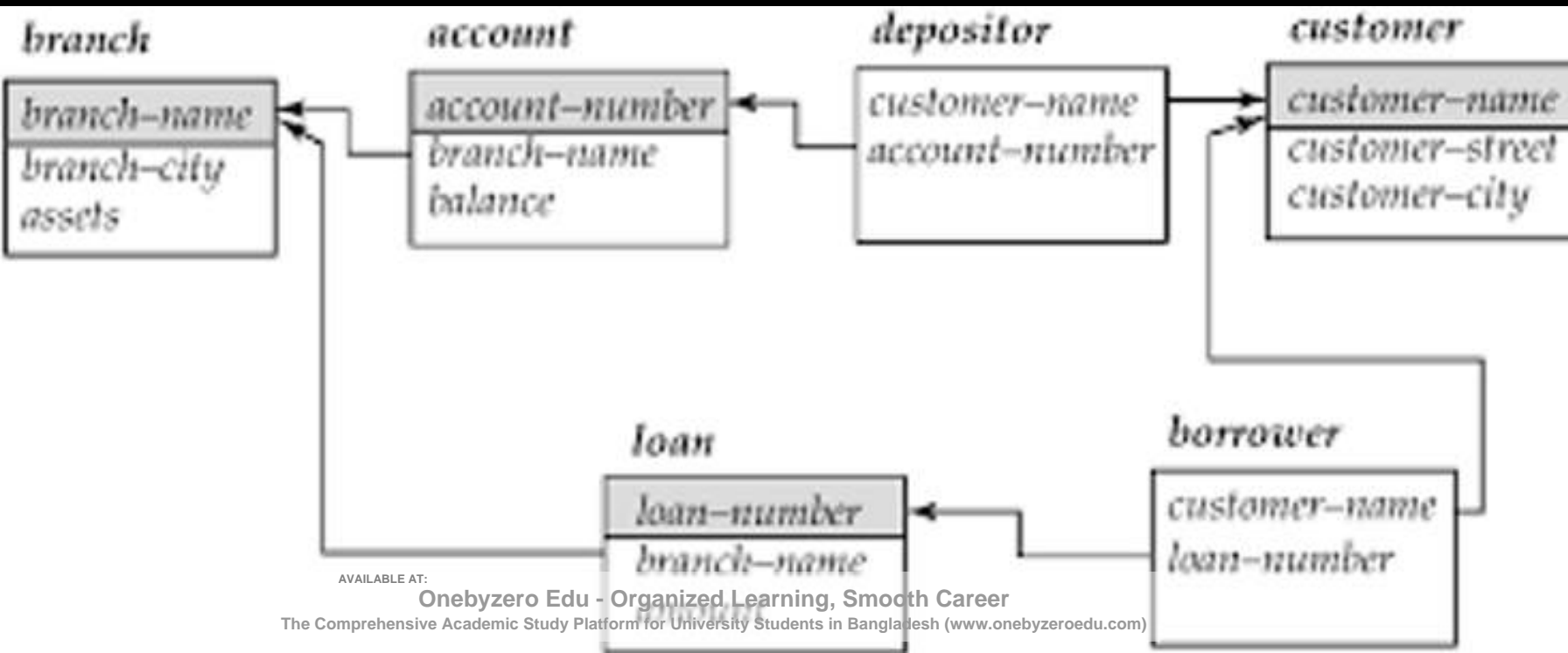
RA: $\{\Pi_{\text{roll no}} (\sigma_{\text{branch} = \text{CSE}} (\text{Student}))\}$

TRC: $\{t. \text{Roll No} \mid \text{Student}(t) \wedge t. \text{branch} = \text{CSE}\}$

DRC:

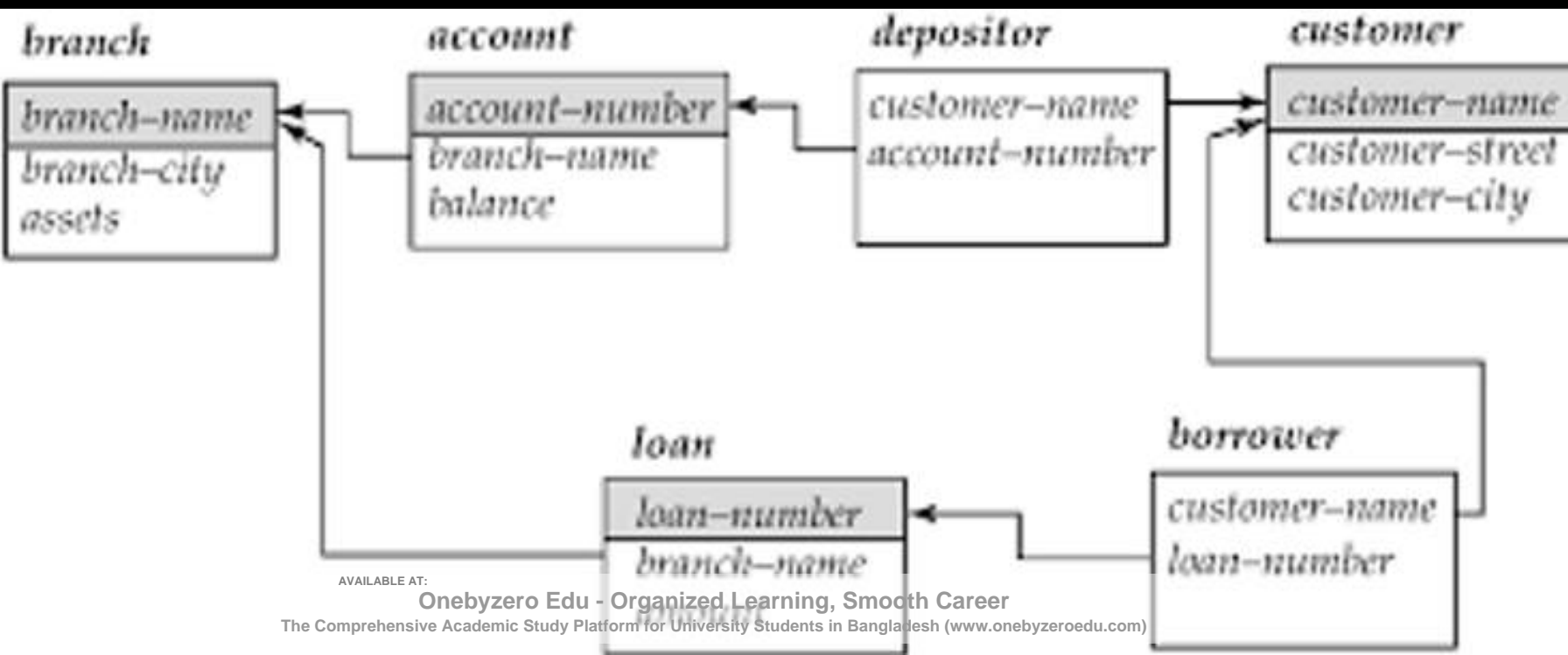
Q Find all the details of loan for amount over 1200?

$\{t \mid t \in \text{loan} \wedge t[\text{amount}] > 1200\}$



Q Find the loan number for each loan of amount over 1200?

$\{t \mid \exists s \in \text{loan } (t[\text{loan number}] = s[\text{loan number}]) \wedge s[\text{amount}] > 1200\}$

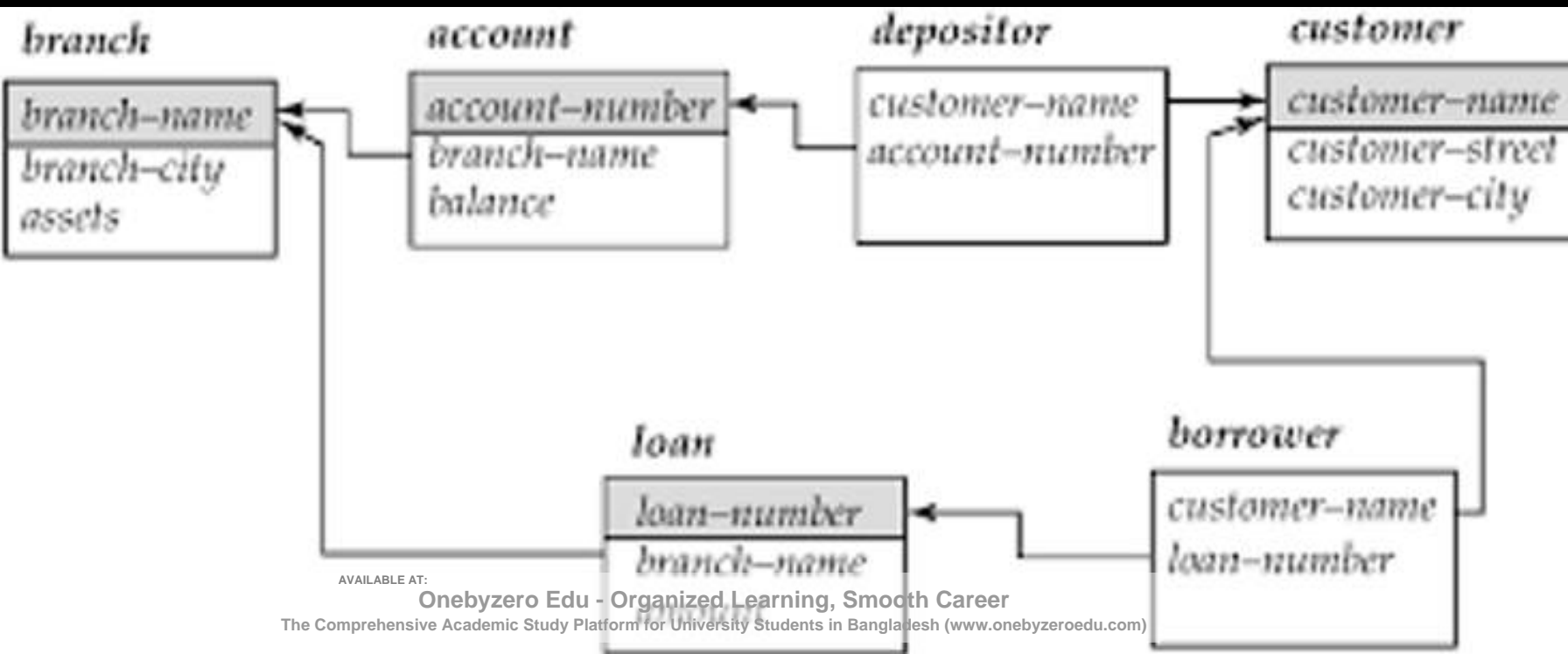


Q Find the name of all the customers who have a loan from Noida branch?

$\{t \mid \exists s \in \text{borrower} (t[\text{customer name}] = s[\text{customer name}])$

$\wedge \exists u \in \text{loan} (u[\text{customer name}] = s[\text{customer name}])$

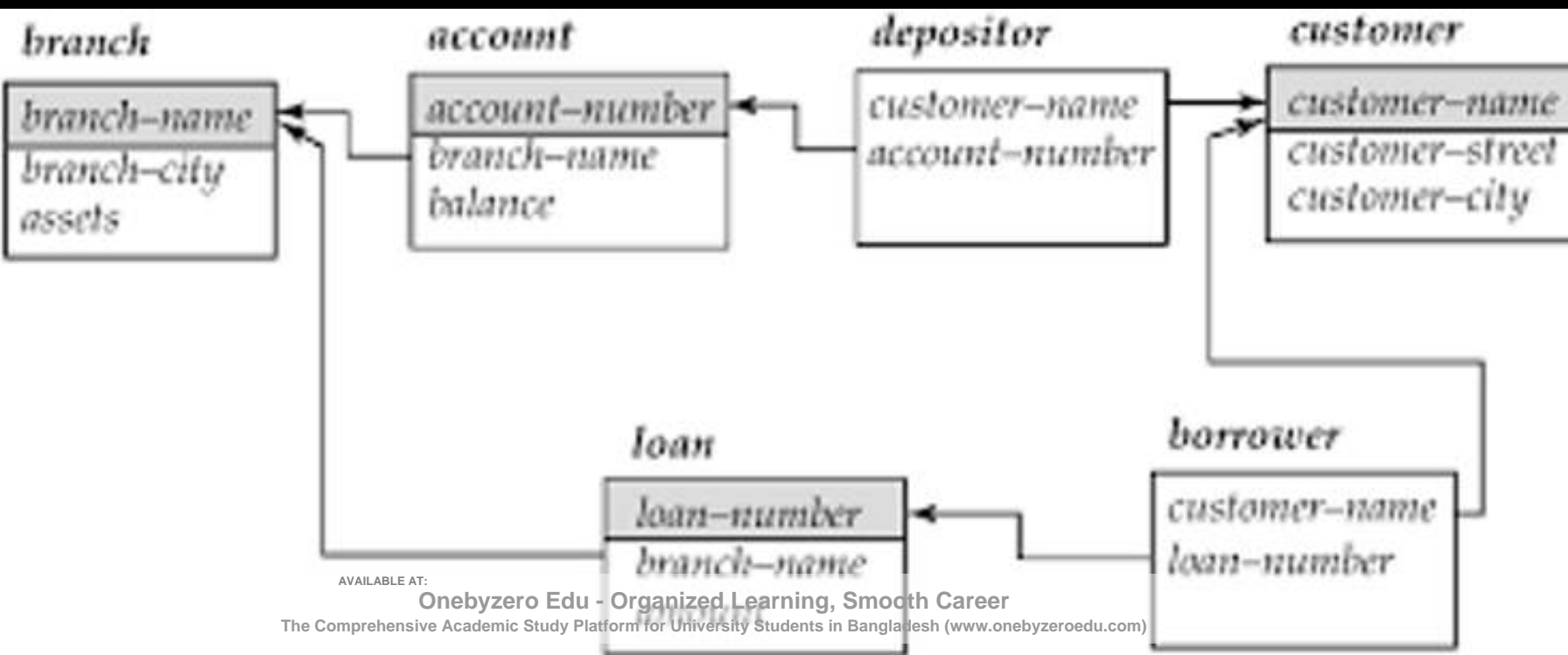
$\wedge u[\text{branch name}] = \text{'Noida'} \}$



Q Find the name of all the customers who have a loan or account or both at the bank?

$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$

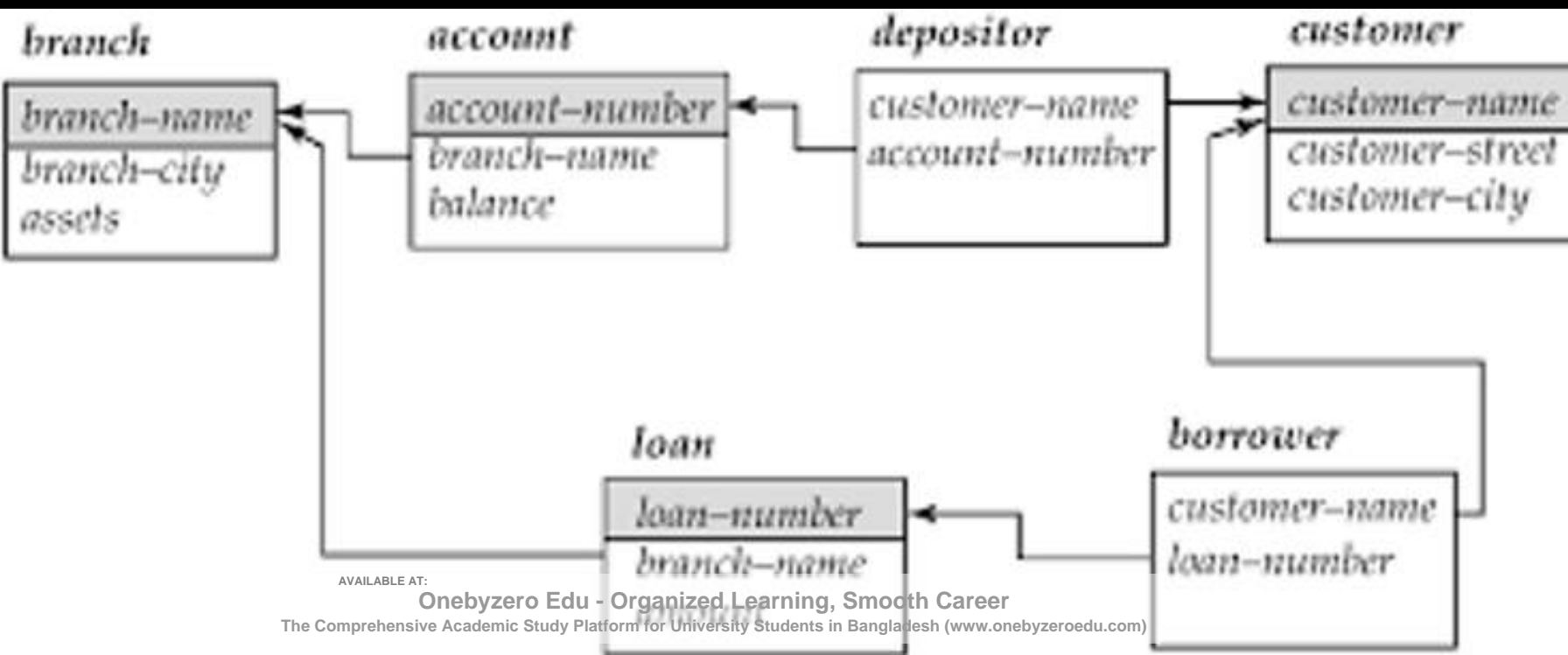
$\vee \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}]) \}$



Q Find the name of all the customers who have a loan and account both at the bank?

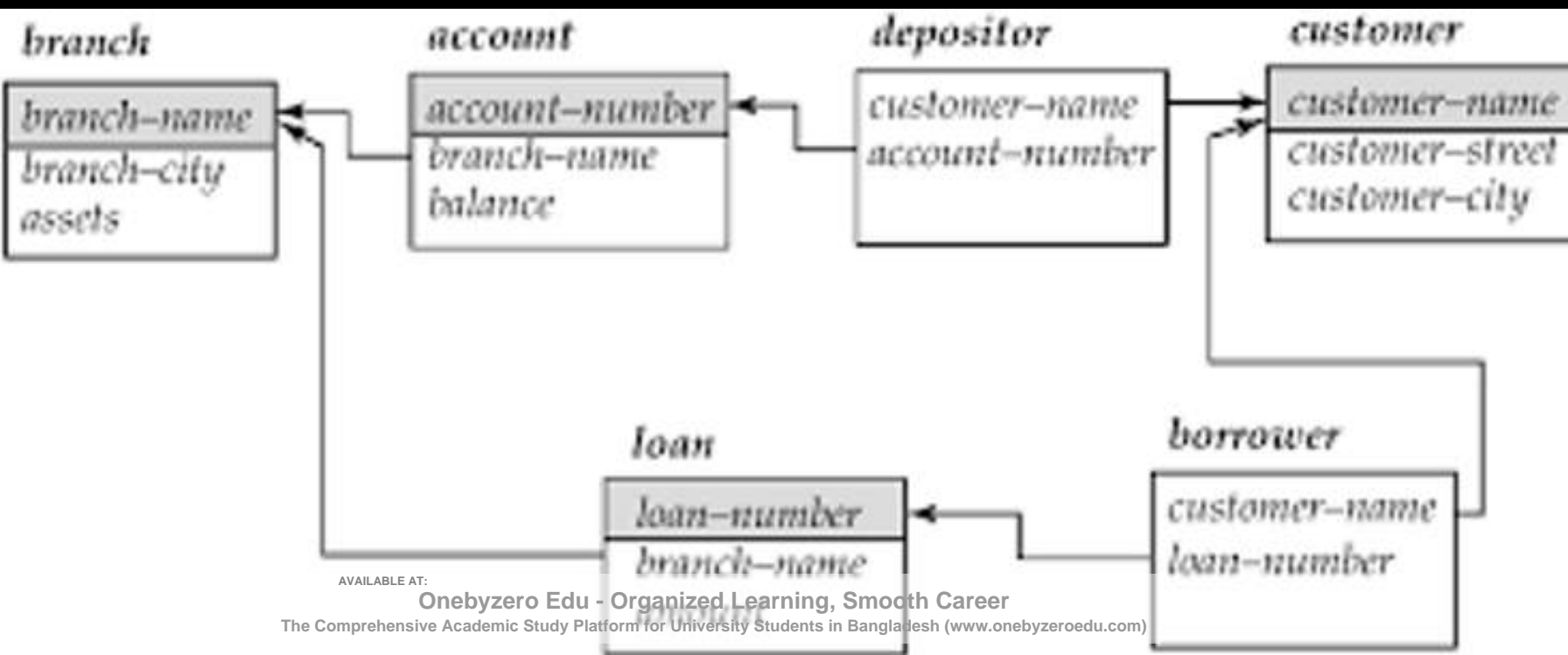
$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$

$\wedge \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}]) \}$



Q Find the name of all the customers who have a loan from the bank and do not have a account?

$\{t \mid \exists u \in \text{depositor borrower } (t[\text{customer name}] = u[\text{customer name}])$
 $\wedge \neg \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}]) \}$



Domain Relational Calculus

- Domain calculus differs from the tuple calculus in the type of variables used in formulas: rather than having variables range over tuples.
- The variable range over single values from domains of attributes. To form a relation of degree n for a query result, we must have n of these domain variables. One for each attribute.
- An expression of the domain calculus is of the form
- $(x_1, x_2, \dots, x_n \mid \text{COND}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}))$
- Where x_1, x_2, \dots, x_n are domain variables that range over domains and COND is a condition of the domain relational calculus.

Student(Roll No, Name, Branch)

Q Find the details of all computer science students?

SQL: select * from student where branch = CSE

RA: $\{\sigma_{\text{branch} = \text{CSE}}(\text{Student})\}$

TRC: $\{t \mid \text{Student}(t) \cap t.\text{branch} = \text{CSE}\}$

DRC: $\{(\text{Roll No, Name, Branch}) \mid \text{Student}(\text{Roll no, Name, Branch}) \cap \text{branch} = \text{CSE}\}$

Student(Roll No, Name, Branch)

Q Find the Roll No of all computer science students?

SQL: select Roll No from student where branch = CSE

RA: $\{\Pi_{\text{Roll No}} (\sigma_{\text{branch} = \text{CSE}} (\text{Student}))\}$

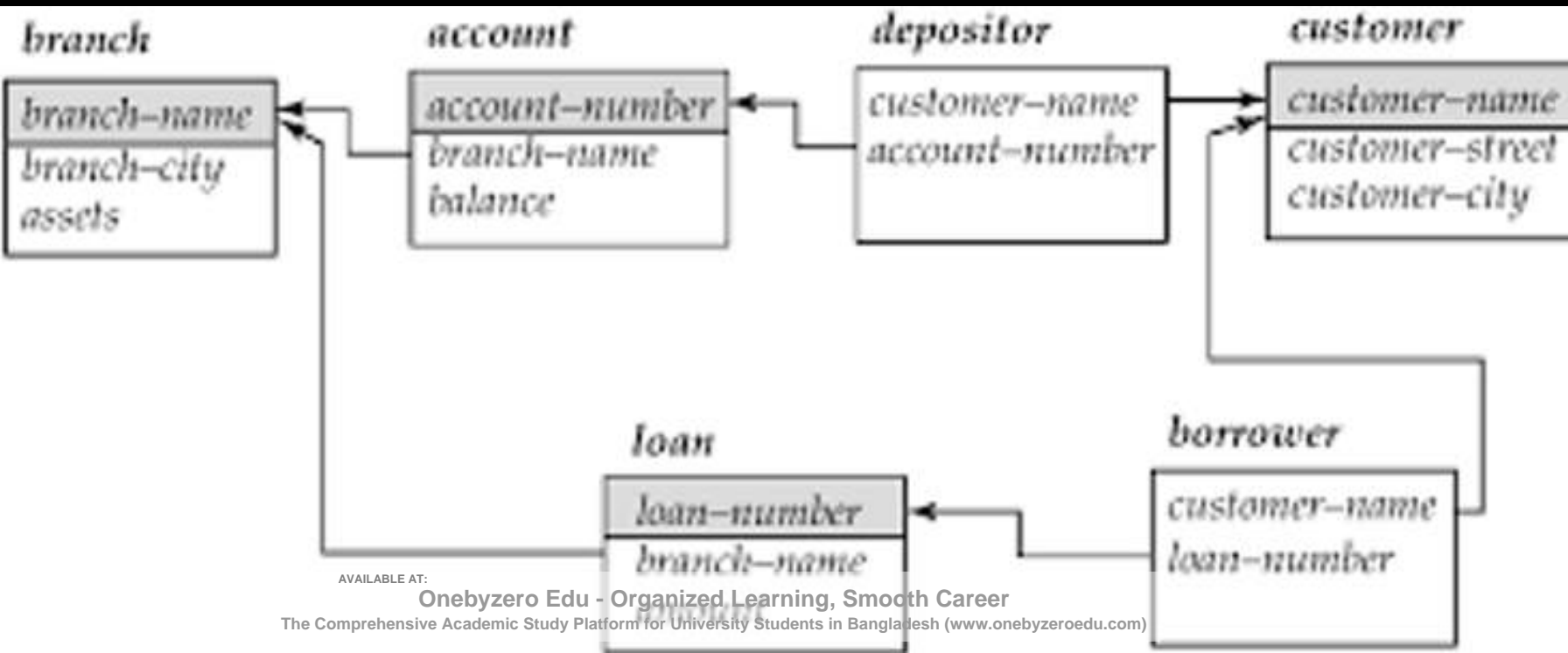
TRC: $\{t. \text{Roll No} \mid \text{Student}(t) \wedge t. \text{branch} = \text{CSE}\}$

DRC: $\{(\text{Roll No}) \mid \text{Student}(\text{Roll no, Name, Branch}) \wedge \text{branch} = \text{CSE}\}$

Example: Find all details of instructors whose salary is greater than \$80,000

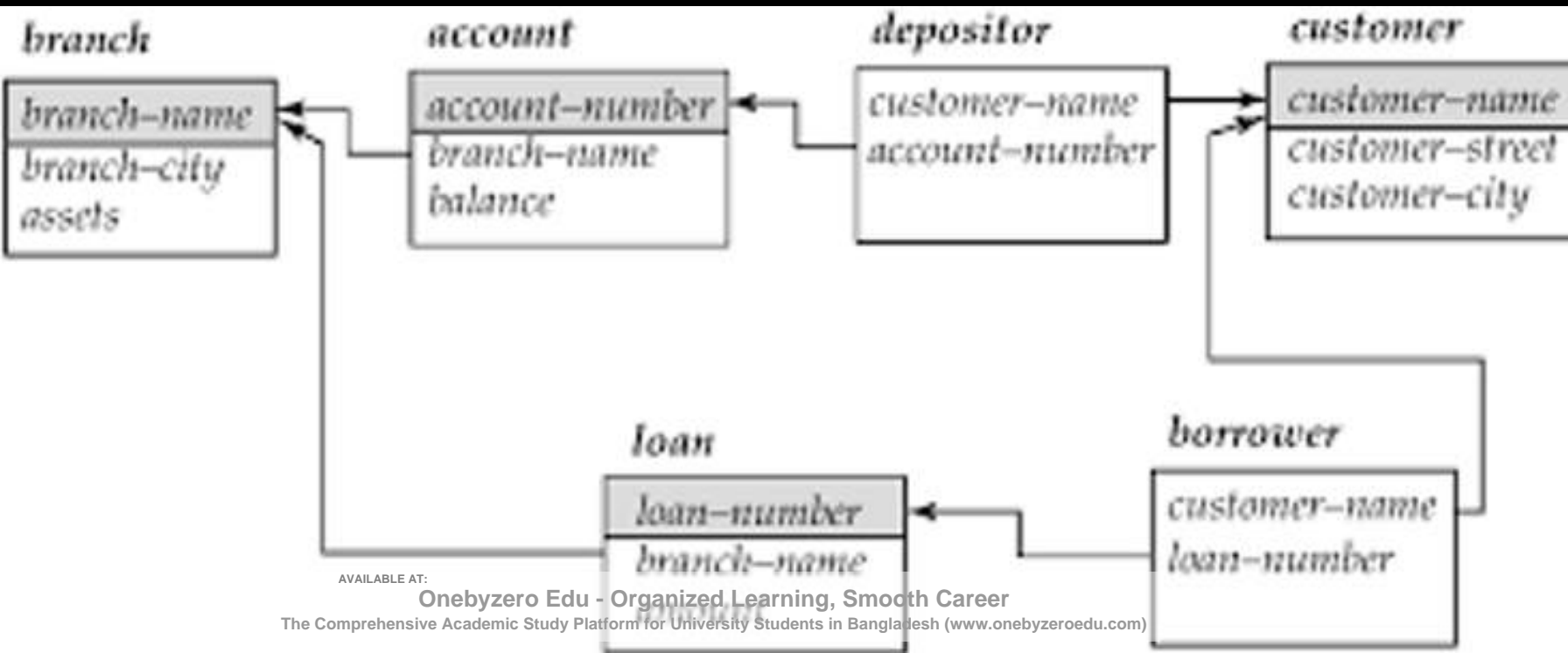
Q Find the branch name, loan number and amount for loan of amount over 1200?

$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$



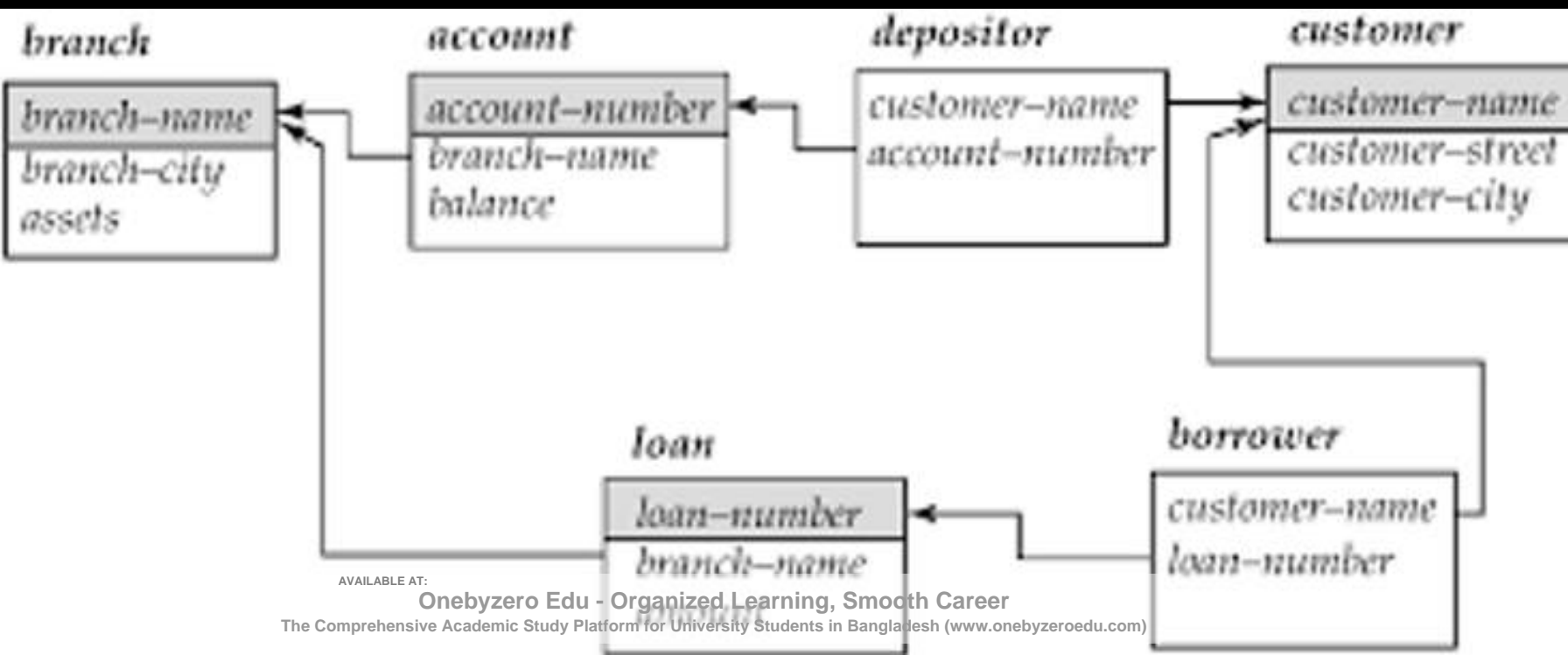
Q Find the loan number for each loan of amount over 1200?

$\{ \langle l \rangle \mid \exists_{b,a} \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$



Q Find the name of all the customers who have a loan from Noida branch with loan amount?

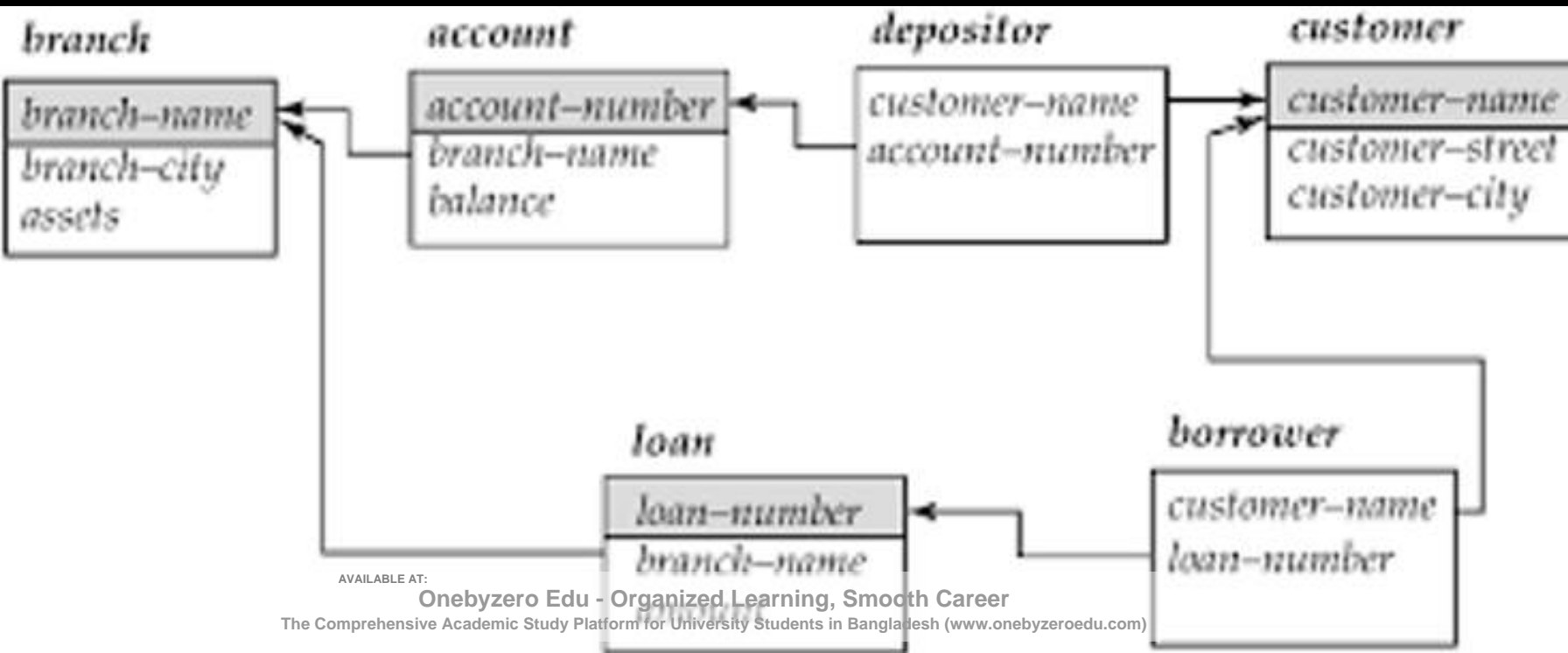
$\{ \langle c, a \rangle \mid \exists_l (\langle c, l \rangle \in \text{borrower} \wedge \exists_b (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{'Noida'})) \}$



Q Find the name of all the customers who have a loan or account or both at the bank?

$\{ \langle c \rangle \mid \exists_l (\langle c, l \rangle \in \text{borrower} \wedge \exists_{b,a} (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{'Noida'}))$

$\vee \exists_a (\langle c, a \rangle \in \text{depositor} \wedge \exists_{b,a} (\langle a, bn, bal \rangle \in \text{account} \wedge bn = \text{'Noida'})) \}$



TRANSACTION

- Why we study transaction?
 - According to general computation principle (operating system) we may have partially executed program, as the level of atomicity is instruction i.e. either an instruction is executed completely or not
 - But in DBMS view, user perform a logical work(operation) which is always atomic in nature i.e. either operation is execute or not executed, there is no concept like partial execution. For example, Transaction T_1 which transfer 100 units from account A to B.
 - In this transaction if a failure occurs after Read(B) then the final statue of the system will be inconsistent as 100 units are debited from account A but not credited in account B, this will generate inconsistency. Here for 'consistency' before $(A + B) ==$ after $(A + B)$ ".

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)

What is transaction

- To remove this partial execution problem, we increase the level of atomicity and bundle all the instructions of a logical operation into a unit called transaction.
- So formally 'A transaction is a Set of logically related instructions to perform a logical unit of work'.

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)

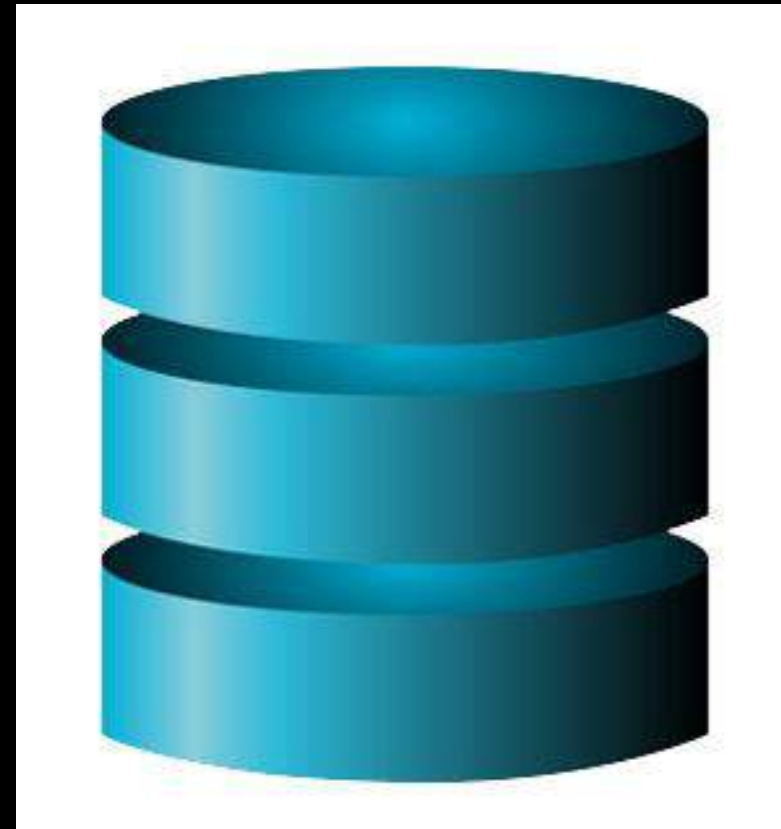


- As here we are only concerned with DBMS so we will only two basic operation on database.
- **READ (X)** - Accessing the database item x from disk (where database stored data) to memory variable also name as X.
- **WRITE (X)** - Writing the data item from memory variable X to disk.

Desirable properties of transaction

- Now as the smallest unit which have atomicity in DBMS view is transaction, so if want that our data should be consistent then instead of concentrating on data base, we must concentrate on the transaction for our data to be consistent.

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)



- Transactions should possess several properties, often called the **ACID** properties; to provide integrity and consistency of the data in the database. The following are the ACID properties:

A = Atomicity

C = Consistency

I = Isolation

D = Durability

- **Atomicity** - A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)

- **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.

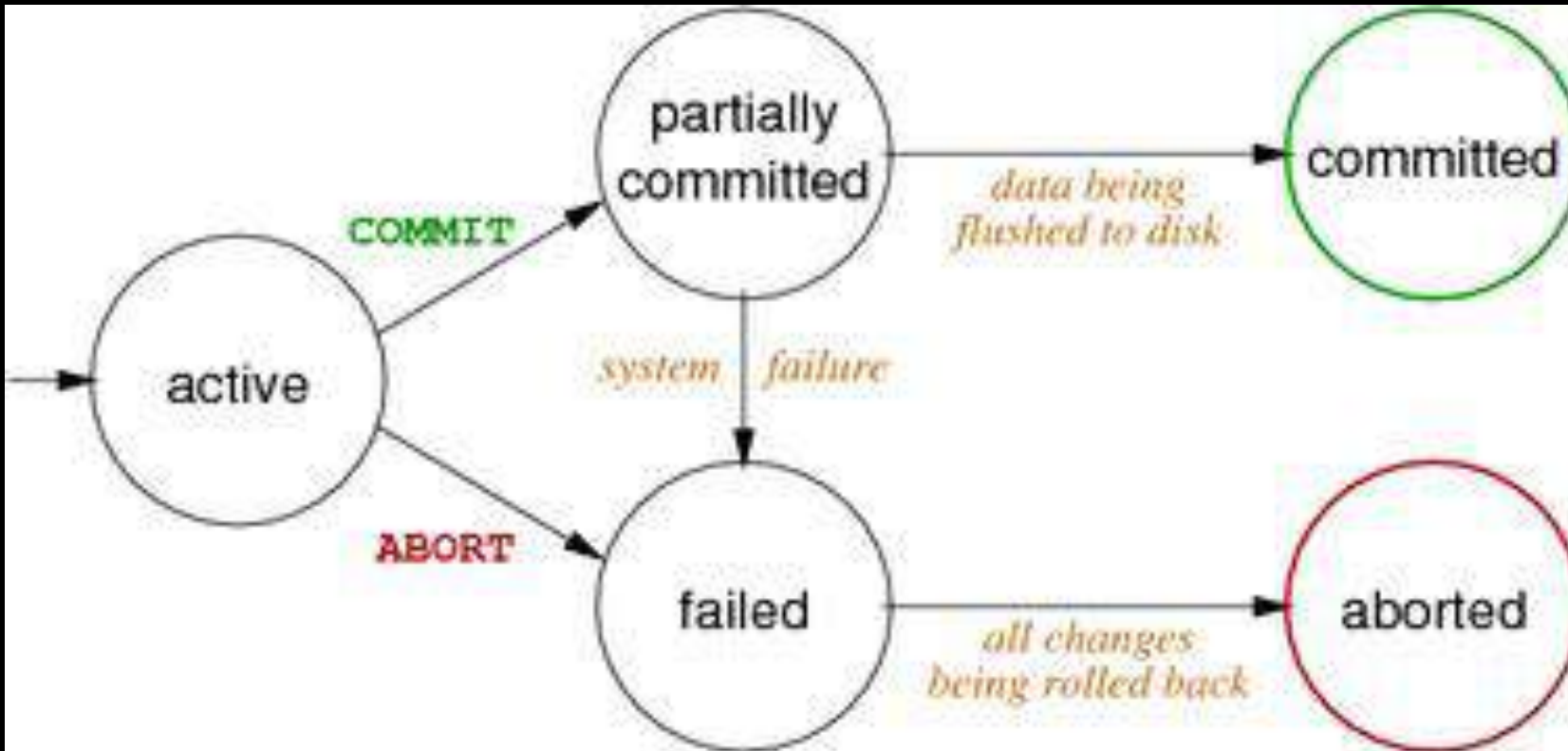


- **Isolation** - A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently.
- That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

- **Durability** - The changes applied to the database by a committed transaction must persist in the database.

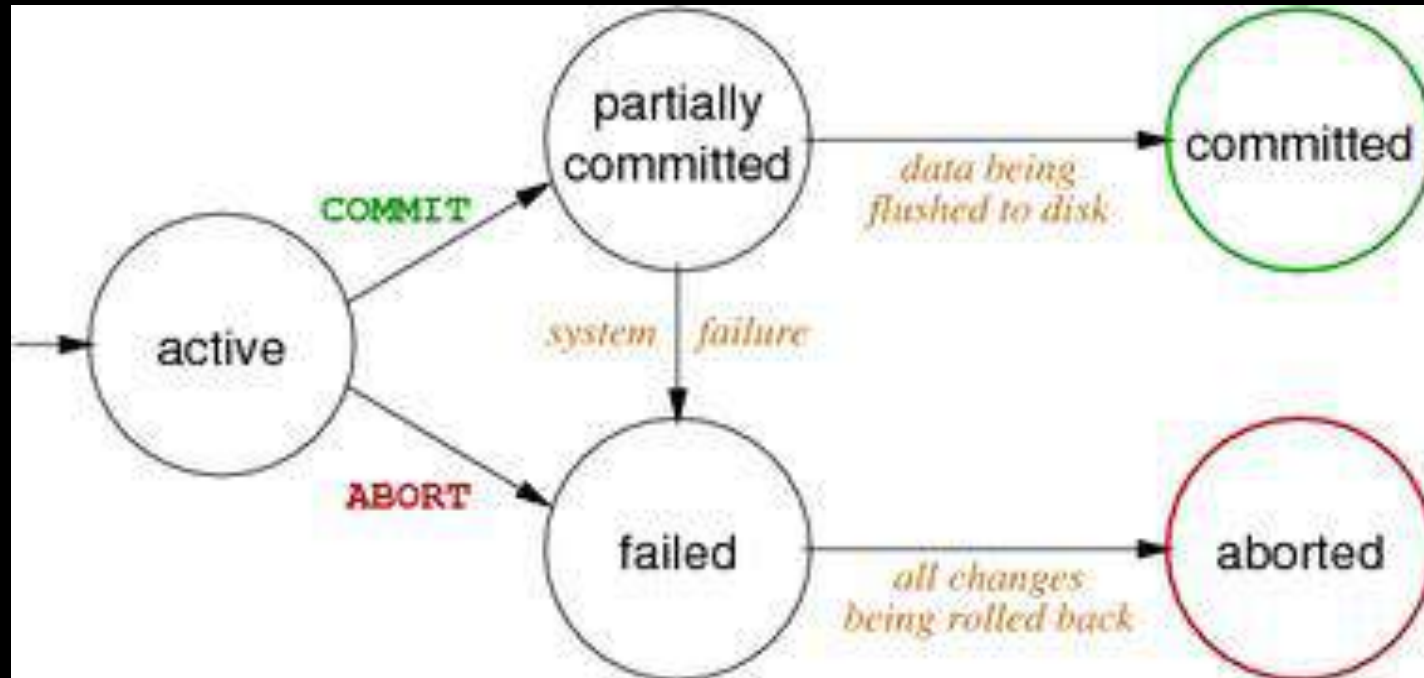
Transaction states

- **ACTIVE** - It is the initial state. Transaction remains in this state while it is executing operations.



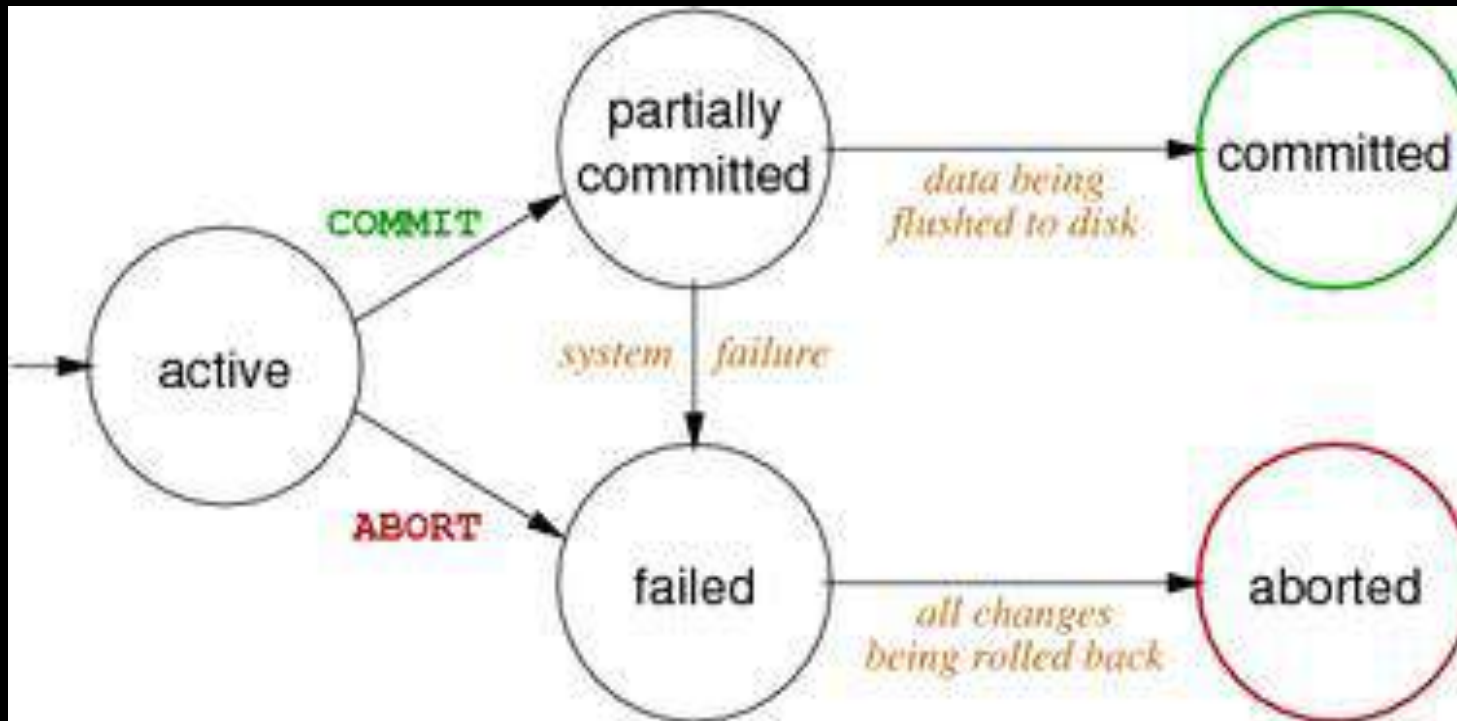
Transaction states

- **PARTIALLY COMMITTED** - After the final statement of a transaction has been executed, the state of transaction is partially committed as it is still possible that it may have to be aborted (due to any failure) since the actual output may still be temporarily residing in main memory and not to disk.



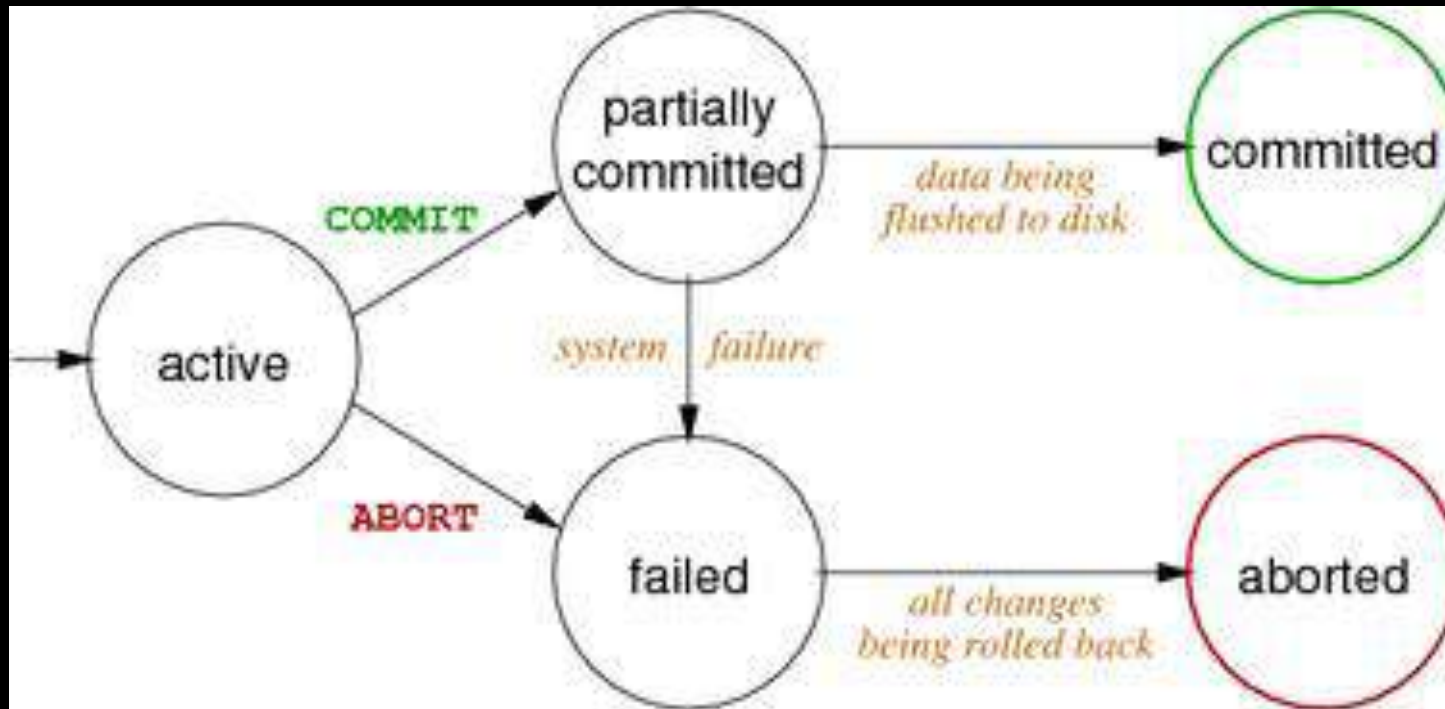
Transaction states

- **FAILED** - After the discovery that the transaction can no longer proceed (because of hardware /logical errors). Such a transaction must be rolled back.



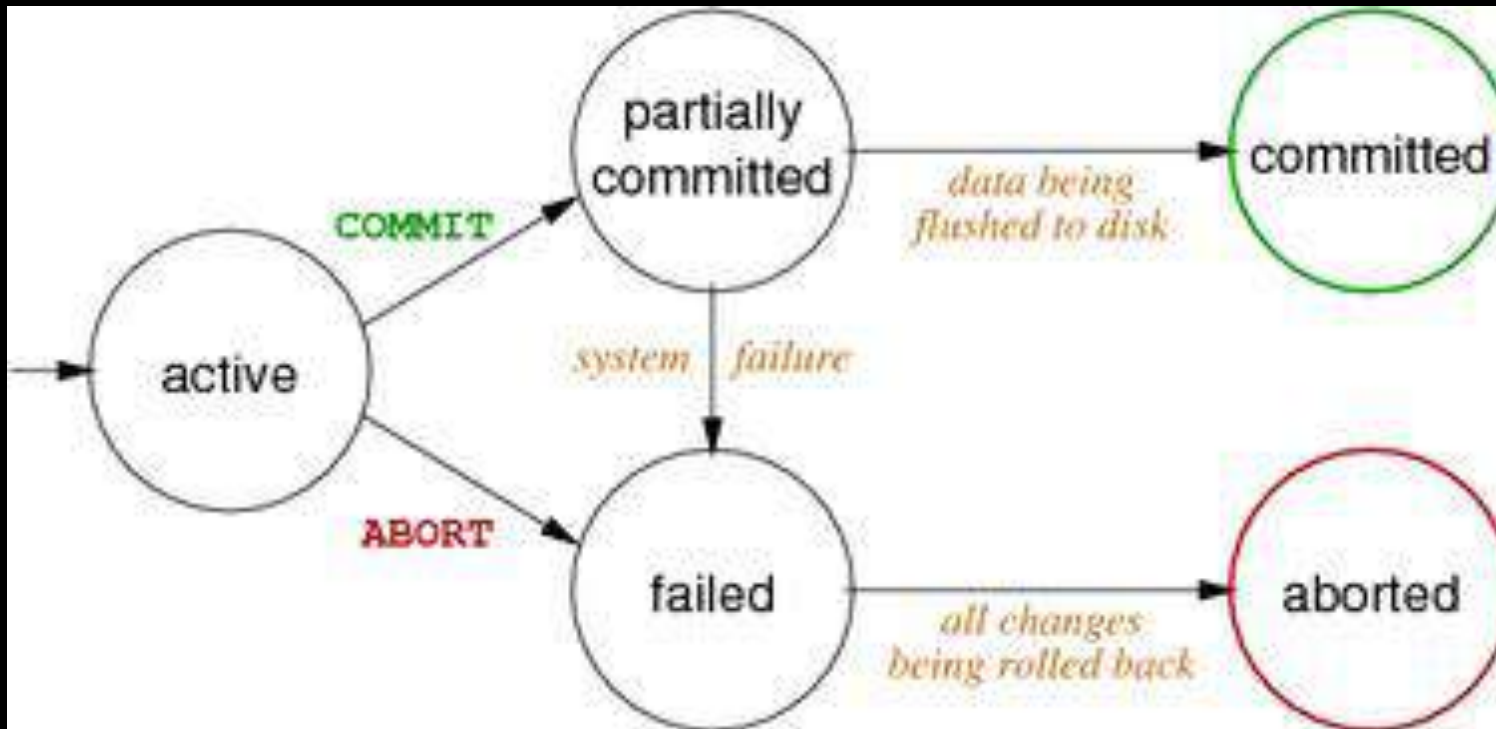
Transaction states

- **ABORTED** - A transaction is said to be in aborted state when the when the transaction has been rolled back and the database has been restored to its state prior to the start of execution.



Transaction states

- **COMMITTED** - A transaction enters committed state after successful completion of a transaction and final updation in the database.



Why we need concurrent execution

- Concurrent execution is necessary because-
 - It leads to good database performance , less waiting time.
 - Overlapping I/O activity with CPU increases throughput and response time.



PROBLEMS DUE TO CONCURRENT EXECUTION OF TRANSACTION

- But interleaving of instructions between transactions may also lead to many problems that can lead to inconsistent database.
- Sometimes it is possible that even though individual transaction are satisfying the acid properties even though the final statues of the system will be inconsistent.

Solution is Schedule

- When two or more transaction executed together or one after another then they can be bundled up into a higher unit of execution called schedule.

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

T_2	T_3
read(B)	
	read(B)
	write(B)
read(A)	
	read(A)
	write(A)

- **Serial schedule** - A serial schedule consists of sequence of instruction belonging to different transactions, where instructions belonging to one single transaction appear together. Before complete execution of one transaction another transaction cannot be started. Every serial schedule lead database into consistent state.

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

- **Non-serial schedule** - A schedule in which sequence of instructions of a transaction appear in the same order as they appear in individual transaction but the instructions may be interleaved with the instructions of different transactions i.e. concurrent execution of transactions takes place.

T_2	T_3
read(B)	read(B)
	write(B)
read(A)	read(A)
	write(A)

- **Conclusion of schedules**
 - We do not have any method to proof that a schedule is consistent, but from the above discussion we understand that a serial schedule will always be consistent.
 - So if somehow we proof that a non-serial schedule will also have same effects as of a serial schedule than we can proof that, this particular non-serial schedule will also be consistent.

On the basis of
SERIALIZABILITY

```
graph TD; A[On the basis of SERIALIZABILITY] --- B[Conflict serializable]; A --- C[View serializable];
```

Conflict
serializable

View
serializable

T ₁	T ₂
R(A)	
R(B)	

T ₁	T ₂
	R(B)
R(A)	

T ₁	T ₂
R(A)	
W(A)	

T ₁	T ₂
	W(A)
R(A)	

T ₁	T ₂
R(A)	
W(B)	

T ₁	T ₂
	W(B)
R(A)	

T ₁	T ₂
	R(A)
W(A)	

T ₁	T ₂
W(A)	
R(A)	

T ₁	T ₂
R(A)	
R(A)	

T ₁	T ₂
	R(A)
R(A)	

T ₁	T ₂
	W(A)
W(A)	

T ₁	T ₂
W(A)	
W(A)	

Conflict equivalent – if one schedule can be converted to another schedule by swapping of non- conflicting instruction then they are called conflict equivalent schedule.

T_1	T_2
R(A)	
A=A-50	
	R(B)
	B=B+50
R(B)	
B=B+50	
	R(A)
	A=A+10

T_1	T_2
	R(B)
	B=B+50
R(A)	
A=A-50	
R(B)	
B=B+50	
	R(A)
	A=A+10

SERIALIZABILITY

- **Conflicting instructions** - Let I and J be two consecutive instructions belonging to two different transactions T_i and T_j in a schedule S, the possible I and J instruction can be as-
 - I = READ(Q), J = READ(Q) -> *Non-conflicting*
 - I = READ(Q), J = WRITE(Q) -> *Conflicting*
 - I = WRITE(Q), J = READ(Q) -> *Conflicting*
 - I = WRITE(Q), J = WRITE(Q) -> *Conflicting*
- So, the instructions I and J are said to be conflicting, if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

CONFLICT SERIALIZABLE

- The schedules which are conflict equivalent to a serial schedule are called conflict serializable schedule.
- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**. A schedule S is ***conflict serializable***, if it is conflict equivalent to a serial schedule.

T_1	T_2
read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)

T_1	T_2
read(A) write(A) read(B) write(B)	read(A) write(A) read(B) write(B)

Q Consider the following schedule for transactions T_1 , T_2 and T_3 : what is the correct serialization of the above?

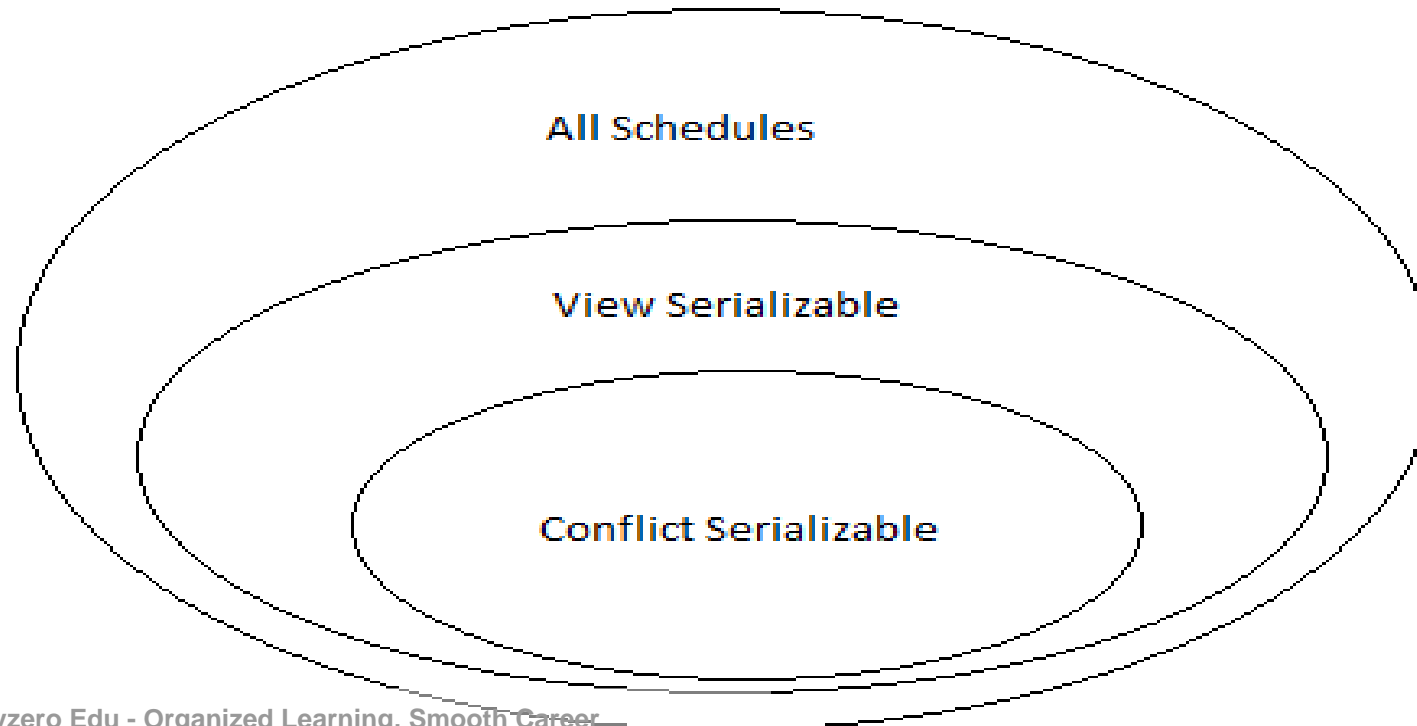
<u>T1</u>	<u>T2</u>	<u>T3</u>
Read (X)		
	Read (Y)	
		Read (Y)
	Write (Y)	
Write (X)		
		Write (X)
	Read (X)	
	Write (X)	

Procedure for determining conflict serializability of a schedule

- It can be determined using PRECEDENCE GRAPH method:
- A precedence graph consists of a pair $G(V, E)$
- V = set of vertices consisting of all the transactions participating in the schedule.
- E = set of edges consists of all edges $T_i \rightarrow T_j$, for which one of the following conditions holds:
 - T_i executes write(Q) before T_j executes read(Q)
 - T_i executes read(Q) before T_j executes write(Q)
 - T_i executes write(Q) before T_j executes write(Q)
- If an edge $T_i \rightarrow T_j$ exists in the precedence graph, then in any serial schedule S' equivalent to S , T_i must appear before T_j .
- **If the precedence graph for S has no cycle, then schedule S is conflict serializable, else it is not.**

VIEW SERIALIZABLE

- If a schedule is not conflict serializable, still it can be consistent, so let us study a weaker form of serializability called View serializability, and even if a schedule is view serializable still it can be consistent.
- If a schedule is conflict serializable then it will also be view serializable, so we must check view serializability only if a schedule is not conflict serializable.



- Two schedules S and S' are view equivalent, if they satisfy following conditions –
 - For each data item Q, if the transaction T_i reads the initial value of Q in schedule S, then the transaction T_i must, in schedule S', also read the initial value of Q.
 - If a transaction T_i in schedule S reads any data item Q, which is updated by transaction T_j , then a transaction T_i must in schedule S' also read data item Q updated by transaction T_j in schedule S'.
 - For each data item Q, the transaction (if any) that performs the final write(Q) operation in schedule S, then the same transaction must also perform final write(Q) in schedule S'.

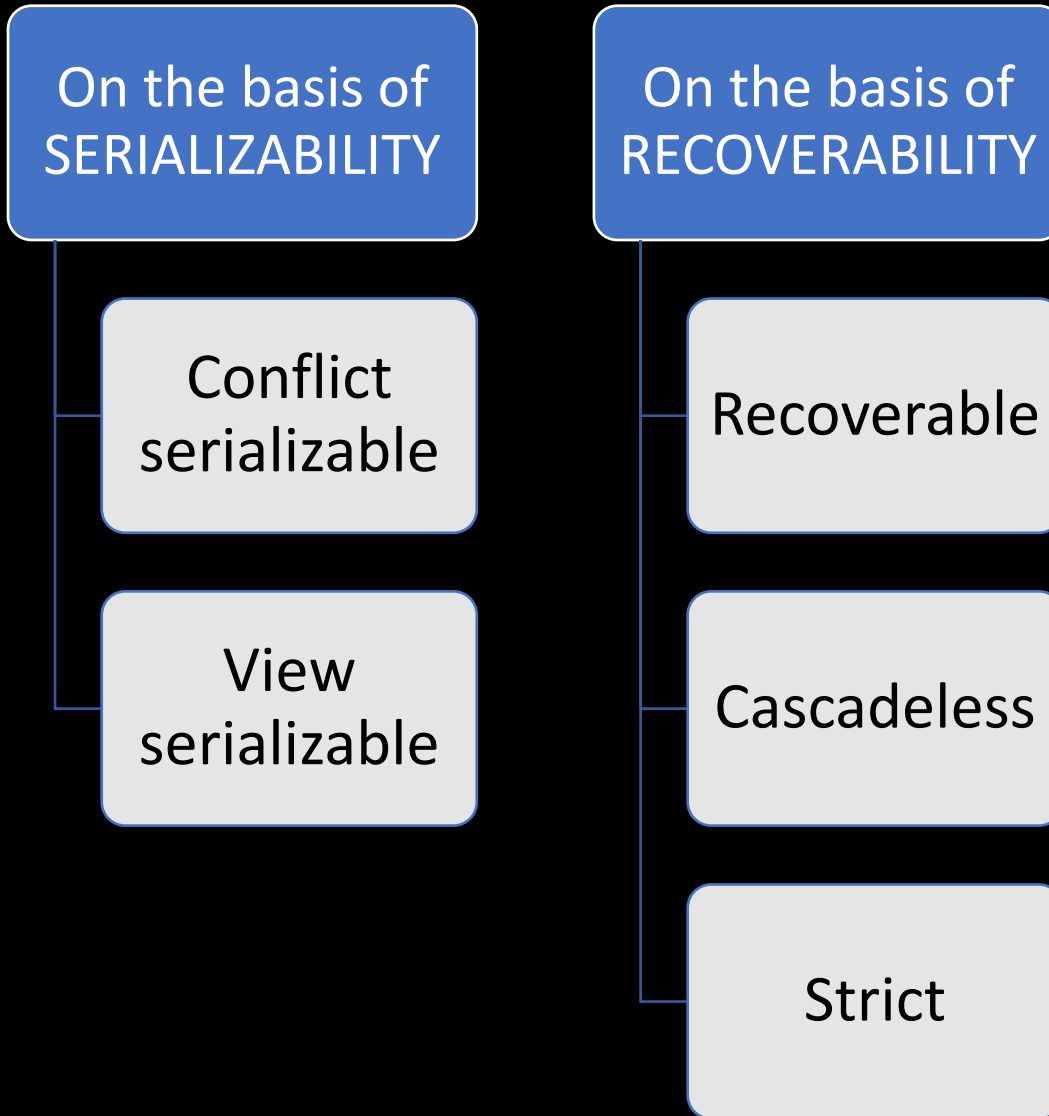
T_1	T_2
read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)

T_1	T_2
read(A) write(A) read(B) write(B)	read(A) write(A) read(B) write(B)

View Serializable

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

Schedule A				Serial schedule <T3,T4,T6>		
T3	T4	T6		T3	T4	T6
read(Q)				read(Q)		
	write(Q)			write(Q)		
write(Q)					write(Q)	
		write(Q)				write(Q)



NON- RECOVERABLE Vs RECOVERABLE SCHEDULE

- A schedule in which for each pair of transaction T_i and T_j , such that if T_j reads a data item previously written by T_i , then the commit or abort operation of T_i appears before T_j . Such a schedule is called Non- Recoverable schedule.
- A schedule in which for each pair of transaction T_i and T_j , such that if T_j reads a data item previously written by T_i , then the commit or abort of T_i must appear before T_j . Such a schedule is called Recoverable schedule.

S	
T_1	T_2
R(X)	
W(X)	
	R(X)
	C
C	

S	
T_1	T_2
R(X)	
W(X)	
	R(X)
C	
	C

CASCADING ROLLBACK Vs CASCADELESS SCHEDULE

- It is a phenomenon, in which even if the schedule is recoverable, the a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback. Cascading rollback is undesirable, since it leads to undoing of a significant amount of work. Uncommitted reads are not allowed in cascade less schedule.
- A schedule in which for each pair of transactions T_i and T_j , such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read operation of T_j . Such a schedule is called cascade less schedule.

S		
T_1	T_2	T_3
R(X)		
W(X)		
	R(X)	
	W(X)	
		R(X)
C		
	C	
		C

S		
T_1	T_2	T_3
R(X)		
W(X)		
C		
	R(X)	
	W(X)	
	C	
		R(X)
		C

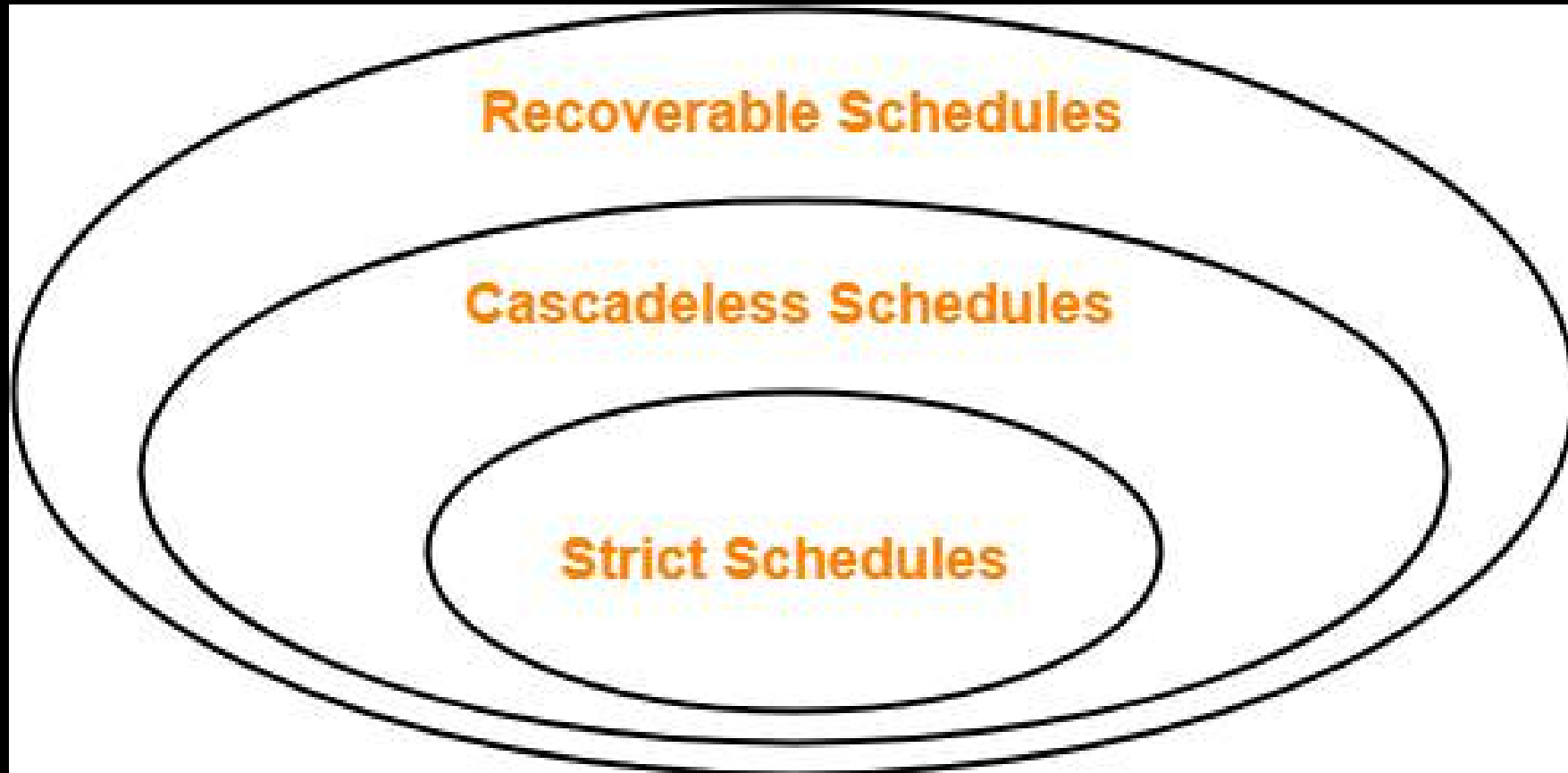
Strict Schedule

- A schedule in which for each pair of transactions T_i and T_j , such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read and write operation of T_j .

S_1	
T_1	T_2
R(a)	
W(a)	
	W(a)
C	
	R(a)
	C

S_2	
T_1	T_2
R(a)	
W(a)	
C	
	W(a)
	R(a)
	C

S_3	
T_1	T_2
R(a)	
	R(b)
W(a)	
	W(b)
C	
	R(a)
	C



Log based Recovery

- The system log, or transaction log, records all the changes or activities happening in the database, ensuring that transactions are durable and can be recovered in the event of a failure.
- Different types of log records represent various stages or actions in a transaction.
 - $\langle T_i \text{ start} \rangle$: This log entry indicates that the transaction T_i has started its execution.
 - $\langle T_i, X_j, V_1, V_2 \rangle$: This log entry documents a write operation. It states that transaction T_i has changed the value of data item X_j from V_1 to V_2 .
 - $\langle T_i \text{ commit} \rangle$: This log entry marks the successful completion of transaction T_i .
 - $\langle T_i \text{ abort} \rangle$: This log entry denotes that the transaction T_i has been aborted, either due to an error or a rollback operation.
- Before any write operation modifies the database, a log record of that operation needs to be created. This is to ensure that in case of a failure, the system can restore the database to a consistent state using the log records.

Aspect	Deferred Database Modification	Immediate Database Modification
Write Operation Timing	Only at the commit point.	As soon as changes occur.
I/O Operations	Reduced, as changes are batched and written at once during the commit, saving on intermediate I/O operations.	Increased, as each change triggers immediate write operations, leading to more frequent I/O operations.
Recovery Complexity	Simpler, as uncommitted changes are not reflected in the database, making rollback easier in case of failures.	More complex, as it may require undoing changes from uncommitted transactions that have been written to the database.

Shadow Paging Recovery Technique

In the shadow paging recovery technique, the database maintains two page tables during a transaction: the current page table (reflecting the state before the transaction began) and the shadow page table (which tracks changes made during the transaction). Here's how it operates:

- **Initialization:** When a transaction begins, the database creates a shadow copy of the page table. The database pages themselves are not duplicated; only the page table entries are duplicated.
- **Modifications:** As the transaction progresses, any changes are reflected in the current page table, while the shadow page table retains the original state. If a page is modified, a new copy of the page is created, and the current page table is updated to point to this new version, thereby ensuring that the shadow page table still points to the original unmodified page.

- **Commit:** Upon transaction commit, the shadow page table is discarded, and the current page table becomes the new committed state of the database. The database atomically switches to using the current page table, ensuring that changes are installed all at once.
- **Recovery:** In case of a system failure before the transaction commits, the database can easily recover by discarding the current page table and reverting back to the shadow page table, thereby restoring the database to its state before the transaction began.

Data fragmentation

- Data fragmentation in the context of a Database Management System (DBMS) refers to the process of breaking down a database into smaller, manageable pieces, and distributing these pieces across a network of computers. This is a significant component in distributed database systems.
 - **Horizontal Fragmentation**: In DBMS, horizontal fragmentation divides a table into sections based on rows. Each fragment contains a subset of rows, usually segmented based on certain conditions or attributes. This can enhance performance by facilitating parallel processing and quicker data retrieval, especially useful in distributed database systems.
 - **Vertical Fragmentation**: This fragmentation type divides a database table by columns, with different sets of columns stored in separate fragments. This approach is employed when different users require access to varied data attributes, promoting efficient data handling and minimizing data transfer times, which is advantageous in scenarios where queries only necessitate a subset of attributes.
 - **Hybrid Fragmentation**: Hybrid or mixed fragmentation in DBMS combines both horizontal and vertical strategies, optimizing data storage and retrieval for complex access patterns. This method can potentially offer performance improvements by utilizing the merits of both horizontal and vertical fragmentation, and is typically implemented in databases with complex, multifaceted data access requirements.

Distributed database

- A distributed database is a database in which storage devices are not all attached to a common processor. It may be stored in multiple computers, located in the same physical location, or dispersed over a network of interconnected computers.
- **Data Replication**
 - **Advantages:** Increased Availability, Improved Performance, Enhanced Reliability
 - **Disadvantages:** Storage Costs, Maintenance Complexity, Write Complexity.
- **Data Fragmentation**
 - **Advantages:** Efficient Data Access, Distributed Processing, Localized Management.
 - **Disadvantages:** Complexity, Dependency on Network, Reconstruction Issues.

CONCURRENCY CONTROL

Here we will study those protocol which guarantee to generate schedule which always satisfy desirable properties like conflict serializability. Along with we desire the following properties from schedule generating protocols

- Concurrency should be as high as possible, as this is our ultimate goal because of which we are making all the effort.
- The time taken by a transaction should also be less.
- Easy to understand and implement.

- **Time stamping based method:** - Where before entering the system, a specific order is decided among the transaction, so in case of a clash we can decide which one to allow and which to stop.
- **Lock based method:** - where we ask a transaction to first lock a data item before using it. So that no different transaction can use a data at the same time, removing any possibility of conflict.
 - 2 phase locking
 1. Basic 2pl
 2. Conservative 2pl
 3. Rigorous 2pl
 4. Strict 2pl
 - Graph based protocol
- **Validation based protocol** – Majority of transactions are read only transactions, the rate of conflicts among the transaction may be low, thus many of transaction, if executed without the supervision of a concurrency control scheme, would nevertheless leave the system in a consistent state.

TIME STAMP ORDERING PROTOCOL

- Basic idea of time stamping is to decide the order between the transaction before they enter in the system using a stamp (time stamp), in case of any conflict during the execution order can be decided using the time stamp.
- Let's understand how this protocol works, here we have two idea of timestamping, one for the transaction, and other for the data item.

- Time stamp for transaction,
 - With each transaction t_i , in the system, we associate a unique fixed timestamp, denoted by $TS(t_i)$.
 - This timestamp is assigned by database system to a transaction at time transaction enters into the system.
 - If a transaction has been assigned a timestamp $TS(t_i)$ and a new transaction t_j , enters into the system with a timestamp $TS(t_j)$, then always $TS(t_i) < TS(t_j)$.

- Two things are to be noted
 1. First time stamp of a transaction remain fixed throughout the execution
 2. Second it is unique means no two transaction can have the same timestamp.

- Time stamp with data item, in order to assure such scheme, the protocol maintains for each data item Q two timestamp values:
 1. **W-timestamp(Q)** is the largest time-stamp of any transaction that executed write(Q) successfully.
 2. **R-timestamp(Q)** is the largest time-stamp of any transaction that executed read(Q) successfully.
- These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

- Suppose a transaction T_i request a ***read(Q)***
 1. If **$TS(T_i) < W\text{-timestamp}(Q)$** , then T_i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T_i is rolled back.
 2. If **$TS(T_i) \geq W\text{-timestamp}(Q)$** , then the read operation is executed, and $R\text{-timestamp}(Q)$ is set to the maximum of $R\text{-timestamp}(Q)$ and $TS(T_i)$.

- Suppose that transaction T_i issues ***write(Q)***.
 1. If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and T_i is rolled back.
 2. If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q . Hence, this write operation is rejected, and T_i is rolled back.
 3. If $TS(T_i) \geq R\text{-timestamp}(Q)$, then the write operation is executed, and $W\text{-timestamp}(Q)$ is set to $\max(W\text{-timestamp}(Q), TS(T_i))$.
 4. If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the write operation is executed, and $W\text{-timestamp}(Q)$ is set to $\max(W\text{-timestamp}(Q), TS(T_i))$.

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule					
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

THOMAS WRITE RULE

- Thomas write is an improvement in time stamping protocol, which makes some modification and may generate those protocols that are even view serializable, because it allows greater potential concurrency.
- It is a Modified version of the timestamp-ordering protocol in which Blind write operations may be ignored under certain circumstances.
- The protocol rules for read operations remain unchanged. while for write operation, there is slightly change in Thomas write rule than timestamp ordering protocol.

When T_i attempts to write data item Q ,

- if $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of $\{Q\}$. Rather than rolling back T_i as the timestamp ordering protocol would have done, this {write} operation can be ignored.

- This modification is valid as the any transaction with $TS(T_i) < W\text{-timestamp}(Q)$, the value written by this transaction will never be read by any other transaction performing Read(Q) ignoring such obsolete write operation is considerable.
- Thomas' Write Rule allows greater potential concurrency. Allows some view-serializable schedules that are not conflict serializable.

T_3	T_4	T_6
read(Q)	write(Q)	
write(Q)		
		write(Q)

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

Lock Based Protocols

- To ensure isolation is to require that data items be accessed in a mutually exclusive manner i.e. while one transaction is accessing a data item, no other transaction can modify that data item. Locking is the most fundamental approach to ensure this.
- Lock based protocols ensure this requirement. Idea is first obtain a lock on the desired data item then if lock is granted then perform the operation and then unlock it.

- In general, we support two modes of lock because, to provide better concurrency.
- **Shared mode**
 - If transaction T_i has obtained a shared-mode lock (denoted by S) on any data item Q, then T_i can read, but cannot write Q, any other transaction can also acquire a shared mode lock on the same data item (this is the reason we called this shared mode).
- **Exclusive mode**
 - If transaction T_i has obtained an exclusive-mode lock (denoted by X) on any data item Q, then T_i can both read and write Q, any other transaction cannot acquire either a shared or exclusive mode lock on the same data item. (this is the reason we called this exclusive mode)

Lock –Compatibility Matrix

- Conclusion shared is compatible only with shared while exclusive is not compatible either with shared or exclusive.
- To access a data item, transaction T_i must first lock that item, if the data item is already locked by another transaction in an incompatible mode, or some other transaction is already waiting in non-compatible mode, then concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. The lock is then granted.

Current State of lock of data items					
Requested Lock			Exclusive	Shared	Unlocked
		Exclusive	N	N	Y
		Shared	N	Y	Y
		Unlock	Y	Y	-

AT:

Onebyzero Edu - Organized Learning, Smooth Career

The Comprehensive Academic Study Platform for University Students in <https://www.onebyzeroedu.com/>

- Lock based protocol do not ensure serializability as granting and releasing of lock do not follow any order and any transaction any time may go for lock and unlock. Here in the example below we can see, that even this transaction in using locking but neither it is conflict serializable nor independent from deadlock.

T ₁	T ₂
LOCK-X(A)	
READ(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK-S(B)
	READ(B)
	UNLOCK(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
UNLOCK(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(A)

- If we do not use locking, or if we unlock data items too soon after reading or writing them, we may get inconsistent states, as there exists a possibility of dirty read. On the other hand, if we do not unlock a data item before requesting a lock on another data item, concurrency will be poor.
- We shall require that each transaction in the system follow a set of rules, called a **locking protocol**, indicating when a transaction may lock and unlock each of the data items for e.g. 2pl or graph based locking.
- Locking protocols restrict the number of possible schedules.

Two phase locking protocol(2PL)

- The protocol ensures that each transaction issue lock and unlock requests in two phases, note that each transaction will be 2 phased not schedule.
- Growing phase- A transaction may obtain locks, but not release any locks.
- Shrinking phase- A transaction may release locks, but may not obtain any new locks.

- Initially a transaction is in growing phase and acquires lock as needed and in between can perform operation reach to lock point and once a transaction releases a lock, it can issue no more lock requests i.e. it enters the shrinking phase.

T ₁	T ₂
LOCK-X(A)	
READ(A)	
WRITE(A)	
	LOCK-S(B)
	READ(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(B)
UNLOCK(A)	
UNLOCK(B)	
	UNLOCK(A)

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

Properties

- 2PL ensures conflict serializability, and the ordering of transaction over lock points is itself a serializability order of a schedule in 2PL.
- If a schedule is allowed in 2PL protocol then definitely it is always conflict serializable. But it is not necessary that if a schedule is conflict serializable then it will be generated by 2pl. Equivalent serial schedule is based on the order of lock points.
- View serializability is also guaranteed.
- Does not ensure freedom from deadlock
- May cause non-recoverability.
- Cascading rollback may occur.

Conservative 2PL

- The idea is there is no growing phase transaction start directly from lock point, i.e. transaction must first acquire all the required locks then only it can start execution. If all the locks are not available then transaction must release the acquired locks and must wait.
- Shrinking phase will work as usual, and transaction can unlock any data item anytime.
- we must have a knowledge in future to understand what is data required so that we can use it

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL					
Strict 2PL					

RIGOROUS 2PL

- Requires that all locks be held until the transaction commits.
- This protocol requires that locking be two phase and also all the locks taken be held by transaction until that transaction commit.
- Hence there is no shrinking phase in the system.

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL	YES	YES	YES	YES	NO
Strict 2PL					

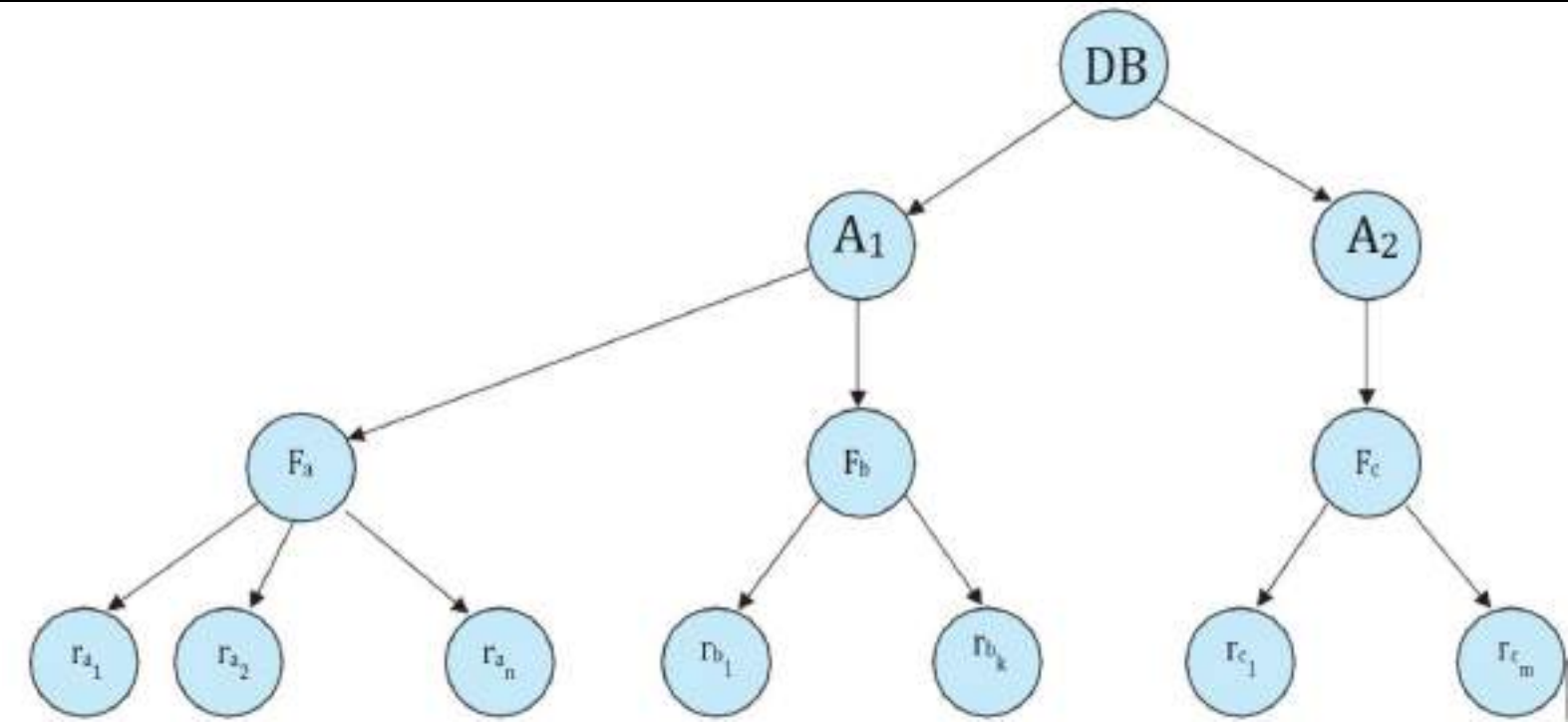
STRICT 2PL

- that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.
- This protocol requires that locking be two phase and also that exclusive –mode locks taken by transaction be held until that transaction commits.
- So it is simplified form of rigorous 2pl

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL	YES	YES	YES	YES	NO
Strict 2PL	YES	YES	YES	YES	NO

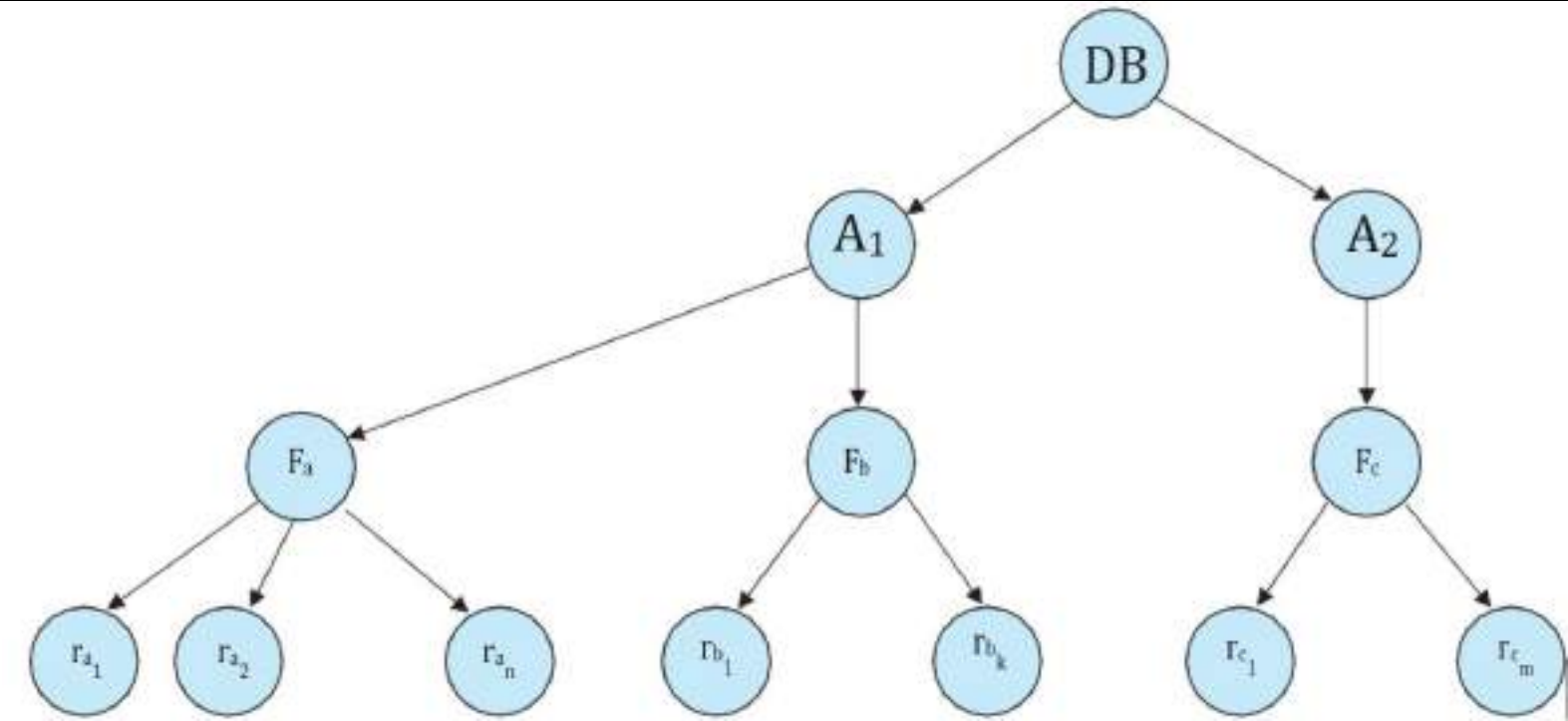
Multiple Granularity

	IS	IX	S	X
IS				
IX				
S				
X				



Multiple Granularity

	IS	IX	S	X
IS	true	true	true	false
IX	true	true	false	false
S	true	false	true	false
X	false	false	false	false



Validation-Based Protocols

- In cases where a majority of transactions are read-only transactions, the rate of conflicts among transactions may be low.
- Thus, many of these transactions, if executed without the supervision of a concurrency-control scheme, would nevertheless leave the system in a consistent state.
- A concurrency-control scheme imposes overhead of code execution and possible delay of transactions. It may be better to use an alternative scheme that imposes less overhead.
- A difficulty in reducing the overhead is that we do not know in advance which transactions will be involved in a conflict. To gain that knowledge, we need a scheme for *monitoring* the system.
- The **validation protocol** requires that each transaction T_i executes in two or three different phases in its lifetime, depending on whether it is a read-only or an update transaction. The phases are, in order:

- **Read phase.** During this phase, the system executes transaction T_i . It reads the values of the various data items and stores them in variables local to T_i . It performs all write operations on temporary local variables, without updates of the actual database.
- **Validation phase.** The validation test (described below) is applied to transaction T_i . This determines whether T_i is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction.
- **Write phase.** If the validation test succeeds for transaction T_i , the temporary local variables that hold the results of any write operations performed by T_i are copied to the database. Read-only transactions omit this phase.

COMPLETE GATE COURSE

DBMS

IN

12
Hours

+

FREE
NOTES



AVAILABLE AT:

Onebyzero Edu - Organized Learning, Smooth Career

Comprehensive Academic Study Platform for University Students in Bangladesh (www.onebyzeroedu.com)

Database

- Core subjects for CS/IT Students
- In GATE 7-8 Marks out of 100 Marks, and 5-6 questions on an average
- Needs less time, very scoring
- Mostly numerical questions
- Indirect Application in Industry, good future

<http://www.knowledgegate.in/gate>

Syllabus

Section 9: Databases

ER-model. Relational model: relational algebra, tuple calculus, SQL. Integrity constraints, normal forms. File organization, indexing (e.g., B and B+ trees). Transactions and concurrency control.

- Transactions and concurrency control
- ER-model
- Relational model
- Integrity constraints
- Normal forms
- File organization, indexing (e.g., B and B+ trees)
- Relational algebra, tuple calculus, SQL

<http://www.knowledgegate.in/gate>

Video chapters

- Chapter-1 (Bscics Transactions and concurrency control)
- Chapter-2 (ER-model)
- Chapter-3 (Relational model, Functional Dependencies, Keys)
- Chapter-4 (Normalization)
- Chapter-5 (File organization, indexing (e.g., B and B+ trees)
- Chapter-6 (Relational algebra, tuple calculus, SQL)

<http://www.knowledgegate.in/gate>

Chapter-1

(Basics Transactions and concurrency control)

<http://www.knowledgegate.in/gate>

Data

- Data are characteristics, usually numerical, that are collected through observation. In a more technical sense, data are a set of values of qualitative or quantitative variables about one or more persons or objects, while a datum (singular of data) is a single value of a single variable.
- Any facts and figures about an entity is called as Data.



AVAILABLE AT:

Onebyzero Edu - Organized Learning, Smooth Career
The Comprehensive Academic Study Platform for University Students in Bangladesh (www.onebyzeroedu.com)

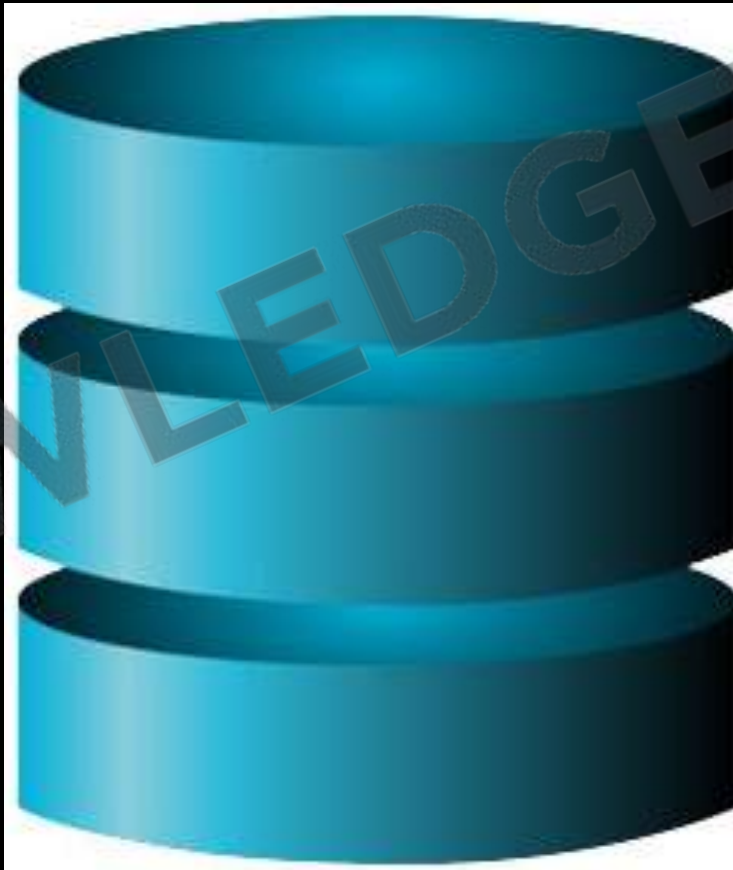
Information

- Although the terms "data" and "information" are often used interchangeably, these terms have distinct meanings. In some popular publications, data are sometimes said to be transformed into information when they are viewed in context or in post-analysis.
- Processed Data is called information.



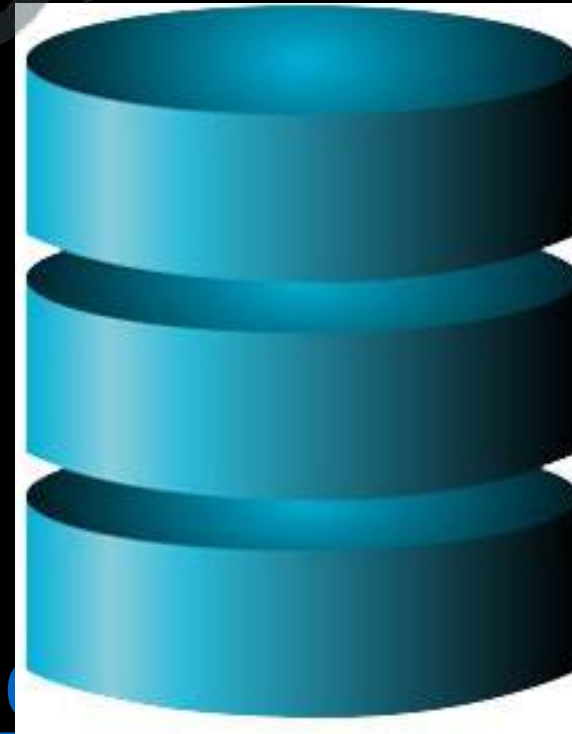
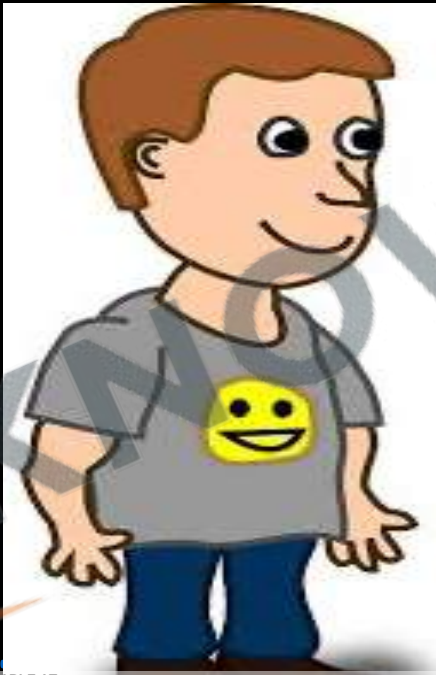
Data Base

- A **database** is an organized collection of data, generally stored and accessed electronically from a computer system.



Data Base Management System

- The database management system (DBMS) is the software that interacts with end users, applications, and the database itself to capture and analyze the data.
- The DBMS software additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system".



Problem With File System

1. **Data redundancy and inconsistency**: Imagine two spreadsheets that list employee contacts, but one is updated with a new phone number while the other is not, leading to confusion about the correct number to use.
2. **Difficulty in accessing Data**: Consider needing a specific report from a database, but you have to request it from an IT specialist instead of getting it yourself directly, causing delays.
3. **Data Isolation**: If customer profiles are stored in one system and their order histories in another, merging this information for a comprehensive view can be cumbersome.
4. **Integrity Problem**: After a customer updates their address on one page of a website, they find the old address on another page because the change did not carry through all systems.
5. **Atomicity Problem**: A banking transaction that deducts money from one account but fails to deposit it in another leaves the transaction incomplete and accounts unbalanced.
6. **Concurrent access Anomalies**: Two people booking the last ticket on a flight at the same time might both get a confirmation, but only one can actually get the seat.

Instance and Schemas

- The collection of information stored in the database at a particular moment is called an instance of the database.
- The overall design of the database is called the database schema.

TRANSACTION

- Why we study transaction?
 - According to general computation principle (operating system) we may have partially executed program, as the level of atomicity is instruction i.e. either an instruction is executed completely or not
 - But in DBMS view, user perform a logical work(operation) which is always atomic in nature i.e. either operation is execute or not executed, there is no concept like partial execution. For example, Transaction T_1 which transfer 100 units from account A to B

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)



- In this transaction if a failure occurs after Read(B) then the final statue of the system will be inconsistent as 100 units are debited from account A but not credited in account B, this will generate inconsistency.
- Here for 'consistency' before $(A + B) == \text{after } (A + B)$

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)



What is transaction

- To remove this partial execution problem, we increase the level of atomicity and bundle all the instruction of a logical operation into a unit called transaction.
- So formally 'A transaction is a Set of logically related instructions to perform a logical unit of work'.

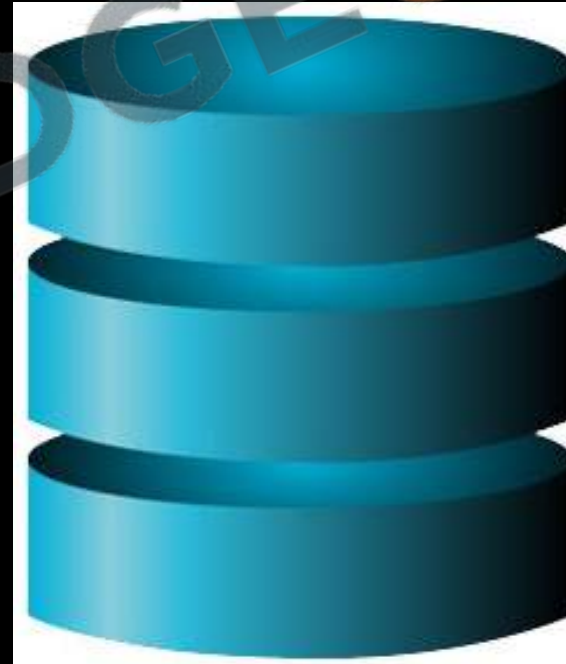
T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)

- As here we are only concerned with DBMS so we will only two basic operation on database
- **READ (X)** - Accessing the database item x from disk (where database stored data) to memory variable also name as X.
- **WRITE (X)** - Writing the data item from memory variable X to disk.

Desirable properties of transaction

- Now as the smallest unit which have atomicity in DBMS view is transaction, so if want that our data should be consistent then instead of concentrating on data base, we must concentrate on the transaction for our data to be consistent.

T_1
Read(A)
$A = A - 100$
Write(A)
Read(B)
$B = B + 100$
Write(B)



- Transactions should possess several properties, often called the **ACID** properties; to provide integrity and consistency of the data in the database. The following are the ACID properties:

A = Atomicity
C = Consistency
I = Isolation
D = Durability

<http://www.knowledgegate.in/gate>

- **Atomicity** - A transaction is an atomic unit of processing; it should either be performed in its entirety or not performed at all.
- It is the responsibility of ***recovery control manager / transaction control manager of DBMS*** to ensure atomicity

<http://www.knowledgegate.in/gate>

- **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.
- The definition of consistency may change from one system to another. The preservation of consistency of database is the responsibility of programmers(users) or the DBMS modules that enforces integrity constraints.

- **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 by any other transactions executing concurrently.
- The isolation property of database is the responsibility of ***concurrency control manager of database***.

- **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. It is the responsibility of *recovery control manager of DBMS*.

<http://www.knowledgegate.in/gate>

Q NOT a part of the **ACID** properties of database transactions?

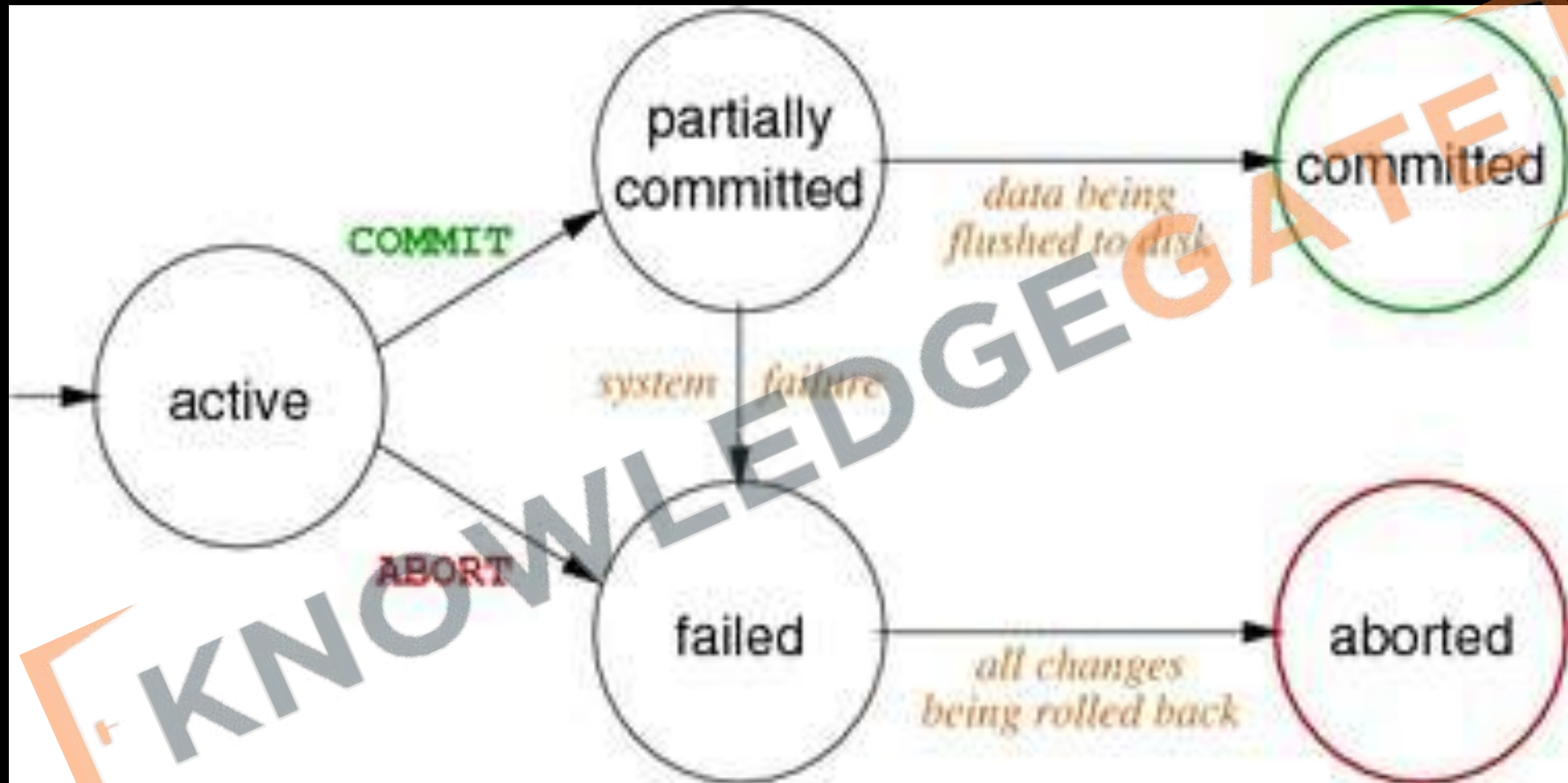
(a) Atomicity

(b) Consistency

(c) Isolation

(d) Deadlock-freedom

Transaction states



Transaction states

- **ACTIVE** - It is the initial state. Transaction remains in this state while it is executing operations.
- **PARTIALLY COMMITTED** - After the final statement of a transaction has been executed, the state of transaction is partially committed as it is still possible that it may have to be aborted (due to any failure) since the actual output may still be temporarily residing in main memory and not to disk.
- **FAILED** - After the discovery that the transaction can no longer proceed (because of hardware /logical errors). Such a transaction must be rolled back.
- **ABORTED** - A transaction is said to be in aborted state when the when the transaction has been rolled back and the database has been restored to its state prior to the start of execution.
- **COMMITTED** - A transaction enters committed state after successful completion of a transaction and final updation in the database.

Q if the transaction is in which of the state that we can guarantee that data base is in consistent state

a) aborted

b) committed

c) both aborted & committed

d) none

<http://www.knowledgegate.in/gate>

Why we need concurrent execution

- Concurrent execution is necessary because-
 - It leads to good database performance , less waiting time.
 - Overlapping I/O activity with CPU increases throughput and response time.



<http://www.knowledgegate.in/gate>

PROBLEMS DUE TO CONCURRENT EXECUTION OF TRANSACTION

- But interleaving of instructions between transactions may also lead to many problems that can lead to inconsistent database.
- Sometimes it is possible that even though individual transaction are satisfying the acid properties even though the final statues of the system will be inconsistent.



Lost update problem / Write - Write problem-

- If there is any two write operation of different transaction on same data value, and between them there is no read operations, then the second write over writes the first write.

T_1	T_2
Read(A)	
Write(A)	
	Write(A)
	Commit
Commit	

Dirty read problem/ Read -Write problem

- In this problem, the transaction reads a data item updated by another uncommitted transaction, this transaction may in future be aborted or failed.
- The reading transactions end with incorrect results.

T_1	T_2
Read(A)	
Write(A)	
	Read(A)
	Commit
Abort	

Unrepeatable read problem

- When a transaction tries to read a value of a data item twice, and another transaction updates the data item in between, then the result of the two read operation of the first transaction will differ, this problem is called, Non-repeatable read problem.

T_1	T_2
Read(A)	
	Read(A)
	Write(A)
Read(A)	

Phantom read problem

T_1	T_2
Read(A)	
	Delete(A)
Read(A)	

Q The following schedule is suffering from

- a) Lost update problem
- b) Unrepeatable read problem
- c) Both A and B
- d) Neither A and B

T ₁	T ₂
R(y)	
	R(x) R(y) $y = x + y$ w(y)
R(y)	

Q Which of the following scenarios may lead to an irrecoverable error in a database system?

- (A)** A transaction writes a data item after it is read by an uncommitted transaction
- (B)** A transaction reads a data item after it is read by an uncommitted transaction
- (C)** A transaction reads a data item after it is written by a committed transaction
- (D)** A transaction reads a data item after it is written by an uncommitted transaction

Solution is Schedule

- When two or more transaction executed together or one after another then they can be bundled up into a higher unit of execution called schedule.
- A **schedule** of n transactions T_1, T_2, \dots, T_n is an ordering of the operations of the transactions. Operations from different transactions can be interleaved in the schedule S .
- However, schedule for a set of transaction must contain all the instruction of those transaction, and for each transaction T_i that participates in the schedule S , the operations of T_i in S must appear in the same order in which they occur in T_i .

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

T_2	T_3
read(B)	
	read(B)
	write(B)
read(A)	
	read(A)
	write(A)

AVAILABLE AT:

- **Serial schedule** - A serial schedule consists of sequence of instruction belonging to different transactions, where instructions belonging to one single transaction appear together. Before complete execution of one transaction another transaction cannot be started.

T_0	T_1
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

- For a set of n transactions, there exist $n!$ different valid serial schedules. Every serial schedule lead database into consistent state. Throughput of system is less.

- **Non-serial schedule** - A schedule in which sequence of instructions of a transaction appear in the same order as they appear in individual transaction but the instructions may be interleaved with the instructions of different transactions i.e. concurrent execution of transactions takes place.

T_2	T_3
read(B)	read(B)
	write(B)
read(A)	
	read(A)
	write(A)

- So the number of schedules for n different transaction $T_1, T_2, T_3, \dots, T_N$ where each transaction contains $n_1, n_2, n_3, \dots, n_n$ respectively will be.
- $\{(n_1 + n_2 + n_3 + \dots + n_n)! / (n_1! n_2! n_3! \dots n_n!)\}$
- $\{(n_1 + n_2 + n_3 + \dots + n_n)! / (n_1! n_2! n_3! \dots n_n!)\} - n!$

- **Conclusion of schedules**
 - We do not have any method to proof that a schedule is consistent, but from the above discussion we understand that a serial schedule will always be consistent, so if somehow we proof that a non-serial schedule will also have same effects as of a serial schedule that we get a proof that, this particular non-serial schedule will also be consistent “find those schedules that are logically equal to serial schedules”.
 - For a concurrent schedule to result in consistent state, it should be equivalent to a serial schedule. i.e. it must be serializable.

On the basis of
SERIALIZABILITY

Conflict
serializable

View
serializable

Result
Equivalent

On the basis of
RECOVERABILITY

Recoverable

Cascadeless

Strict

T ₁	T ₂
R(A)	
R(B)	

T ₁	T ₂
	R(B)
R(A)	

T ₁	T ₂
R(A)	
W(A)	

T ₁	T ₂
	W(A)
R(A)	

T ₁	T ₂
R(A)	
W(B)	

T ₁	T ₂
	W(B)
R(A)	

T ₁	T ₂
	R(A)
W(A)	

T ₁	T ₂
W(A)	
R(A)	

T ₁	T ₂
R(A)	
R(A)	

T ₁	T ₂
	R(A)
R(A)	

T ₁	T ₂
	W(A)
W(A)	

T ₁	T ₂
W(A)	
W(A)	

SERIALIZABILITY

- **Conflicting instructions** - Let I and J be two consecutive instructions belonging to two different transactions T_i and T_j in a schedule S, the possible I and J instruction can be as-
 - I = READ(Q), J = READ(Q) -> *Non-conflicting*
 - I = READ(Q), J = WRITE(Q) -> *Conflicting*
 - I = WRITE(Q), J = READ(Q) -> *Conflicting*
 - I = WRITE(Q), J = WRITE(Q) -> *Conflicting*
- So, the instructions I and J are said to be conflicting, if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

Conflict equivalent – if one schedule can be converted to another schedule by swapping of non- conflicting instruction then they are called conflict equivalent schedule.

T_1	T_2
R(A)	
A=A-50	
	R(B)
	B=B+50
R(B)	
B=B+50	
	R(A)
	A=A+10

T_1	T_2
	R(B)
	B=B+50
R(A)	
A=A-50	
R(B)	
B=B+50	
	R(A)
	A=A+10

Q Consider a schedule of transactions T_1 and T_2 : Here, RX stands for “Read(X)” and WX stands for “Write(X)”. Which one of the following schedules is conflict equivalent to the above schedule?

a)

S	
T_1	T_2
	RB
	WB
	RD
RA	
RC	
WD	
WB	
	WC
Commit	
	Commit

b)

S	
T_1	T_2
RA	
RC	
WD	
WB	
	RB
	WB
	RD
	WC
Commit	
	Commit

c)

S	
T_1	T_2
RA	
RC	
WD	
	RB
	WB
	RD
WB	
	WC
Commit	
	Commit

d)

S	
T_1	T_2
	RB
	WB
	RD
	WC
RA	
RC	
WD	
WB	
Commit	
	Commit

S	
T_1	T_2
RA	
	RB
	WB
RC	
	RD
WD	
	WC
WB	
Commit	
	Commit

CONFLICT SERIALIZABLE

- The schedules which are conflict equivalent to a serial schedule are called conflict serializable schedule.
- If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instructions, we say that S and S' are **conflict equivalent**.
- A schedule S is ***conflict serializable***, if it is conflict equivalent to a serial schedule.

T_1	T_2
read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)

T_1	T_2
read(A) write(A) read(B) write(B)	read(A) write(A) read(B) write(B)

Procedure for determining conflict serializability of a schedule

- It can be determined using PRECEDENCE GRAPH method:
- A precedence graph consists of a pair $G (V, E)$
- V = set of vertices consisting of all the transactions participating in the schedule.
- E = set of edges consists of all edges $T_i \rightarrow T_j$, for which one of the following conditions holds:
 - T_i executes write(Q) before T_j executes read(Q)
 - T_i executes read(Q) before T_j executes write(Q)
 - T_i executes write(Q) before T_j executes write(Q)

- If an edge $T_i \rightarrow T_j$ exists in the precedence graph, then in any serial schedule S' equivalent to S , T_i must appear before T_j .
- **If the precedence graph for S has no cycle, then schedule S is conflict serializable, else it is not.** This cycle detection can be done by cycle detection algorithms, one of them based on depth first search takes $O(n^2)$ time.
- The serializability order of transactions of equivalent serial schedule can be determined using **topological order** in a precedence graph.

Q Let $R_i(z)$ and $W_i(z)$ denote read and write operations on a data element z by a transaction T_i , respectively. Consider the schedule S with four transactions.
 $S: R_4(x) R_2(x) R_3(x) R_1(y) W_1(y) W_2(x) W_3(y) R_4(y)$ Which one of the following serial schedules is conflict equivalent to S ?

(a) $T_1 \rightarrow T_3 \rightarrow T_4 \rightarrow T_2$

(b) $T_1 \rightarrow T_4 \rightarrow T_3 \rightarrow T_2$

(c) $T_4 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2$

(d) $T_3 \rightarrow T_1 \rightarrow T_4 \rightarrow T_2$

S			
T ₁	T ₂	T ₃	T ₄
			R(x)
	R(x)		
		R(x)	
R(y)			
W(y)			
	W(x)		
		W(y)	
			R(y)

Q Let $r_i(z)$ and $w_i(z)$ denote read and write operations respectively on a data item z by a transaction T_i . Consider the following two schedules.

- $S_1:r_1(x)r_1(y)r_2(x)r_2(y)w_2(y)w_1(x)$
- $S_2:r_1(x)r_2(x)r_2(y)w_2(y)r_1(y)w_1(x)$

Which one of the following options is correct?

- a) S_1 is conflict serializable, and S_2 is not conflict serializable
- b) S_1 is not conflict serializable, and S_2 is conflict serializable
- c) Both S_1 and S_2 are conflict serializable
- d) Neither S_1 nor S_2 is conflict serializable

S ₁	
T ₁	T ₂
R(X)	
R(Y)	
	R(X)
	R(Y)
	W(Y)
W(X)	

S ₂	
T ₁	T ₂
R(X)	
	R(X)
	R(Y)
	W(Y)
R(Y)	
W(X)	

Q Consider the following schedule for transactions T_1 , T_2 and T_3 : Which one of the schedules below is the correct serialization of the above?

(A) $T_1 \rightarrow T_3 \rightarrow T_2$

(B) $T_2 \rightarrow T_1 \rightarrow T_3$

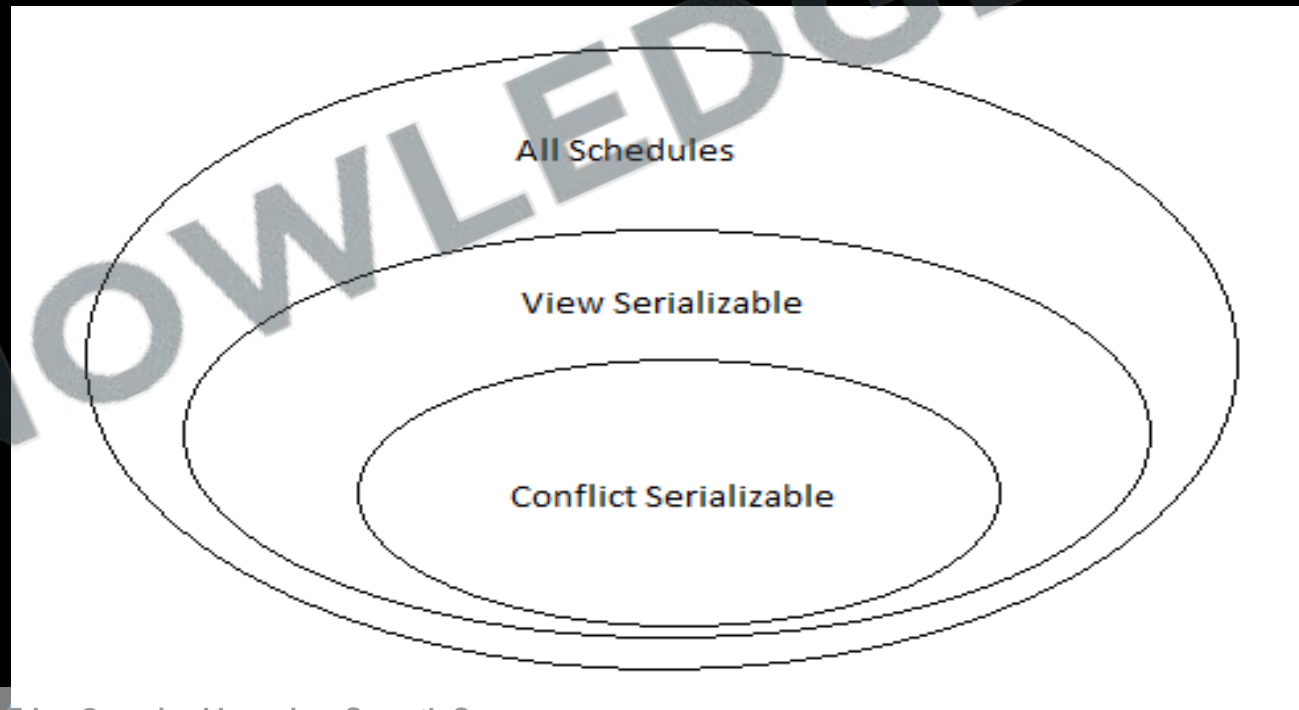
(C) $T_2 \rightarrow T_3 \rightarrow T_1$

(D) $T_3 \rightarrow T_1 \rightarrow T_2$

<u>T1</u>	<u>T2</u>	<u>T3</u>
Read (X)		
	Read (Y)	
		Read (Y)
	Write (Y)	
Write (X)		
		Write (X)
	Read (X)	
	Write (X)	

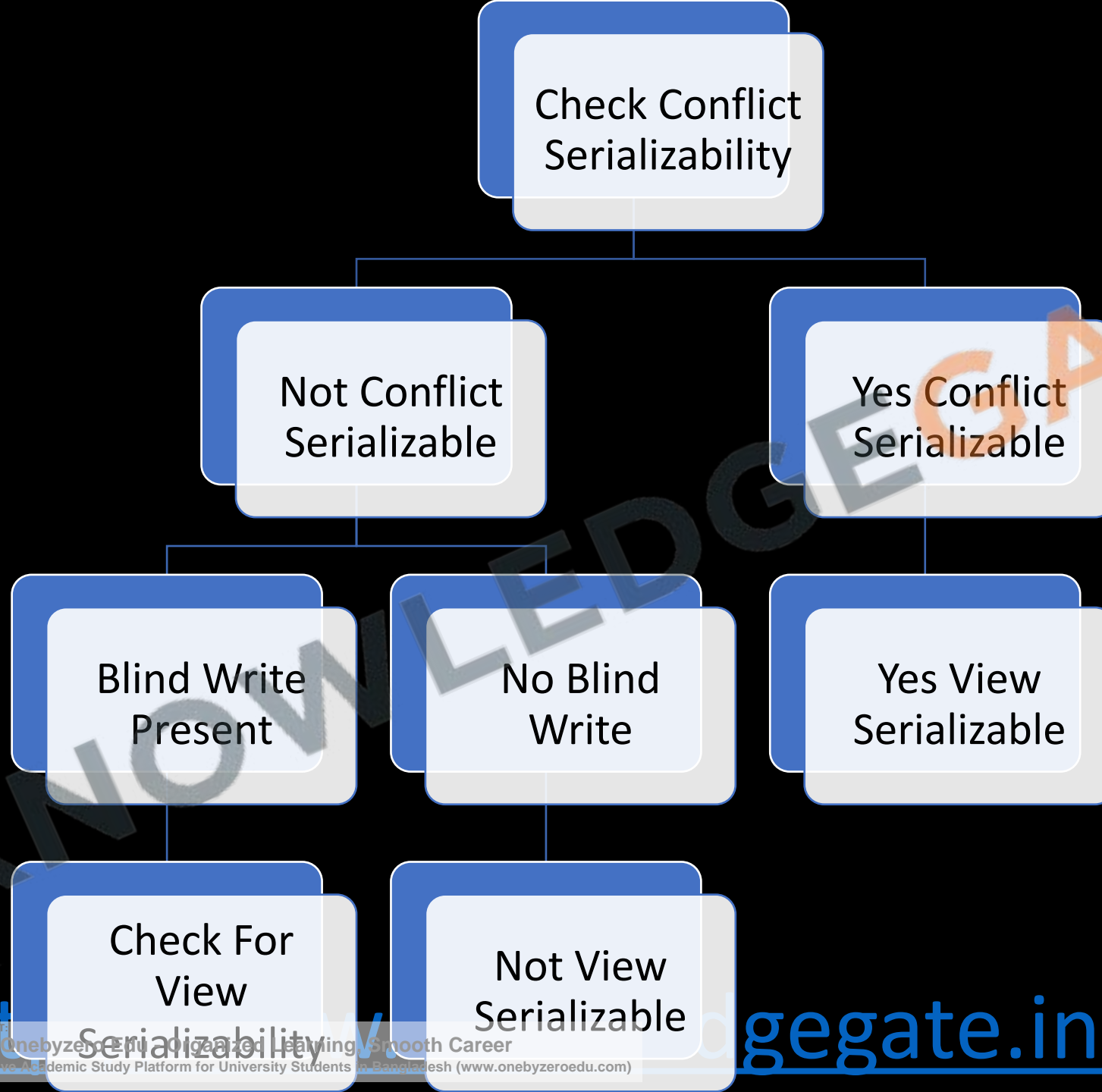
VIEW SERIALIZABLE

- If a schedule is not conflict serializable, still it can be consistent, so let us study a weaker form of serializability called View serializability, and even if a schedule is view serializable still it can be consistent.
- If a schedule is conflict serializable then it will also be view serializable, so we must check view serializability only if a schedule is not conflict serializable.



- if a schedule is not conflict serializable and it does not contain any blind write then it can never be view serializable, but if not conflict serializable and have blind write then may or may not be view serializable.
- If a schedule is not conflict serializable and if there exist a blind write.
 - First tabulate all serial schedules possible. Then check one by one whether given schedule is view equivalent to any of the serial schedule.
 - If yes then schedule is view serializable otherwise not.

<http://www.knowledgegate.in/gate>



- Two schedules S and S' are view equivalent, if they satisfy following conditions –
 - For each data item Q , if the transaction T_i reads the initial value of Q in schedule S , then the transaction T_i must, in schedule S' , also read the initial value of Q .
 - If a transaction T_i in schedule S reads any data item Q , which is updated by transaction T_j , then a transaction T_i must in schedule S' also read data item Q updated by transaction T_j in schedule S' .
 - For each data item Q , the transaction (if any) that performs the final write(Q) operation in schedule S , then the same transaction must also perform final write(Q) in schedule S' .

T_1	T_2
read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)

T_1	T_2
read(A) write(A) read(B) write(B)	read(A) write(A) read(B) write(B)

- Complexity wise finding the schedule is view serializable or not is a NP-complete problem.

View Serializable

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

Schedule A		
T3	T4	T6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

View Serializable

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

Schedule A				Serial schedule <T3,T4,T6>		
T3	T4	T6		T3	T4	T6
read(Q)				read(Q)		
	write(Q)			write(Q)		
write(Q)					write(Q)	
		write(Q)				write(Q)

- **BLIND WRITES**- In the above example, transaction T_4 and T_6 perform write operation on data item Q without accessing (reading the data item), such updation without knowing/accessing previous value of data item, are called Blind updation or **BLIND WRITE**.

Q A Schedule that is not conflict serializable and contains at least one blind write then the schedule is

a) Always view serializable

b) Always non-serializable

c) May be View serializable

d) None of the above

<http://www.knowledgegate.in/gate>

Q Which of the following statements (s) is/are TRUE?

S₁: All view serializable schedules are also conflict serializable.

S₂: All conflict serializable schedules are also view serializable.

S₃: If a schedule is not conflict serializable then it is not view serializable.

S₄: If a schedule is not view serializable then it is not conflict serializable.

a) S₁ and S₂ only

b) S₂ and S₃ only

c) S₂ and S₄ only

d) S₁ and S₃ only

<http://www.knowledgegate.in/gate>

Q Consider the following schedule 'S' with three transactions.

S: R₁(B); R₃(C); R₁ (A); W₂ (A); W₁(A), W₂ (B); W₃ (A); W₁ (B); W₃ (B), W₃ (C)

Which of the following is TRUE with respect to the above schedule?

- a) It is conflict serializable with sequence [T₁, T₂ T₃]
- b) It is conflict serializable with sequence [T₂, T₁ T₃]
- c) It is view serializable but not conflict serializable
- d) It is neither conflict serializable nor view serializable

S		
T ₁	T ₂	T ₃
R(B)		
		R(C)
R(A)		
	W(A)	
W(A)		
	W(B)	
		W(A)
W(B)		
		W(B)
		W(C)

NON- RECOVERABLE SCHEDULE

- A schedule in which for each pair of transaction T_i and T_j , such that if T_j reads a data item previously written by T_i , then the commit or abort operation of T_i appears before T_j . Such a schedule is called Non- Recoverable schedule.

S	
T_1	T_2
R(X)	
W(X)	
	R(X)
	C
C	

RECOVERABLE SCHEDULE

- A schedule in which for each pair of transaction T_i and T_j , such that if T_j reads a data item previously written by T_i , then the commit or abort of T_i must appear before T_j . Such a schedule is called Recoverable schedule.

S	
T_1	T_2
R(X)	
W(X)	
	R(X)
C	
	C

Q Consider the following schedule S of transactions T_1, T_2, T_3, T_4

Which one of the following statements is CORRECT?

- (A) S is conflict-serializable but not recoverable
- (B) S is not conflict-serializable but is recoverable
- (C) S is both conflict-serializable and recoverable
- (D) S is neither conflict-serializable nor is it recoverable

T1	T2	T3	T4
	Reads(X)		
		Writes(X)	
		Commit	
Writes(X)			
Commit			
	Writes(Y)		
	Reads(Z)		
	Commit		
			Reads(X)
			Reads(Y)
			Commit

Q Let S be the following schedule of operations of three transactions T1, T2 and T3 in a relational database system: R2(Y),R1(X),R3(Z),R1(Y)W1(X),R2(Z),W2(Y),R3(X),W3(Z), Consider the statements P and Q below:

P: S is conflict-serializable.

Q: If T3 commits before T1 finishes, then S is recoverable.

Which one of the following choices is correct?

(a) Both P and Q are true

(b) P is true and Q is false

(c) P is false and Q is true

(d) Both P and Q are false

S		
T ₁	T ₂	T ₃
	R(Y)	
R(X)		
		R(Z)
W(X)		W(Z)
	R(z)	
	W(Y)	
		R(X)
		W(z)

CASCADING ROLLBACK

- It is a phenomenon, in which a single transaction failure leads to a series of transaction rollbacks, is called cascading rollback. Even if the schedule is recoverable the, the commit of transaction may lead lot of transaction to rollback.
- Cascading rollback is undesirable, since it leads to undoing of a significant amount of work. Uncommitted reads are not allowed in cascade less schedule.

S		
T ₁	T ₂	T ₃
R(X)		
W(X)		
	R(X)	
	W(X)	
		R(X)
C		
	C	
		C

CASCADELESS SCHEDULE

- To avoid cascading rollback, cascade less schedule are used.
- A schedule in which for each pair of transactions T_i and T_j , such that if T_j reads a data item previously written by T_i then the commit or abort of T_i must appear before read operation of T_j . Such a schedule is called cascade less schedule.

S		
T_1	T_2	T_3
R(X)		
W(X)		
C		
	R(X)	
	W(X)	
	C	
		R(X)
		C



Q Consider the following partial Schedule S involving two transactions T_1 and T_2 . Only the read and the write operations have been shown. The read operation on data item P is denoted by read(P) and the write operation on data item P is denoted by write(P).

Suppose that the transaction T_1 fails immediately after time instance 9. Which one of the following statements is correct?

- (A) T_2 must be aborted and then both T_1 and T_2 must be re-started to ensure transaction atomicity
- (B) Schedule S is non-recoverable and cannot ensure transaction atomicity
- (C) Only T_2 must be aborted and then re-started to ensure transaction atomicity
- (D) Schedule S is recoverable and can ensure atomicity and nothing else needs to be done

Time	Transaction-id	
	<i>T1</i>	<i>T2</i>
1	<i>read(A)</i>	
2	<i>write(A)</i>	
3		<i>read(C)</i>
4		<i>write(C)</i>
5		<i>read(B)</i>
6		<i>write(B)</i>
7		<i>read(A)</i>
8		<i>commit</i>
9	<i>read(B)</i>	

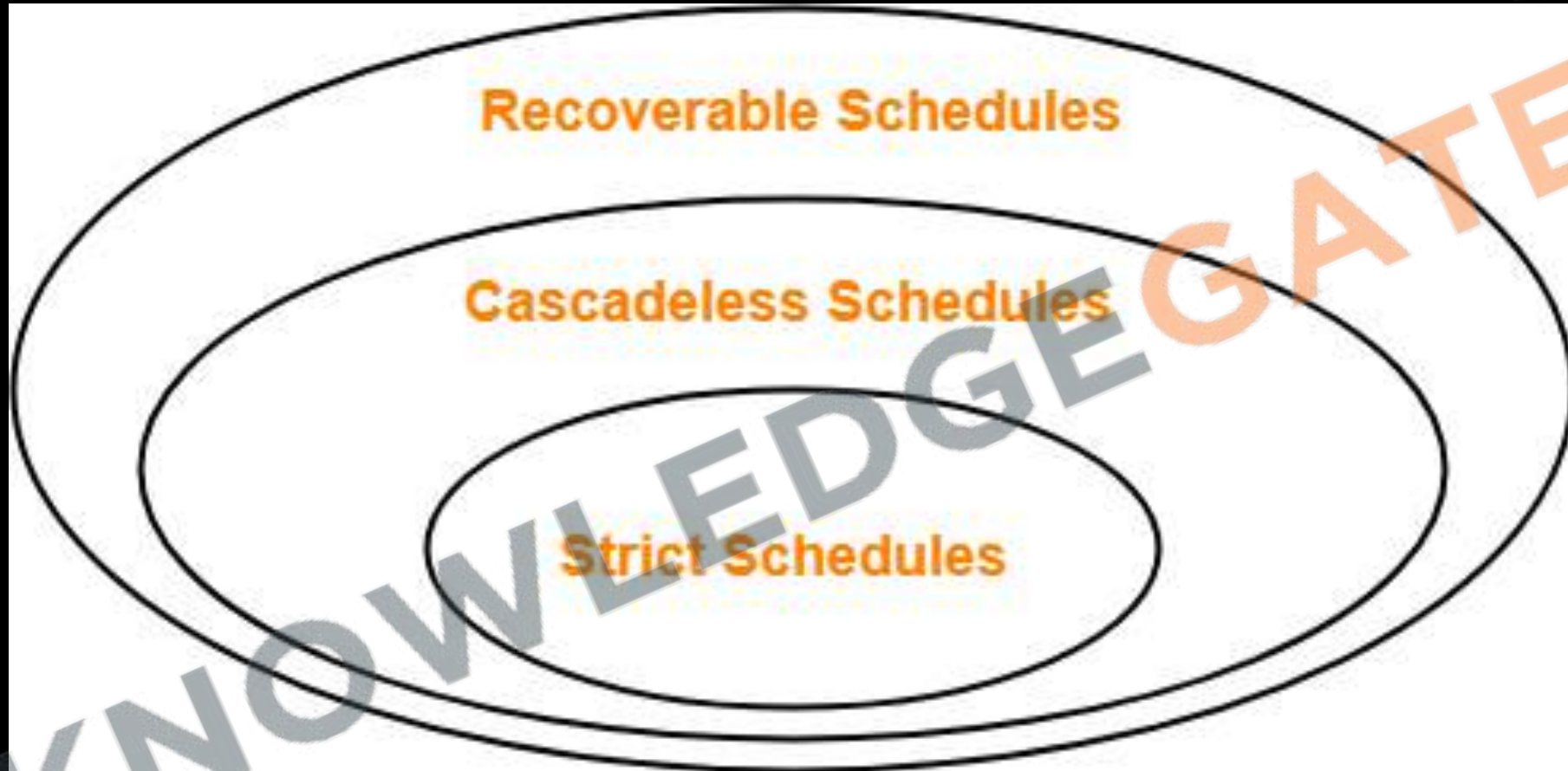
Strict Schedule

- A schedule in which for each pair of transactions T_i and T_j , such that if T_j reads a data item previously written by T_i , then the commit or abort of T_i must appear before read and write operation of T_j .

S_1	
T_1	T_2
R(a)	
W(a)	
	W(a)
C	
	R(a)
	C

S_2	
T_1	T_2
R(a)	
W(a)	
C	
	W(a)
	R(a)
	C

S_3	
T_1	T_2
R(a)	
	R(b)
W(a)	
	W(b)
C	
	R(a)
	C



<http://www.knowledgegate.in/gate>

Q Consider the following database schedule with two transactions, T_1 and T_2 . $S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$. where $r_i(Z)$ denotes a read operation by transaction T_i on a variable Z , $w_i(Z)$ denotes a write operation by T_i on a variable Z and a_i denotes an abort by transaction T_i . Which one of the following statements about the above schedule is **TRUE**?

- (a) S is non-recoverable
- (b) S is recoverable, but has a cascading abort
- (c) S does not have a cascading abort
- (d) S is strict

S	
T_1	T_2
$R(X)$	$R(x)$
	$R(Y)$
$W(X)$	
$R(Y)$	
	$W(X)$
a_1	
	a_2

CONCURRENCY CONTROL

- Now we understood that if there is a schedule how to check whether it will work correctly or not i.e. whether it will maintain the consistency of the data base or not. (conflict serializability, view serializability, recoverability and cascade less)
- Now we will understand those protocol which guarantee how to design those schedules which ensure conflict serializability or other properties. There are different approach or idea to ensure conflict serializability which is the most important property.
- So first we must understand what is the possibility of conflict between two instruction and if somehow, we manage then the generated schedule will always be conflict serializable

Goals of a Protocol: - We desire the following properties from schedule generating protocols

- Concurrency should be as high as possible, as this is our ultimate goal because of which we are making all the effort.
- The time taken by a transaction should also be less.
- Desirable Properties satisfied by the protocol
- Easy to understand and implement

<http://www.knowledgegate.in/gate>

TIME STAMP ORDERING PROTOCOL

- Basic idea of time stamping is to decide the order between the transaction before they enter in the system using a stamp (time stamp), in case of any conflict during the execution order can be decided using the time stamp.
- Let's understand how this protocol works, here we have two idea of timestamping, one for the transaction, and other for the data item.

- Time stamp for transaction,
 - With each transaction t_i , in the system, we associate a unique fixed timestamp, denoted by $TS(t_i)$.
 - This timestamp is assigned by database system to a transaction at time transaction enters into the system.
 - If a transaction has been assigned a timestamp $TS(t_i)$ and a new transaction t_j , enters into the system with a timestamp $TS(t_j)$, then always $TS(t_i) < TS(t_j)$.

- Two things are to be noted
 1. First time stamp of a transaction remain fixed throughout the execution
 2. Second it is unique means no two transaction can have the same timestamp.
- The reason why we called time stamp not stamp, because for stamping we use the value of the system clock as stamp, advantage is,
 - T_t will always be unique as time never repeats
 - There is no requirement of refreshing and starting with fresh value.
- The time stamp of the transaction also determines the serializability order.
- Thus if $TS(t_i) < TS(t_j)$, then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction t_i appears before transaction t_j .

- Time stamp with data item, in order to assure such scheme, the protocol maintains for each data item Q two timestamp values:
 1. W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q) successfully.
 2. R-timestamp(Q) is the largest time-stamp of any transaction that executed read(Q) successfully.
- These timestamps are updated whenever a new read(Q) or write(Q) instruction is executed.

- Suppose a transaction T_i request a *read(Q)*
 1. If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and T_i is rolled back.
 2. If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the read operation is executed, and $R\text{-timestamp}(Q)$ is set to the maximum of $R\text{-timestamp}(Q)$ and $TS(T_i)$.

- Suppose that transaction T_i issues **write(Q)**.
 1. If $TS(T_i) < R\text{-timestamp}(Q)$, then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. Hence, the write operation is rejected, and T_i is rolled back.
 2. If $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of Q. Hence, this write operation is rejected, and T_i is rolled back.
 3. If $TS(T_i) \geq R\text{-timestamp}(Q)$, then the write operation is executed, and $W\text{-timestamp}(Q)$ is set to $\max(W\text{-timestamp}(Q), TS(T_i))$.
 4. If $TS(T_i) \geq W\text{-timestamp}(Q)$, then the write operation is executed, and $W\text{-timestamp}(Q)$ is set to $\max(W\text{-timestamp}(Q), TS(T_i))$.

If a transaction T_i is rolled back by the concurrency control scheme as a result of either a read or write operation, the system assigns it's a new timestamp and restarts it.

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering					
Thomas Write Rule					
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

<http://www.knowledgegate.in/gate>

THOMAS WRITE RULE

- Thomas write is an improvement in time stamping protocol, which makes some modification and may generate those protocols that are even view serializable, because it allows greater potential concurrency.
- It is a Modified version of the timestamp-ordering protocol in which Blind write operations may be ignored under certain circumstances.
- The protocol rules for read operations remain unchanged. while for write operation, there is slightly change in Thomas write rule than timestamp ordering protocol.

When T_i attempts to write data item Q ,

- if $TS(T_i) < W\text{-timestamp}(Q)$, then T_i is attempting to write an obsolete value of $\{Q\}$. Rather than rolling back T_i as the timestamp ordering protocol would have done, this {write} operation can be ignored.
- This modification is valid as the any transaction with $TS(T_i) < W\text{-timestamp}(Q)$, the value written by this transaction will never be read by any other transaction performing $Read(Q)$ ignoring such obsolete write operation is considerable.
- Thomas' Write Rule allows greater potential concurrency. Allows some view-serializable schedules that are not conflict serializable.

T_3	T_4	T_6
read(Q)	write(Q)	write(Q)
write(Q)		

<http://www.knowledgegate.in/gate>

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule					
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

<http://www.knowledgegate.in/gate>

Lock Based Protocols

- To ensure isolation is to require that data items be accessed in a mutually exclusive manner i.e. while one transaction is accessing a data item, no other transaction can modify that data item. Locking is the most fundamental approach to ensure this.
- Lock based protocols ensure this requirement. Idea is first obtain a lock on the desired data item then if lock is granted then perform the operation and then unlock it.

- In general, we support two modes of lock because, to provide better concurrency.
- **Shared mode**
 - If transaction T_i has obtained a shared-mode lock (denoted by S) on any data item Q, then T_i can read, but cannot write Q, any other transaction can also acquire a shared mode lock on the same data item (this is the reason we called this shared mode).
- **Exclusive mode**
 - If transaction T_i has obtained an exclusive-mode lock (denoted by X) on any data item Q, then T_i can both read and write Q, any other transaction cannot acquire either a shared or exclusive mode lock on the same data item. (this is the reason we called this exclusive mode)

Lock –Compatibility Matrix

- Conclusion shared is compatible only with shared while exclusive is not compatible either with shared or exclusive.
- To access a data item, transaction T_i must first lock that item, if the data item is already locked by another transaction in an incompatible mode, or some other transaction is already waiting in non-compatible mode, then concurrency control manager will not grant the lock until all incompatible locks held by other transactions have been released. The lock is then granted.

Current State of lock of data items				
Requested Lock		Exclusive	Shared	Unlocked
	Exclusive	N	N	Y
	Shared	N	Y	Y
	Unlock	Y	Y	-

- Lock based protocol do not ensure serializability as granting and releasing of lock do not follow any order and any transaction any time may go for lock and unlock. Here in the example below we can see, that even this transaction in using locking but neither it is conflict serializable nor independent from deadlock.

T ₁	T ₂
LOCK-X(A)	
READ(A)	
WRITE(A)	
UNLOCK(A)	
	LOCK-S(B)
	READ(B)
	UNLOCK(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
UNLOCK(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(A)

Two phase locking protocol(2PL)

- The protocol ensures that each transaction issue lock and unlock requests in two phases, note that each transaction will be 2 phased not schedule.
- Growing phase- A transaction may obtain locks, but not release any locks.
- Shrinking phase- A transaction may release locks, but may not obtain any new locks.
- Initially a transaction is in growing phase and acquires lock as needed and in between can perform operation reach to lock point and once a transaction releases a lock, it can issue no more lock requests i.e. it enters the shrinking phase.

T_1	T_2
LOCK-X(A)	
READ(A)	
WRITE(A)	
	LOCK-S(B)
	READ(B)
LOCK-X(B)	
READ(B)	
WRITE(B)	
	LOCK-S(A)
	READ(A)
	UNLOCK(B)
UNLOCK(A)	
UNLOCK(B)	
	UNLOCK(A)

<http://www.knowledgegate.in/gate>

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL					
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

<http://www.knowledgegate.in/gate>

Q Which of the following concurrency protocol ensures both conflict serializability and freedom from deadlock?

(1) 2 - phase Locking

(2) Time stamp - ordering

(a) Both (1) and (2)

(b) (1) only

(c) (2) only

(d) Neither (1) nor (2)

Conservative 2PL

- The idea is there is no growing phase transaction start directly from lock point, i.e. transaction must first acquire all the required locks then only it can start execution. If all the locks are not available then transaction must release the acquired locks and must wait.
 - Shrinking phase will work as usual, and transaction can unlock any data item anytime.
 - we must have a knowledge in future to understand what is data required so that we can use it

- Q** In conservative two phase locking protocol, a transaction
- a)** Should release exclusive locks only after the commit operation
 - b)** Should release all the locks only at beginning of the transaction
 - c)** should acquire all the locks at beginning of the transaction
 - d)** Should acquire all the exclusive locks at beginning transaction

<http://www.knowledgegate.in/gate>

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL					
Rigorous 2PL					
Strict 2PL					

<http://www.knowledgegate.in/gate>

RIGOROUS 2PL

- Requires that all locks be held until the transaction commits.
- This protocol requires that locking be two phase and also all the locks taken be held by transaction until that transaction commit.
- Hence there is no shrinking phase in the system.

<http://www.knowledgegate.in/gate>

Q In a Rigorous 2 phase protocol

- a)** All shared locks held by the transaction are released after the transaction is committed
- b)** All exclusive locks held by the transaction are released after the transaction is committed
- c)** All locks held by the transaction are released after the transaction is committed
- d)** All locks held by the transaction are released before the transaction is committed

<http://www.knowledgegate.in/gate>

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL					
Strict 2PL					

<http://www.knowledgegate.in/gate>

STRICT 2PL

- that all exclusive-mode locks taken by a transaction be held until that transaction commits. This requirement ensures that any data written by an uncommitted transaction are locked in exclusive mode until the transaction commits, preventing any other transaction from reading the data.
- This protocol requires that locking be two phase and also that exclusive –mode locks taken by transaction be held until that transaction commits.
- So it is simplified form of rigorous 2pl

	Conflict Serializability	View Serializability	Recoverability	Cascadelessness	Deadlock Freedom
Time Stamp Ordering	YES	YES	NO	NO	YES
Thomas Write Rule	NO	YES	NO	NO	YES
Basic 2PL	YES	YES	NO	NO	NO
Conservative 2PL	YES	YES	NO	NO	YES
Rigorous 2PL	YES	YES	YES	YES	NO
Strict 2PL					

<http://www.knowledgegate.in/gate>

Q Consider the following two statements about database transaction schedules:

I. Strict two-phase locking protocol generates conflict serializable schedules that are also recoverable.

II. Timestamp-ordering concurrency control protocol with Thomas Write Rule can generate view serializable schedules that are not conflict serializable.

Which of the above statements is/are TRUE?

(a) Both I and II

(b) Neither I nor II

(c) II only

(d) I only

<http://www.knowledgegate.in/gate>

Q Which of the following statement is/are correct

- a)** Every conflict serializable schedule allowed under 2PL protocol is allowed by basic time stamping protocol.
- b)** Every schedule allowed under basic time stamping protocol is allowed by Thomas-write rule
- c)** Every schedule allowed under Thomas-write rule is allowed by basic time stamping protocol
- d)** none

<http://www.knowledgegate.in/gate>

Video chapters

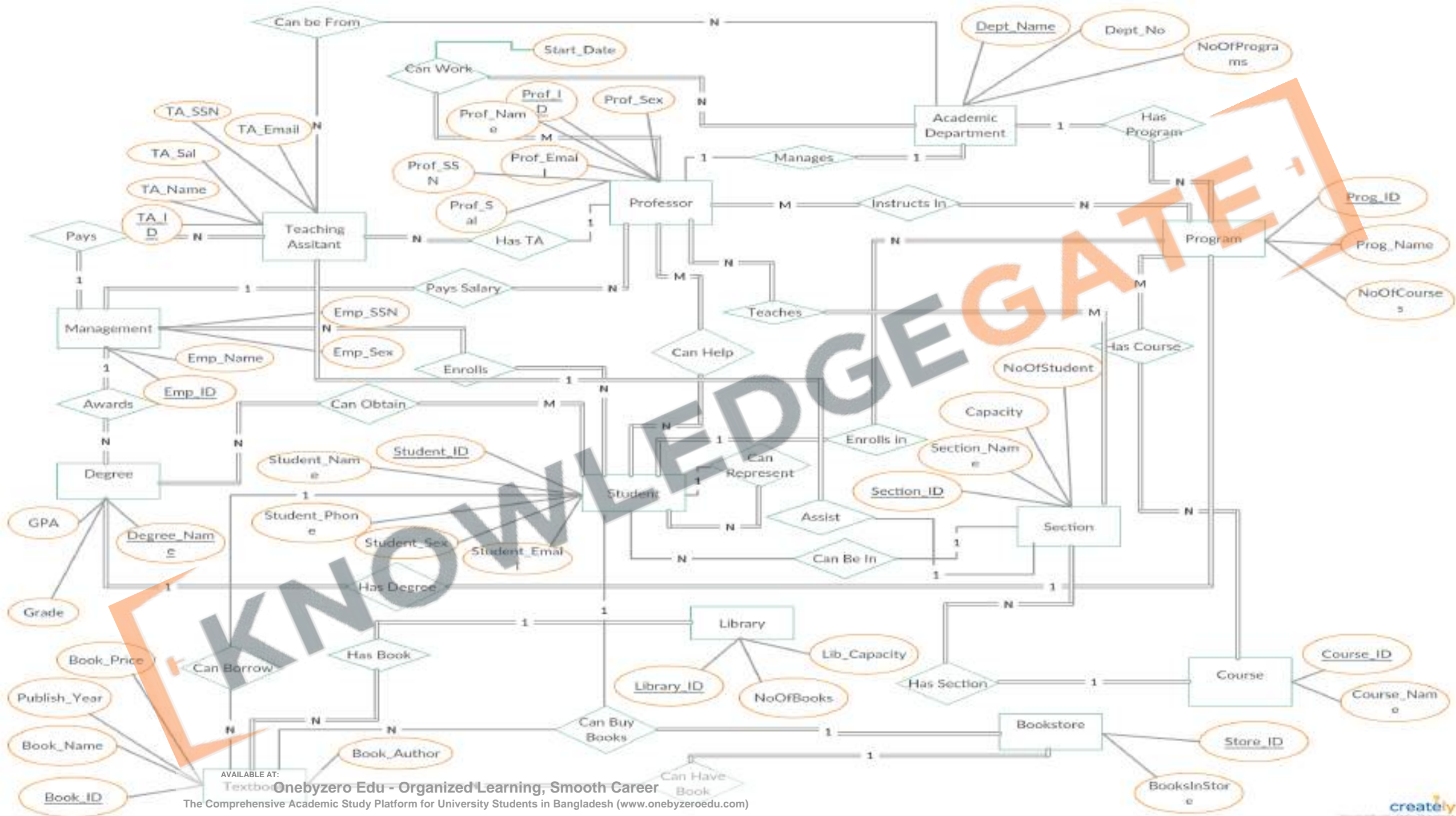
- Chapter-1 (Transactions and concurrency control)
- Chapter-2 (ER-model)
- Chapter-3 (Relational model, Functional Dependencies, Keys)
- Chapter-4 (Normalization)
- Chapter-5 (File organization, indexing (e.g., B and B+ trees))
- Chapter-6 (Relational algebra, tuple calculus, SQL)

<http://www.knowledgegate.in/gate>

E-R DIAGRAM/MODEL

- Introduced in 1976 by Dr Peter Chen, a non-technical design method works on conceptual level based on the perception of the real world.
- E-R data model was developed to facilitate database design by allowing specification of an enterprise schema that represents ***the overall logical structure of a database.***





Conclusion

- It consists of collections of basic objects, called entities and of relationships among these entities and attributes which defines their properties.
- E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema.
- It is free from ambiguities and provides a standard and logical way of visualizing the data.
- As basically it is a diagrammatical representation easy to understand even by a non-technical user.

<http://www.knowledgegate.in/gate>



<http://www.knowledgegate.in/gate>

ENTITY

- An entity is a thing or an object in the real world that is distinguishable from other object based on the values of the attributes it possesses.
- An entity may be **concrete**, such as a person or a book, or it may be **abstract**, such as a course, a course offering, or a flight reservation.

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250

- *Types of Entity*
 - **Tangible** - *Entities which physically exist in real world. E.g. - Car, Pen, locker*
 - **Intangible** - *Entities which exist logically. E.g. – Account, video.*



- **ENTITY SET**- Collection of same type of entities that share the same properties or attributes. In an ER diagram an entity set is represented by a rectangle. In a relational model it is represented by a separate table.

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250



- In ER diagram we cannot represent an entity, as entity is an instant not schema, and ER diagram is designed to understand schema.
- In a relational model entity is represented by a row or a tuple or a record in a table.

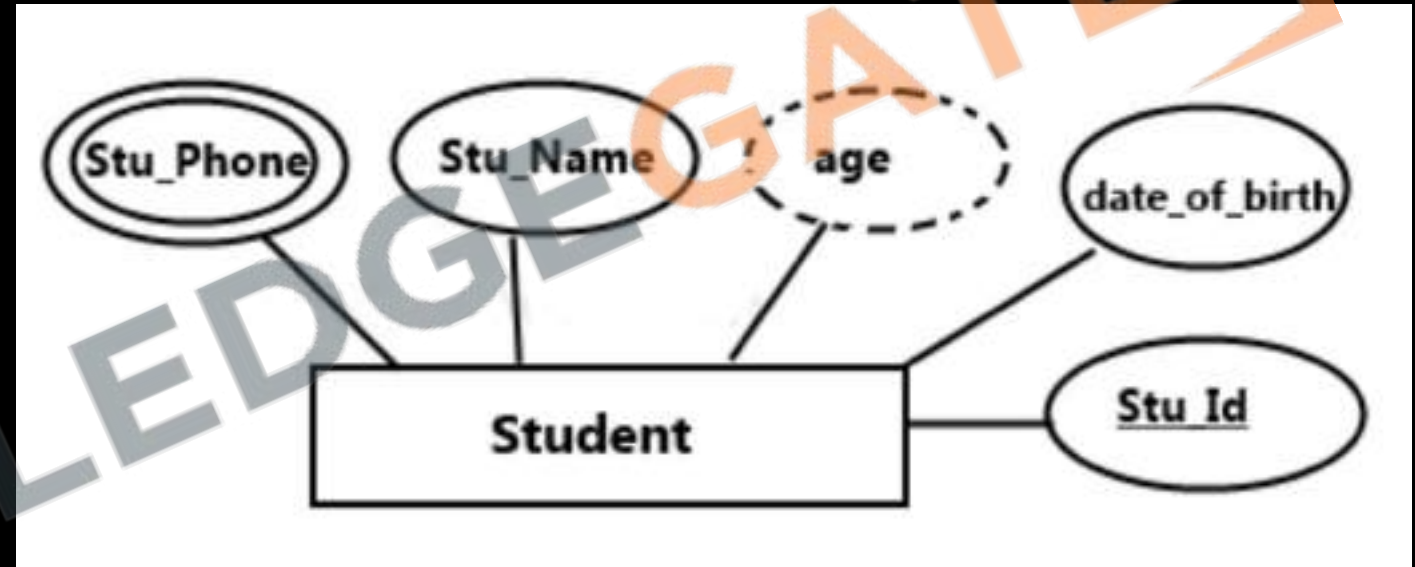
ATTRIBUTES

- Attributes are the units defines and describe properties and characteristics of entities. Attributes are the descriptive properties possessed by each member of an entity set. for each attribute there is a set of permitted values called domain.

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250

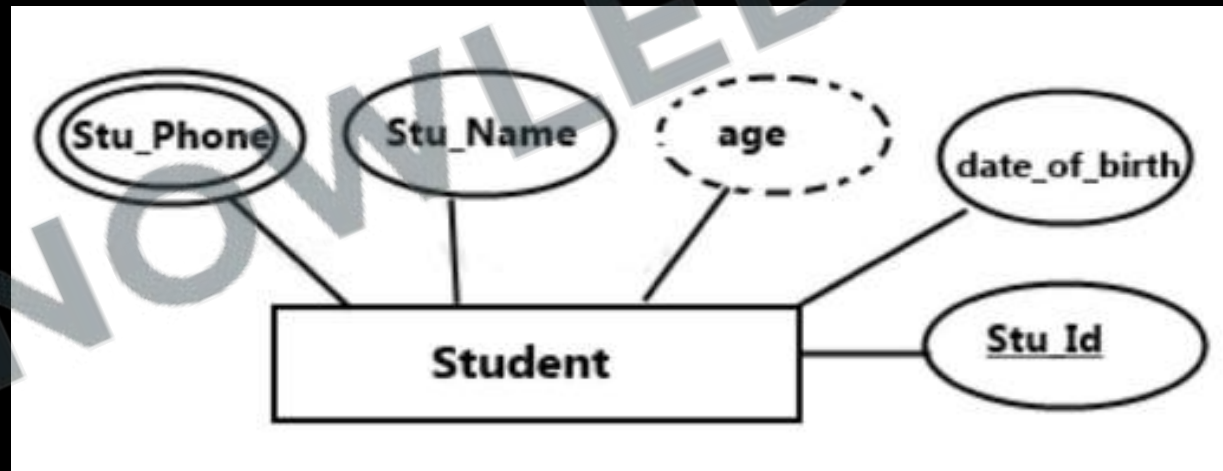
- In an ER diagram attributes are represented by ellipse or oval connected to rectangle.
- While in a relational model they are represented by independent column. e.g. Instructor (ID, name, salary, dept_name)

Name	FName	City	Age	Salary
Smith	John	3	35	\$280
Doe	Jane	1	28	\$325
Brown	Scott	3	41	\$265
Howard	Shemp	4	48	\$359
Taylor	Tom	2	22	\$250



Types of Attributes

- **Single valued**- Attributes having single value at any instance of time for an entity. E.g. – Aadhar no, dob.
- **Multivalued** - Attributes which can have more than one value for an entity at same time. E.g.
 - Phone no, email, address.
 - A multivalued attribute is represented by a double ellipse in an ER diagram and by an independent table in a relational model.
 - Separate table for each multivalued attribute, by taking mva and pk of main table as fk in new table



Customer			
Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

<http://www.knowledgegate.in/gate>

Customer			
Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

जुगाड़ technology



Customer				
Customer ID	First Name	Surname	Telephone Number1	Telephone Number2
123	Pooja	Singh	555-861-2025	192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53	182-929-2929
789	John	Doe	555-808-9633	

<http://www.knowledgegate.in/gate>

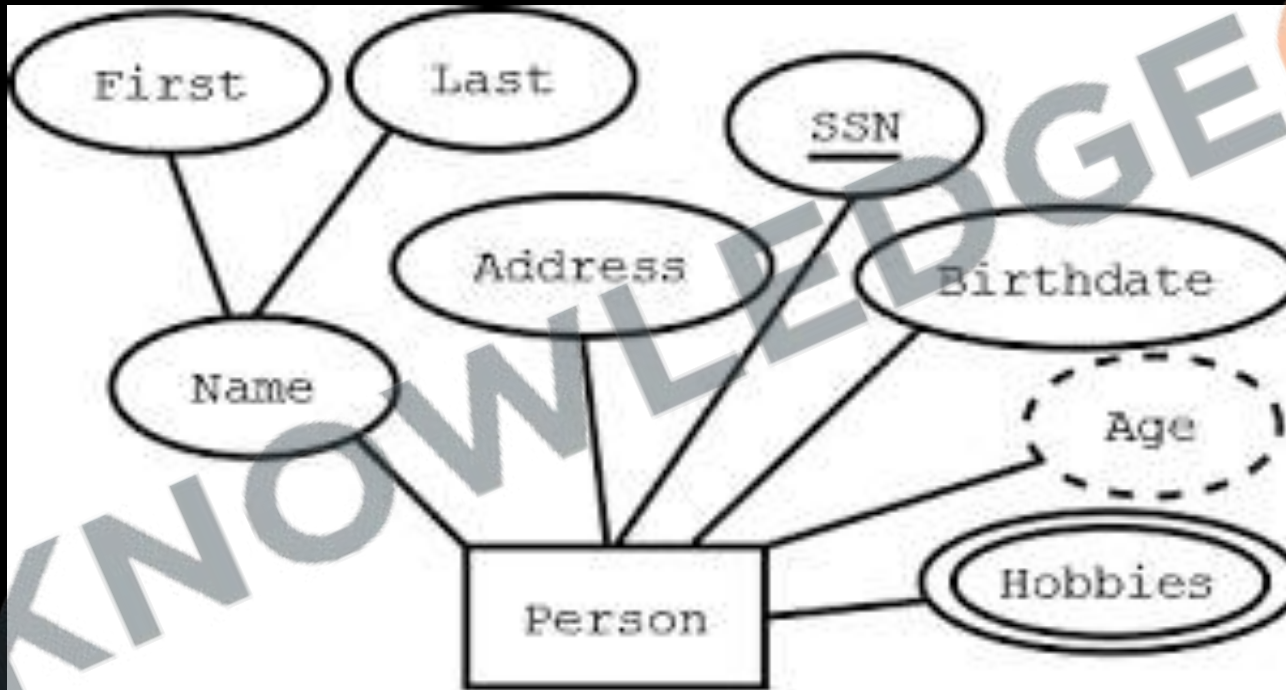
Customer			
Customer ID	First Name	Surname	Telephone Number
123	Pooja	Singh	555-861-2025, 192-122-1111
456	San	Zhang	(555) 403-1659 Ext. 53; 182-929-2929
789	John	Doe	555-808-9633

Customer Name		
Customer ID	First Name	Surname
123	Pooja	Singh
456	San	Zhang
789	John	Doe

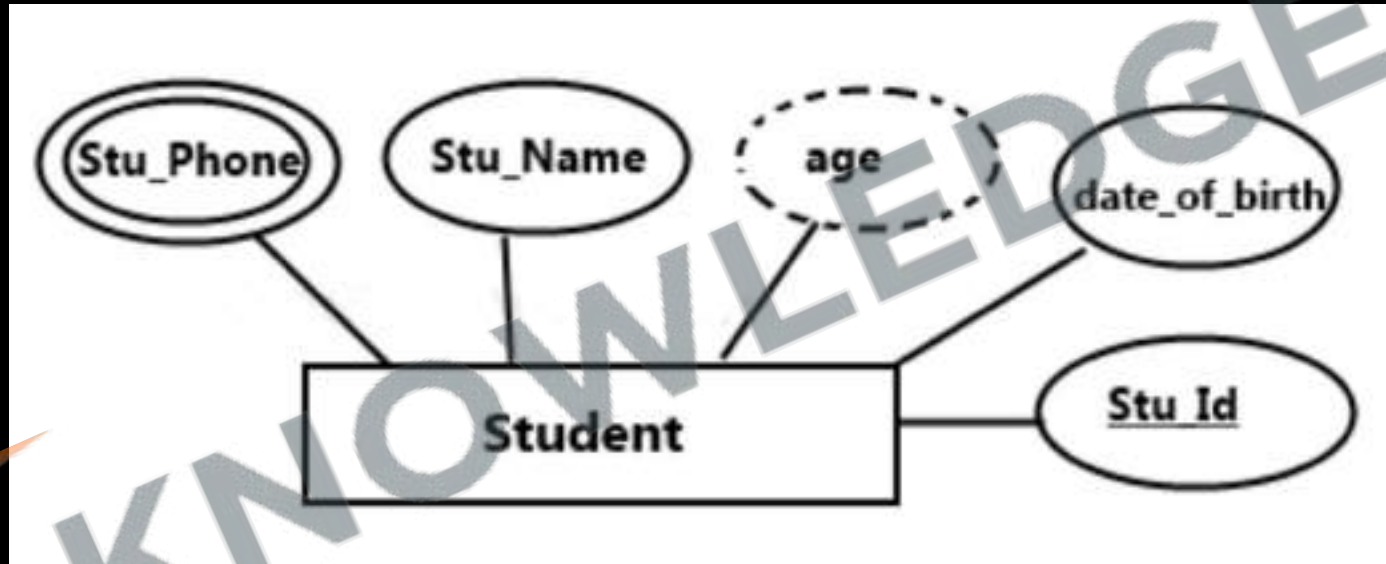
Customer Phone Number	
Customer ID	Telephone Number
123	555-861-2025
123	192-122-1111
456	(555) 403-1659 Ext. 53
456	182-929-2929
789	555-808-9633

<http://www.knowledgegate.in/gate>

- **Simple** - Attributes which cannot be divided further into sub parts. E.g. Age
- **Composite** - Attributes which can be further divided into sub parts, as simple attributes. A composite attribute is represented by an ellipse connected to an ellipse and in a relational model by a separate column.

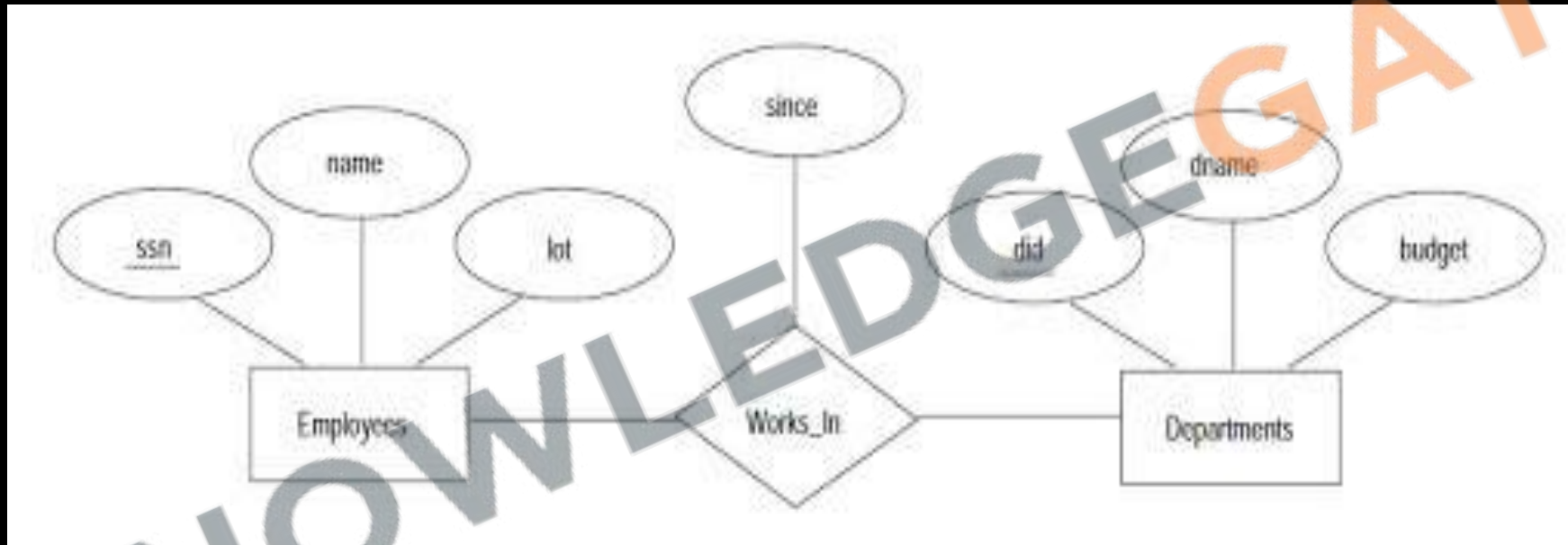


- **Stored** - Main attributes whose value is permanently stored in database. E.g. date_of_birth
- **Derived** -The value of these types of attributes can be derived from values of other Attributes. E.g. - Age attribute can be derived from date_of_birth and Date attribute.



Descriptive attribute - Attribute of relationship is called descriptive attribute.

- An attribute takes a null value when an entity does not have a value for it. The null value may indicate “not applicable” — that is, that the value does not exist for the entity.



- Null can also designate that an attribute value is **unknown**. An unknown value may be either missing (the value does exist, but we do not have that information) or not known (we do not know whether or not the value actually exists).

<http://www.knowledgegate.in/gate>

Relationship / Association

Relationship Status: ✓

Family:

Featured Friends:

Select Relation: ▾ ×

Single

In a relationship

Engaged

Married

It's complicated

In an open relationship

Widowed

Separated

Divorced

In a civil union

In a domestic partnership

Create new list - Add an existing list or group

Save Changes

Relationship / Association

- Is an association between two or more entities of same or different entity set.
- In ER diagram we cannot represent individual relationship as it is an instance or data.



- In an ER diagram it is represented by a diamond, while in relational model sometimes through foreign key and other time by a separate table.



- Note: - normally people use word relationship for relationship type so don't get confused.

<http://www.knowledgegate.in/gate>

- Every relationship type has three components.
 - Name- Every relation must have a unique name.
 - Degree-
 - Structural constraints (cardinalities ratios, participation)

<http://www.knowledgegate.in/gate>

Degree of a relationship/Relationship Set

- Means number of entities set(relations/tables) associated(participate) in the relationship set.
- Most of the relationship sets in a data base system are binary.
- Occasionally however relationship sets involve more than two entity sets.
- Logically, we can associate any number of entity set in a relationship called N-ary Relationship.



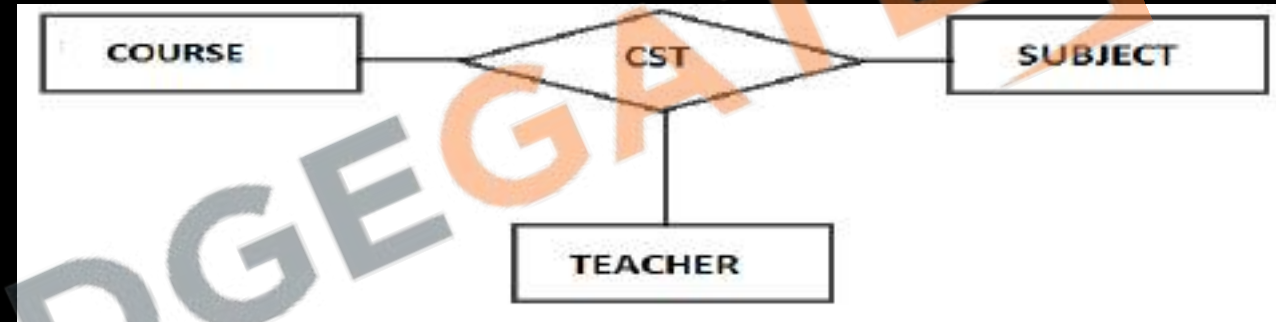
- **Unary Relationship** - One single entity set participate in a Relationship, means two entities of the same entity set are related to each other.
- These are also called as self -referential Relationship set.
- E.g.- A member in a team maybe supervisor of another member in team.



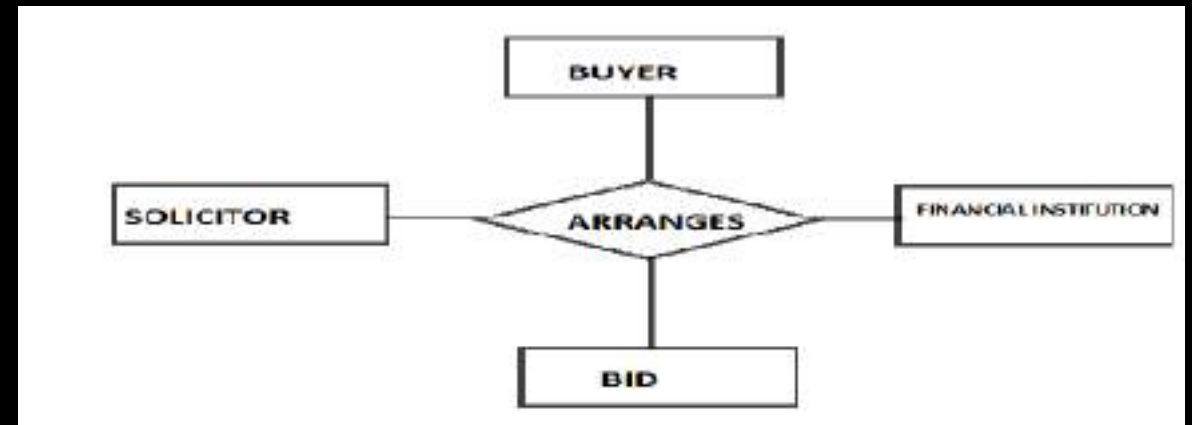
- **Binary Relationship** - Two entity sets participate in a Relationship. It is most common Relationship.



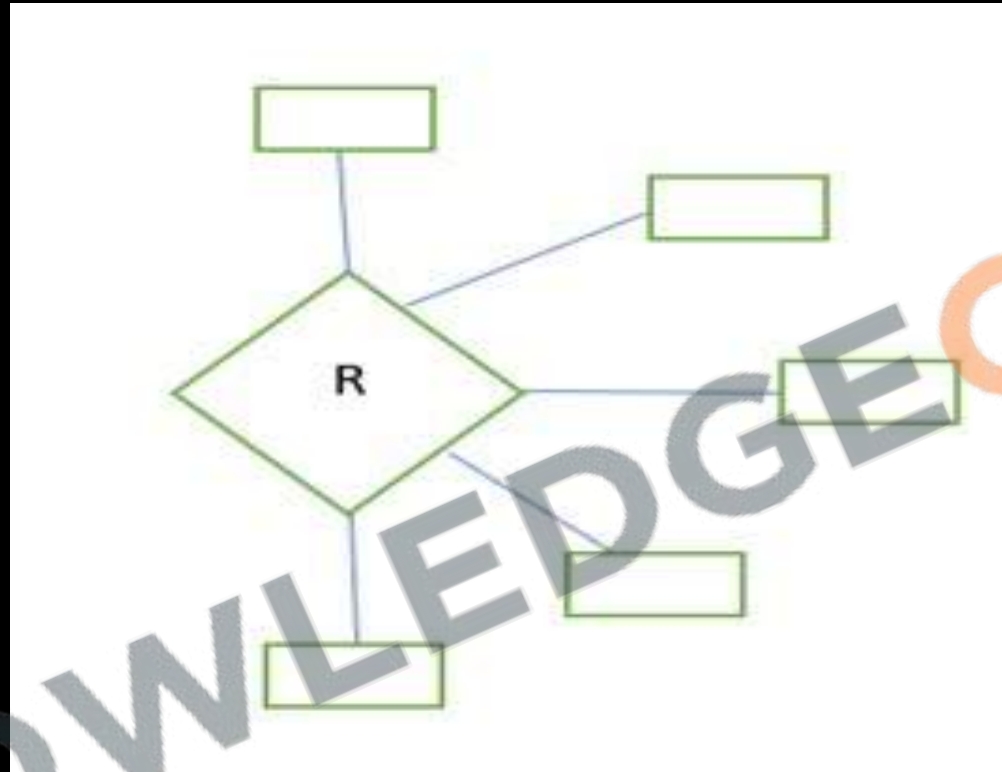
- **Ternary Relationship** - When three entities participate in a Relationship. E.g. The University might need to record which teachers taught which subjects in which courses.



- **Quaternary Relationship** - When four entities participate in a Relationship.



- **N-ary relationship** – where n number of entity set are associated



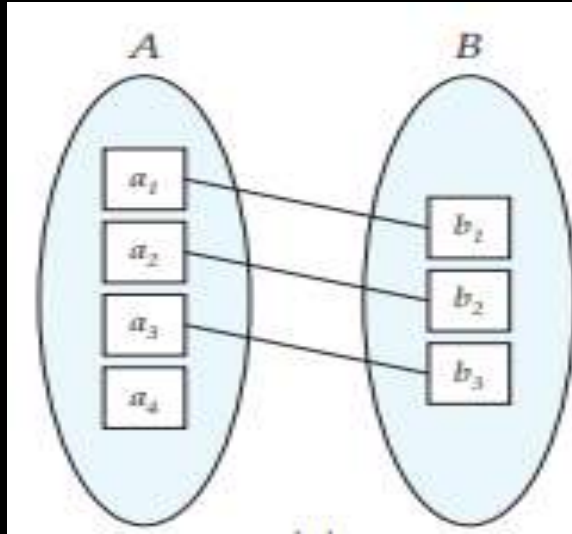
- But the most common relationships in ER models are **Binary**.

<http://www.knowledgegate.in/gate>

Structural constraints (Cardinalities Ratios, Participation)

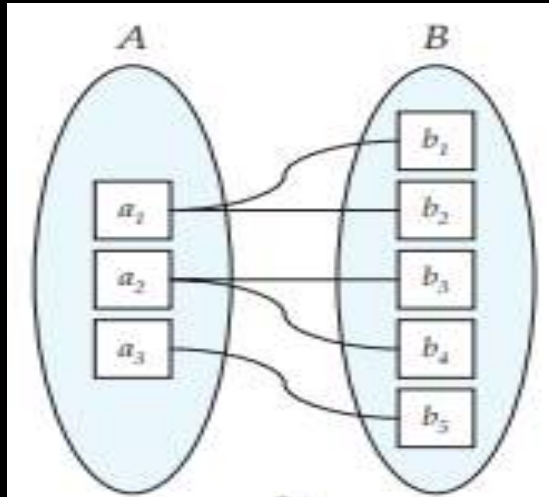
- An E-R enterprise schema may define certain constraints to which the contents of a database must conform.
- Express the number of entities to which another entity can be associated via a relationship set. Four possible categories are-
 - One to One (1:1) Relationship.
 - One to Many (1: M) Relationship.
 - Many to One (M: 1) Relationship.
 - Many to Many (M: N) Relationship.

One to One (1:1) Relationship - An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.



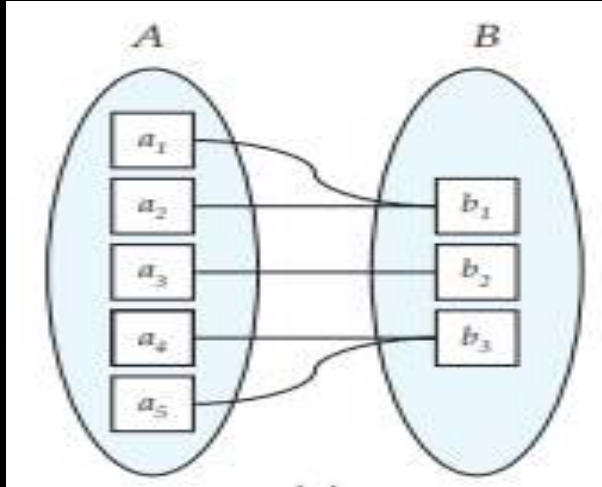
E.g.- The directed line from relationship set advisor to both entities set indicates that 'an instructor may advise at most one student, and a student may have at most one advisor'.

One to Many (1: M) Relationship - An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.



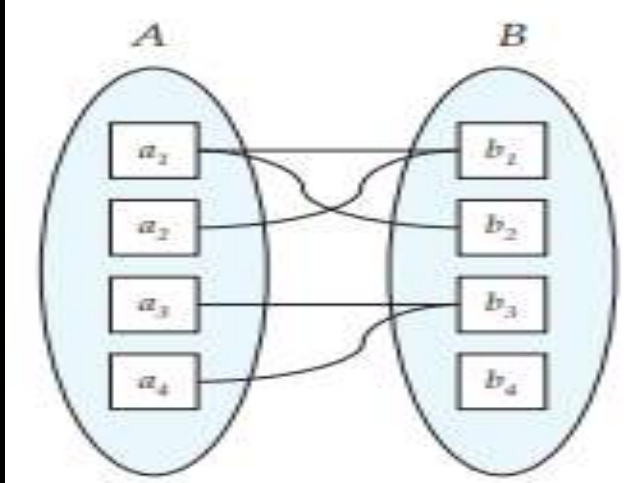
E.g.- This indicates that an instructor may advise many students, but a student may have at most one advisor.

Many to One (M: 1) Relationship - An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.



E.g.- This indicates that student may have many instructors but an instructor can advise at most one student.

Many to Many(M:N) Relationship - An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



E.g.- This indicates a student may have many advisors and an instructor may advise many students.

Participation Constraints

- Participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type.
- These constraints specify the minimum and maximum number of relationship instances that each entity must/can participate in.
- **Max cardinality** – it defines the maximum no of times an entity occurrence participating in a relationship.
- **Min cardinality** - it defines the minimum no of times an entity occurrence participating in a relationship.



- **PARTICIPATION CONSTRAINTS**- it defines participations of entities of an entity type in a relationship.
- **PARTIAL PARTICIPATION (min cardinality zero)** - In Partial participation only some entities of entity set participate in Relationship set, that is there exists at least one entity which do not participate in a relation.
- **TOTAL PARTICIPATION (min cardinality at least one)** - In total participation every entity of an entity set participates in at least one relationship in Relationship set.

- **Conversion of 1-1 relationship(binary)**
 - No separate table is required, take pk of one side as pk on other side, priority must be given to the side having total participation.
- **Conversion of 1-n or n-1 relationship (binary)**
 - No separate table is required, modify n side by taking pk of 1 side a foreign key on n side.
- **Conversion of n-n relationship (binary)**
 - Separate table is required take pk of both table and declare their combination as a pk of new table.

Q In an Entity-Relationship (ER) model, suppose R is a many-to-one relationship from entity set E_1 to entity set E_2 . Assume that E_1 and E_2 participate totally in R and that the cardinality of E_1 is greater than the cardinality of E_2 . Which one of the following is true about R ?

- (a)** Every entity in E_1 is associated with exactly one entity in E_2
- (b)** Some entity in E_1 is associated with more than one entity in E_2
- (c)** Every entity in E_2 is associated with exactly one entity in E_1
- (d)** Every entity in E_2 is associated with at most one entity in E_1

Q Given the basic ER and relational models, which of the following is INCORRECT?

- (A)** An attribute of an entity can have more than one value
- (B)** An attribute of an entity can be composite
- (C)** In a row of a relational table, an attribute can have more than one value
- (D)** In a row of a relational table, an attribute can have exactly one value or a NULL value

<http://www.knowledgegate.in/gate>

Q Consider the entities 'hotel room', and 'person' with a many to many relationship 'lodging' as shown below:

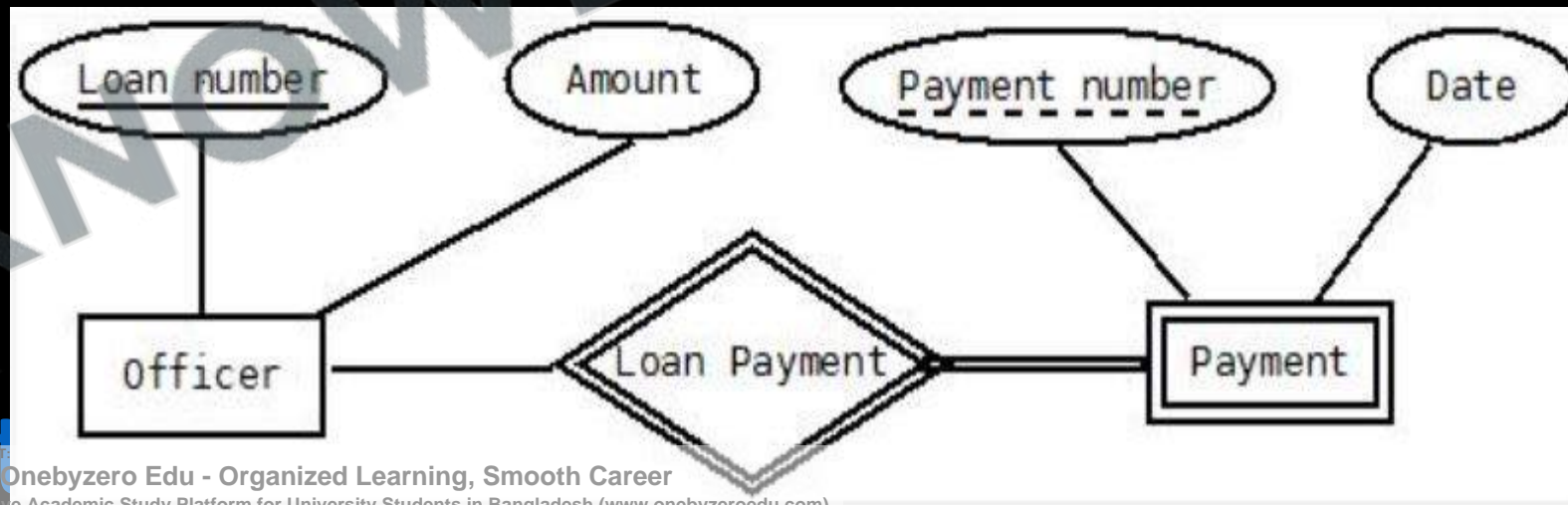


If we wish to store information about the rent payment to be made by person (s) occupying different hotel rooms, then this information should appear as an attribute of

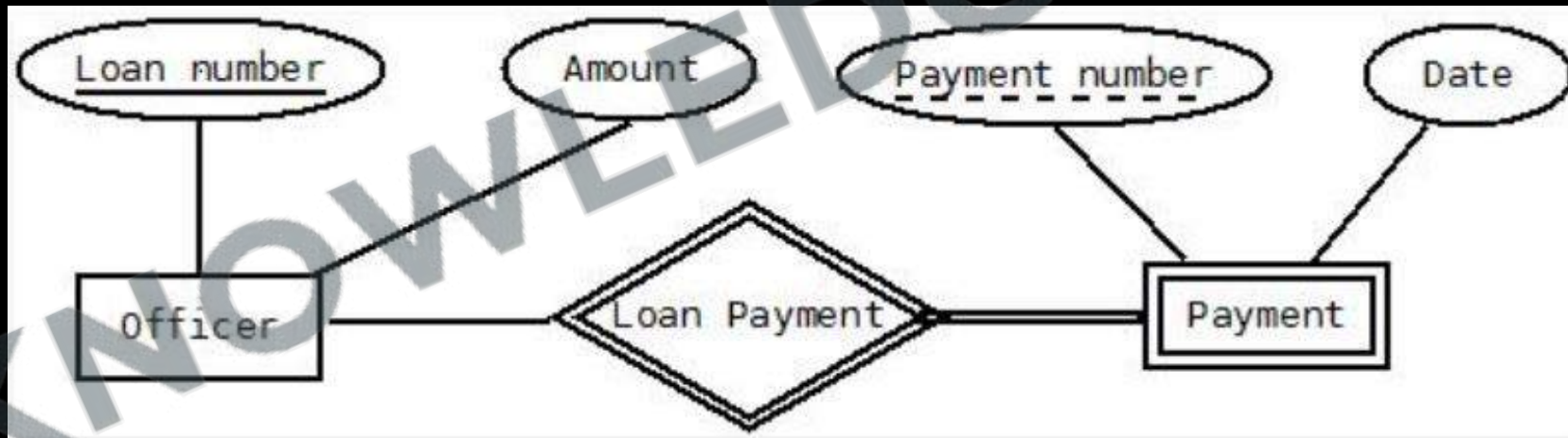
- (A) Person
- (B) Hotel Room
- (C) Lodging
- (D) None of these

STRONG AND WEAK ENTITY SET

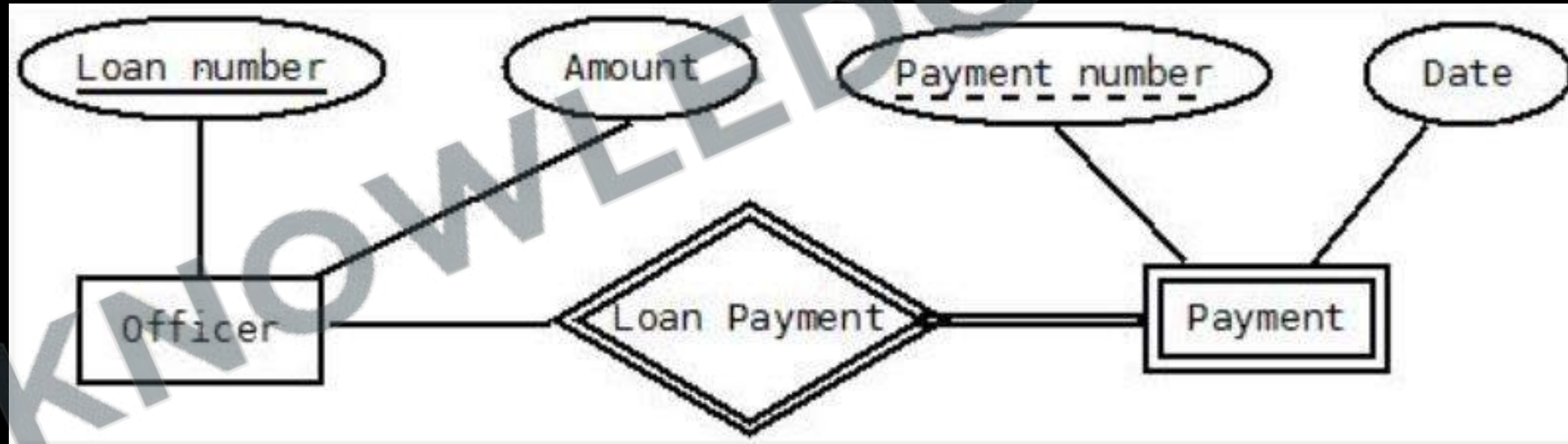
- An entity set is called strong entity set, if it has a primary key, all the tuples in the set are distinguishable by that key.
- An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set.
- It contains discriminator attributes (partial key) which contain partial information about the entity set, but it is not sufficient enough to identify each tuple uniquely. Represented by double rectangle.



- For a weak entity set to be meaningful and converted into strong entity set, it must be associated with another strong entity set called the **identifying or owner entity set** i.e. weak entity set is said to be **existence dependent** on the identity set.
- The identifying entity set is said to own weak entity set that it identifies.
- A weak entity set may participate as owner in an identifying relationship with another weak entity set.



- The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship (double diamonds)**.
- The identifying relationship is many to one from the weak entity set to identifying entity set, and the participation of the weak entity set in relationship is always total.
- The primary key of weak entity set will be the union of primary key and discriminator attributes.



REASONS TO HAVE WEAK ENTITY SET

- Weak entities reflect the logical structure of an entity being dependent on another.
- Weak entity can be deleted automatically when their strong entity is deleted.
- Without weak entity set it will lead to duplication and consequent possible inconsistencies.

Conversion From ER Diagram To Relational Model

- Entity Set
 - Convert every strong entity set into a separate table.
 - Convert every weak entity set into a separate table, by making it dependent into one strong entity set (**identifying or owner entity set**).

Conversion From ER Diagram To Relational Model

- Relationship(Unary)
 - No separate table is required, add a new column as fk which refer the pk of the same table.

Conversion From ER Diagram To Relational Model

- Relationship(Binary)
 - 1:1 No separate table is required, take pk of one side and put it as fk on other side, priority must be given to the side having total participation.

<http://www.knowledgegate.in/gate>

Conversion From ER Diagram To Relational Model

- Relationship(Binary)
 - 1:n or n:1 No separate table is required, modify n side by taking pk of 1 side as foreign key on n side

<http://www.knowledgegate.in/gate>

Roll_no	name	Age
1	A	19
2	B	18
3	C	20
4	D	20

Edu_id	name	Subject
1	A	OS
2	B	DBMS
3	C	TOC
4	D	CN

<http://www.knowledgegate.in/gate>

Conversion From ER Diagram To Relational Model

- Relationship(Binary)
 - (n-n) Separate table is required take pk of both table and declare their combination as a pk of new table

<http://www.knowledgegate.in/gate>

Roll no	name	Age
1	A	19
2	B	18
3	C	20
4	D	20

Edu_id	name	Subject
1	A	OS
2	B	DBMS
3	C	TOC
4	D	CN

Roll no	name	Age	Edu_id
1	A	19	1
1	A	19	3
1	A	19	4
2	B	18	2
2	B	18	3
3	C	20	2
4	D	20	3
4	D	20	4

Edu_id	name	Subject	Roll no
1	A	OS	1
2	B	DBMS	2
2	B	DBMS	3
3	C	TOC	1
3	C	TOC	2
3	C	TOC	4
4	D	CN	1
4	D	CN	4

Roll no	Edu_id
1	1
1	3
1	4
2	2
2	3
3	2
4	3
4	4

Roll no	name	Age
1	A	19
2	B	18
3	C	20
4	D	20

Edu_id	name	Subject
1	A	OS
2	B	DBMS
3	C	TOC
4	D	CN

Roll no	Edu_id
1	1
1	3
1	4
2	2
2	3
3	2
4	3
4	4

<http://www.knowledgegate.in/gate>

Conversion From ER Diagram To Relational Model

- Relationship(3 or More)
 - Take the pk of all participating entity sets as fk and declare their combinations as pk in the new table.

<http://www.knowledgegate.in/gate>

Conversion From ER Diagram To Relational Model

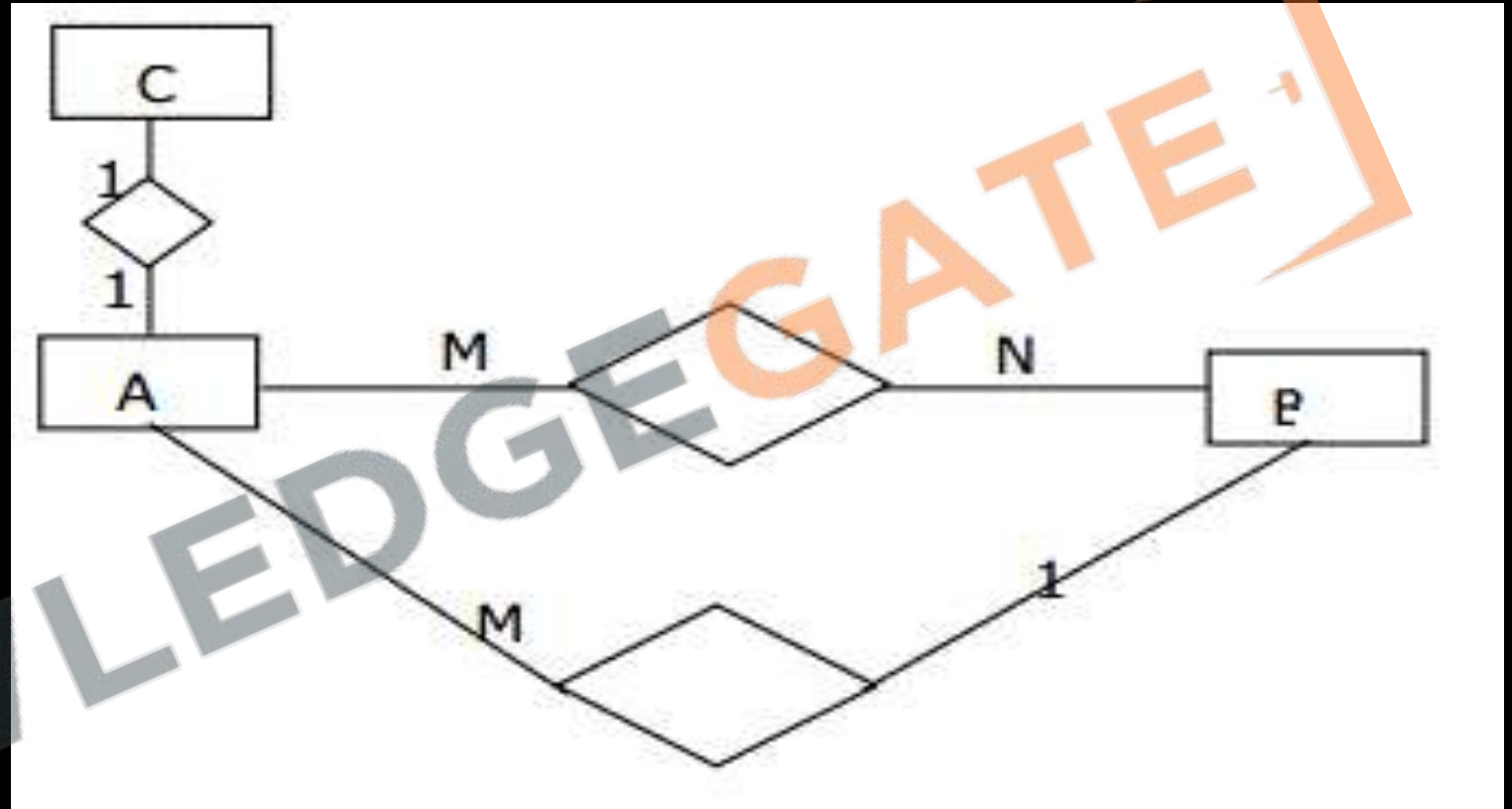
- Multivalued Attributes
 - A separate table must be taken for all multivalued attributes, where we take pk of the main table as fk and declare combination of fk and multivalued attribute are pk in the new table.

Conversion From ER Diagram To Relational Model

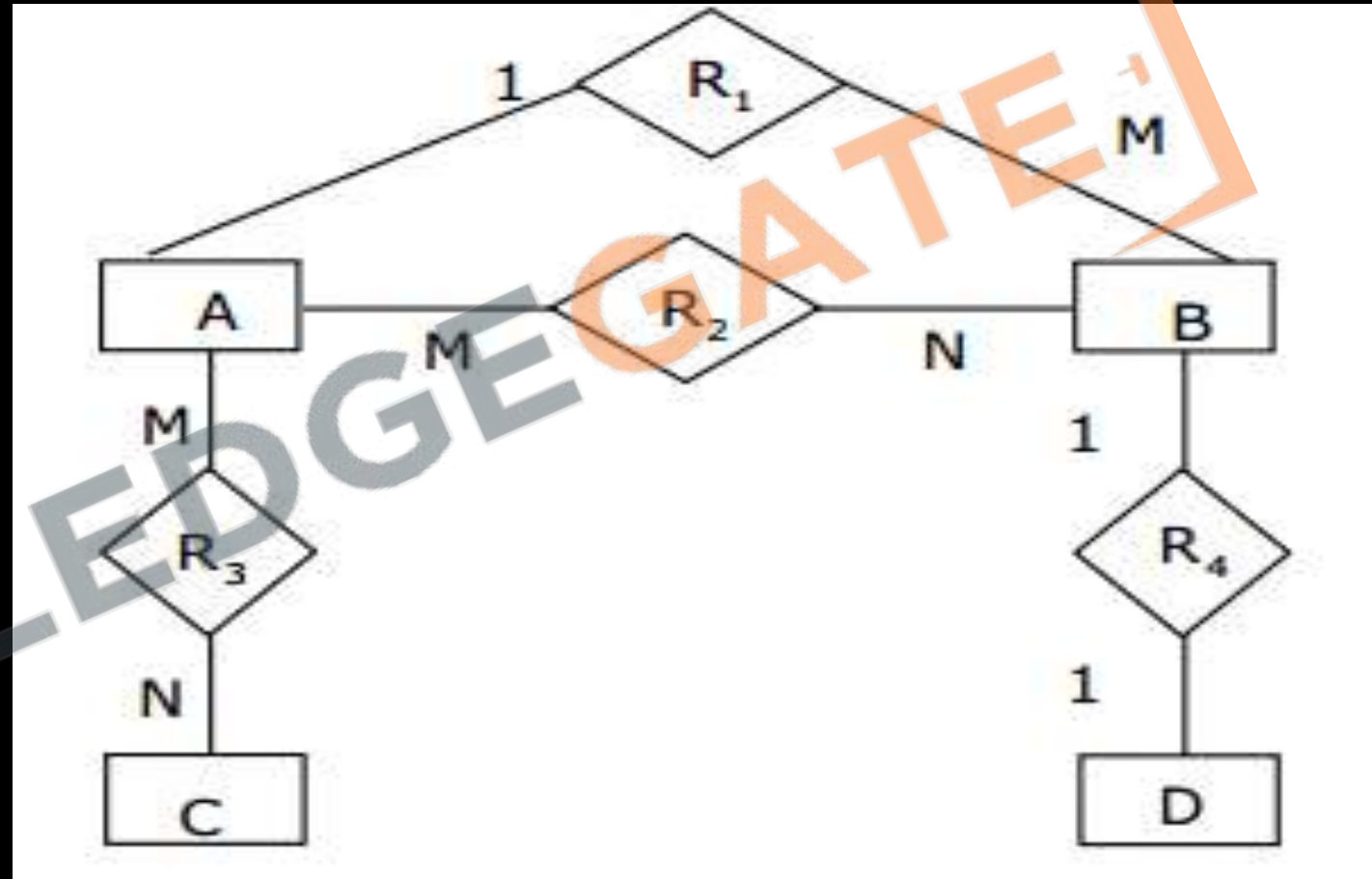
- Composite Attributes
 - A separate column must be taken for all simple attributes of the composite attribute.

<http://www.knowledgegate.in/gate>

Q The minimum number of tables required to convert the following ER diagram to relation model is _____



Q The minimum number of tables required to convert the following ER diagram to Relational model is _____



Q An ER model of a database consists of entity types A and B. These are connected by a relationship R which does not have its own attribute. Under which one of the following conditions, can the relational table for R be merged with that of A?

- (a)** Relationship R is one-to-many and the participation of A in R is total
- (b)** Relationship R is one-to-many and the participation of A in R is partial
- (c)** Relationship R is many-to-one and the participation of A in R is total
- (d)** Relationship R is many-to-one and the participation of A in R is partial

Q Consider an Entity-Relationship (ER) model in which entity sets E_1 and E_2 are connected by an $m:n$ relationship R_{12} , E_1 and E_3 are connected by a $1:n$ (1 on the side of E_1 and n on the side of E_3) relationship R_{13} . E_1 has two single-valued attributes a_{11} and a_{12} of which a_{11} is the key attribute. E_2 has two single-valued attributes a_{21} and a_{22} of which a_{21} is the key attribute. E_3 has two single valued attributes a_{31} and a_{32} of which a_{31} is the key attribute. The relationships do not have any attributes. If a relational model is derived from the above ER model, then the minimum number of relations that would be generated is

_____.

a) 2

b) 3

c) 4

d) 5

<http://www.knowledgegate.in/gate>

Q Let E_1 and E_2 be two entities in an E/R diagram with simple single-valued attributes. R_1 and R_2 are two relationships between E_1 and E_2 , where R_1 is one-to-many and R_2 is many-to-many. R_1 and R_2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model?

a) 2

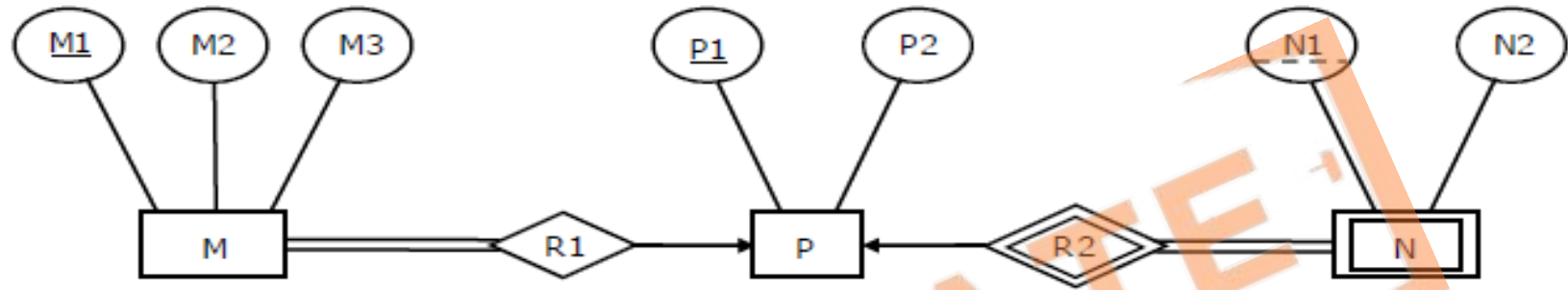
b) 3

c) 4

d) 5

<http://www.knowledgegate.in/gate>

Consider the following ER diagram



The minimum number of tables needed to represent M, N, P, R1, R2 is

Q Consider the following entity relationship diagram (ERD), where two entities E_1 and E_2 have a relation R of cardinality $1 : m$. The attributes of E_1 are A_{11} , A_{12} and A_{13} where A_{11} is the key attribute. The attributes of E_2 are A_{21} , A_{22} and A_{23} where A_{21} is the key attribute and A_{23} is a multi-valued attribute. Relation R does not have any attribute. A relational database containing minimum number of tables from the above ERD. The number of tables in the database is

(A) 2

(B) 3

(C) 5

(D) 4



- **ADVANTAGES OF E-R DIGRAM**
 - Constructs used in the ER diagram can easily be transformed into relational tables.
 - It is simple and easy to understand with minimum training.
- **DISADVANTAGE OF E-R DIGRAM**
 - Loss of information content.
 - Limited constraints representation.
 - It is overly complex for small projects.

Video chapters

- Chapter-1 (Transactions and concurrency control)
- Chapter-2 (ER-model)
- Chapter-3 (Relational model, Functional Dependencies, Keys)
- Chapter-4 (Normalization)
- Chapter-5 (File organization, indexing (e.g., B and B+ trees))
- Chapter-6 (Relational algebra, tuple calculus, SQL)

<http://www.knowledgegate.in/gate>

RELATIONAL DATABASE MANAGEMENT SYSTEM

- A Relational Database Management System (RDBMS) is a software system that uses a relational model to create, manage, and query data in databases through tables linked by defined relationships. Most modern commercial and open-source database applications are relational in nature.
- Based on the relational model specified by Edgar F. Codd. The father of modern relational database design in 1970.



BASICS OF RDBMS

- **Domain (set of permissible value in particular column)** is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn.
- E.g. Names: The set of character strings that represent names of persons.

**Domain/
Field/
Column/
Arity/
Degree**



NAME	ID	CITY	COUNTRY	HOBBY
NISHA	1	AGRA	INDIA	PLAYING
NIKITA	2	DELHI	INDIA	DANCING
AJAY	3	AGRA	INDIA	CHESS
ARPIT	4	PATNA	INDIA	READING

- **Table (Relation)** - A Relation is a set of tuples/rows/entities/records.
- **Tuple** - Each row of a Relation/Table is called Tuple.
- **Arity/Degree** - No. of columns/attributes of a Relation. E.g. - Arity is 5 in Table Student.
- **Cardinality** - No of rows/tuples/record of a Relational instance. E.g. - Cardinality is 4 in table Student.

Rows/Tuples/Record/
Cardinality

NAME	ID	CITY	COUNTRY	HOBBY
NISHA	1	AGRA	INDIA	PLAYING
NIKITA	2	DELHI	INDIA	DANCING
AJAY	3	AGRA	INDIA	CHESS
ARPIT	4	PATNA	INDIA	READING

Properties of Relational tables

1. Cells contains atomic values
2. Values in a column are of the same kind
3. Each row is unique
4. No two tables can have the same name in a relational schema.
5. Each column has a unique name
6. The sequence of rows is insignificant
7. The sequence of columns is insignificant.

Problems in relational database

- **Update Anomalies**- Anomalies that cause redundant work to be done during insertion into and Modification of a relation and that may cause accidental loss of information during a deletion from a relation
 - **Insertion Anomalies**
 - **Modification Anomalies**
 - **Deletion Anomalies**

- **Insertion anomalies:** An independent piece of information cannot be recorded into a relation unless an irrelevant information must be inserted together at the same time.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

- **Modification anomalies:** The update of a piece of information must occur at multiple locations.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

- **Deletion Anomalies:** The deletion of a piece of information unintentionally removes other information.

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

PK
↓

FK
↓

PK
↓

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	ec	Pqr

Conclusion

- Like every paragraph must have a single idea similarly every table must have a single idea and if a table contains more than one idea then that table must be decomposed until each table contains only one idea.
- We need some tools to approach this decomposition or normalization on large database which contains a number of table, and that tool is functional dependencies.

Purpose of Normalization

- Without normalization data base system may be inaccurate, slow and inefficient and they might not produce the data we expect. Normalization may be simply defined as refinement process.
- Which includes creating tables and establishing relationships between those tables according to rules designed both to protect data and make the database more flexible by eliminating two factors;
 - Redundancy
 - Inconsistent Dependency



<http://www.knowledgegate.in/gate>

Br_code



Br_hod_name

Br_code



Br_hod_name

<http://www.knowledgegate.in/gate>

Functional Dependency कोई बताता नहीं, इसकी feel आ जाती है



<http://www.knowledgegate.in/gate>

FUNCTIONAL DEPENDENCY

- A formal tool for analysis of relational schemas.
- In a Relation R, if ' α ' \subseteq R AND ' β ' \subseteq R, then attribute or a Set of attribute ' α ' Functionally derives an attribute or set of attributes ' β ', iff each ' α ' value is associated with precisely one ' β ' value.
- For all pairs of tuples t_1 and t_2 in R such that
 - If $T_1[\alpha] = T_2[\alpha]$
 - Then, $T_1[\beta] = T_2[\beta]$

X	Y	Z
1	4	2
1	4	3
2	6	3
3	2	2

Q Which of the following functional dependencies are satisfied by the instance?

(A) $XY \rightarrow Z$ and $Z \rightarrow Y$

(B) $YZ \rightarrow X$ and $Y \rightarrow Z$

(C) $YZ \rightarrow X$ and $X \rightarrow Z$

(D) $XZ \rightarrow Y$ and $Y \rightarrow X$

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

- Shortcut Steps to find whether a FD from $\alpha \rightarrow \beta$ can be concluded on a given instance or not
 1. Find Whether all values of α are different or not, if yes then FD valid
 2. Find Whether all values of β are same or not, if yes then FD valid
 3. जब कुछ भी काम ना करे, Try to find two same values of α on which we get different values of β

Q Consider the following relation instance, which of the following dependency doesn't hold

A) $A \rightarrow B$

B) $BC \rightarrow A$

C) $B \rightarrow C$

D) $AC \rightarrow B$

A	B	C
1	2	3
4	2	3
5	3	3

Q Which of the following dependency doesn't hold good?

A) $A \rightarrow BC$

B) $DE \rightarrow C$

C) $C \rightarrow DE$

D) $BC \rightarrow A$

A	B	C	D	E
a	2	3	4	5
2	a	3	4	5
a	2	3	6	5
a	2	3	6	6

- Trivial Functional dependency - If β is a subset of α , then the functional dependency $\alpha \rightarrow \beta$ will always hold.

जिसका होना न होना बराबर हो

X	Y	Z
1	4	2
1	4	3
2	6	3
3	2	2

1. α - Determinant (Determines β value).
2. β - Dependent (Dependent on α).
3. If $k \rightarrow R$, the K is a super key of R
4. Note: A functional dependency is a property of the relation schema R, not of a particular legal relation state/instance r of R.

X	Y	Z
1	4	2
1	4	3
2	6	3
3	2	2

Q Consider the relation $X(P, Q, R, S, T, U)$ with the following set of functional dependencies

$F = \{$

$\{P, R\} \rightarrow \{S, T\},$

$\{P, S, U\} \rightarrow \{Q, R\}$

$\}$

Which of the following is the trivial functional dependency in F^+ is closure of F ?

(a) $\{P, R\} \rightarrow \{S, T\}$

(b) $\{P, R\} \rightarrow \{R, T\}$

(c) $\{P, S\} \rightarrow \{S\}$

(d) $\{P, S, U\} \rightarrow \{Q\}$

<http://www.knowledgegate.in/gate>

Q Which of the following dependencies are satisfied by the relation instance?

$A \rightarrow B$

$B \rightarrow C$

$B \rightarrow A$

$C \rightarrow B$

$C \rightarrow A$

$A \rightarrow C$

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

ATTRIBUTES CLOSURE/CLOSURE ON ATTRIBUTE SET/ CLOSURE SET OF ATTRIBUTES

- Attribute closure of an attribute set can be defined as set of attributes which can be functionally determined from F.
 - DENOTED BY F^+
- Set of all attributes Functionally determined by X either directly from FD'S or logically derived.

<http://www.knowledgegate.in/gate>

DIRECT METHOD

E.g. In a Relation $R (A, B, C, D)$, with set of functional dependencies as-

$\{$
 $A \rightarrow B$
 $B \rightarrow C$
 $AB \rightarrow D$
 $\}$

$A^+ =$

Q R(ABCDEFGG)

$A \rightarrow B$

$BC \rightarrow DE$

$AEG \rightarrow G$

$(AC)^+ = ?$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)

$A \rightarrow BC$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

$(B)^+ =$

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)

$AB \rightarrow C$

$BC \rightarrow AD$

$D \rightarrow E$

$CF \rightarrow B$

$(AB)^+ =$

<http://www.knowledgegate.in/gate>

Q R(ABCDEFGH)

$A \rightarrow BC$

$CD \rightarrow E$

$E \rightarrow C$

$D \rightarrow AEH$

$ABH \rightarrow BD$

$DH \rightarrow BC$

$(BCD)^+ =$

<http://www.knowledgegate.in/gate>

Q Suppose the following functional dependencies hold on a relation U with attributes P,Q,R,S, and T:

$P \rightarrow QR$

$RS \rightarrow T$

Which of the following functional dependencies can be inferred from the above functional dependencies?

(A) $PS \rightarrow T$

(B) $R \rightarrow T$

(C) $P \rightarrow R$

(D) $PS \rightarrow Q$

<http://www.knowledgegate.in/gate>

ARMSTRONG'S AXIOMS

1. An **axiom** or **postulate** is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments.
2. **Armstrong's axioms** are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database.
3. They were developed by William W. Armstrong in his 1974 paper.
4. The axioms are sound in generating only functional dependencies in the closure of a set of functional dependencies (denoted as F^+) when applied to that set (denoted as F).



Armstrong Axioms

- **Reflexivity:** If Y is a subset of X , then $X \rightarrow Y$
- **Augmentation:** If $X \rightarrow Y$, then $XZ \rightarrow YZ$
- **Transitivity:** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

From these rules, we can derive these secondary rules-

- **Union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- **Decomposition:** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- **Pseudo transitivity:** If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
- **Composition:** If $X \rightarrow Y$ and $Z \rightarrow W$, then $XZ \rightarrow YW$

Why Armstrong axioms refers to the Sound and Complete

- By sound, we mean that given a set of functional dependencies F specified on a relation schema R , any dependency that we can infer from F by using the primary rules of Armstrong axioms holds in every relation state r of R that satisfies the dependencies in F .
- By complete, we mean that using primary rules of Armstrong axioms repeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F .

APPLICATION OF ATTRIBUTE CLOSURE

Equivalence of Two FD sets-

Two FD sets F_1 and F_2 are equivalent if –

$$F_1^+ = F_2^+$$

Or

$$F_1 \subseteq F_2^+ \text{ and } F_2 \subseteq F_1^+$$

Race एक जगह से शुरू होने जरूरी है

<http://www.knowledgegate.in/gate>

Q Consider the following set of fd R(ACDEH)

F	G
$A \rightarrow C$	$A \rightarrow CD$
$AC \rightarrow D$	$E \rightarrow AH$
$E \rightarrow AD$	
$E \rightarrow H$	

Q Consider the following set of fd R(ABCDE)

F	G
$B \rightarrow CD$	$B \rightarrow CDE$
$AD \rightarrow E$	$A \rightarrow BC$
$B \rightarrow A$	$AD \rightarrow E$

Q consider the following relation R(PQRS)

F	G
$P \rightarrow Q$	$P \rightarrow QR$
$Q \rightarrow R$	
$R \rightarrow S$	$R \rightarrow S$

To find the MINIMAL COVER /CANONICAL COVER/IRREDUCIBLE SET

- Minimal cover- It means to eliminate any kind of redundancy from a FD set.
- A canonical cover of a set of functional dependencies, F is a simplified set of functional dependencies that has the same closure as the original set F.
- There may be any following type of redundancy in the set of functional dependencies: -
 - Complete production may be Redundant.
 - One or more than one attributes may be redundant on right hand side of a production.
 - One or more than one attributes may be redundant on Left hand side of a production.

Gate aspirant के हाथों से minimize होने के बाद



<http://www.knowledgegate.in/gate>

Procedure to find MINIMAL COVER

- Use decomposition rule wherever applicable so that RHS of a production/FD contains only single attribute.
- Remove extraneous attribute on LHS of a production by finding the closure for every possible subset, if in any case the closure is same it means remaining attributes are redundant.
- For every production find the closure value of LHS of production keeping the production in a set, and next time ignoring the production to be in a set. If both closure set matches, it means the production is redundant.

$Q \rightarrow R(ABCD)$

$A \rightarrow B$

$C \rightarrow B$

$D \rightarrow ABC$

$AC \rightarrow D$

<http://www.knowledgegate.in/gate>

Q The following functional dependencies hold true for the relational schema {V, W, X, Y, Z}:

$V \rightarrow W$

$VW \rightarrow X$

$Y \rightarrow VX$

$Y \rightarrow Z$

Which of the following is irreducible equivalent for this set of functional dependencies?

(a)	(b)	(c)	(d)
$V \rightarrow W$	$V \rightarrow W$	$V \rightarrow W$	$V \rightarrow W$
$V \rightarrow X$	$W \rightarrow X$	$V \rightarrow X$	$W \rightarrow X$
$Y \rightarrow V$	$Y \rightarrow V$	$Y \rightarrow V$	$Y \rightarrow V$
$Y \rightarrow Z$	$Y \rightarrow Z$	$Y \rightarrow X$	$Y \rightarrow X$
		$Y \rightarrow Z$	$Y \rightarrow Z$

Q R(VWXYZ)

P	Q
$V \rightarrow W$	$V \rightarrow W$
$VW \rightarrow X$	$V \rightarrow X$
$Y \rightarrow VX$	$Y \rightarrow V$
$Y \rightarrow Z$	$Y \rightarrow Z$

<http://www.knowledgegate.in/gate>

Key



<http://www.knowledgegate.in/gate>

Super key

- Set of attributes using which we can identify each tuple uniquely is called Super key, i.e. the set of attributes used to differentiate each tuple of a relation.
- Let X be a set of attributes in a Relation R , if X^+ (Closure of X) determines all attributes of R then X is said to be Super key of R .
- There should be at least one Super key with Not Null constraint.

Q Consider a relation R(A, B, C, D, E) with the following three functional dependencies.

$AB \rightarrow C$; $BC \rightarrow D$; $C \rightarrow E$;

The number of super keys in the relation R is_____.

A B C D E

Q The maximum number of super keys for the relation schema $R(E, F, G, H)$ with E as the key is

a) 5

E F G H

b) 6

c) 7

d) 8

<http://www.knowledgegate.in/gate>

Candidate key

1. Minimum set of attributes that differentiates the tuple of a Relation. No proper subset as super key Also called as **MINIMAL SUPER KEY**.
2. There should be at least one candidate key with Not Null constraint.
3. Prime attribute - Attributes that are member of candidate Keys are called Prime attributes.

Primary key

1. One of the candidate keys is selected by database administrator as a Primary Key.
2. Primary Key attribute are not allowed to have Null values.
3. At most one Primary Key per table is allowed in RDMS.
4. Candidate key which are not chosen as primary key is alternate key.

Q Which of the following is NOT a superkey in a relational schema with attributes V, W, X, Y, Z and primary key VY?

(a) VXYZ

(b) VWXZ

(c) VWXY

(d) VWXYZ

<http://www.knowledgegate.in/gate>

Roll no	name	Age	Br_code	Br_name	Br_hod_name
1	A	19	101	Cs	Abc
2	B	18	101	Cs	Abc
3	C	20	101	Cs	Abc
4	D	20	102	Ec	Pqr

PK
↓

FK
↓

PK
↓

Roll no	name	Age	Br_code
1	A	19	101
2	B	18	101
3	C	20	101
4	D	20	102

Br_code	Br_name	Br_hod_name
101	Cs	Abc
102	ec	Pqr



Roll no	CR
1	1
2	2
3	1
4	2
5	1
6	2
7	1
8	2
9	1
10	2
11	1
12	2
13	1
14	2
15	null

Foreign Keys

- A foreign key is a column or group of columns in a relational database table that refers the primary key of the same table or some other table to represent relationship.
- The ***concept of referential integrity*** is derived from foreign key theory.

<http://www.knowledgegate.in/gate>

Q Consider the following statements S1 and S2 about the relational data model:

- S1: A relation scheme can have at most one foreign key.
- S2: A foreign key in a relation scheme R cannot be used to refer to tuples of R.

Which one of the following choices is correct?

- (a) Both S1 and S2 are true
- (b) S1 is true and S2 is false
- (c) S1 is false and S2 is true
- (d) Both S1 and S2 are false

Q The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with on-delete cascade.

The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted is:

- a) (3,4) and (6,4)
- b) (5,2) and (7,2)
- c) (5,2),(7,2) and (9,5)
- d) (3,4),(4,3) and (6,4)

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

Q Consider the following tables T_1 and T_2 . In table T_1 , **P** is the primary key and **Q** is the foreign key referencing **R** in table T_2 with on-delete cascade and on-update cascade. In table T_2 , **R** is the primary key and **S** is the foreign key referencing **P** in table T_1 with on-delete set NULL and on-update cascade. In order to delete record $\langle 3, 8 \rangle$ from table T_1 , the number of additional records that need to be deleted from table T_1 is _____.

T1		T2	
P	Q	R	S
2	2	2	2
3	8	8	3
7	3	3	2
5	8	9	7
6	9	5	7
8	5	7	2
9	8		

- **Composite key** – Composite key is a key composed of more than one column sometimes it is also known as concatenated key.
- **Secondary key** – Secondary key is a key used to speed up the search and retrieval contrary to primary key, a secondary key does not necessarily contain unique values.

Q R(ABCD)

$A \rightarrow B$

A

B

C

D

$B \rightarrow C$

$C \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCD)

AB → CD

D → A

A

B

C

D

<http://www.knowledgegate.in/gate>

Q R(ABC)

$AB \rightarrow C$

A

B

C

$C \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCDEFGH IJ)

$AB \rightarrow C$

A B C D E F G H I J

$A \rightarrow DE$

$B \rightarrow F$

$F \rightarrow GH$

$D \rightarrow IJ$

<http://www.knowledgegate.in/gate>

Q R(ABCDEFGHJI)

$AB \rightarrow C$

A B C D E F G H I J

$AD \rightarrow GH$

$BD \rightarrow EF$

$A \rightarrow I$

$H \rightarrow J$

<http://www.knowledgegate.in/gate>

Q R(ABCDEFGH)

A → BC

A B C D E F G H

ABE → CDGH

C → GD

D → G

E → F

<http://www.knowledgegate.in/gate>

Q R(ABCDE)

$A \rightarrow B$

A

B

C

D

E

$BC \rightarrow E$

$DE \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)

$A \rightarrow B$

A

B

C

D

E

$BC \rightarrow E$

$DE \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)

AB → CD

D → A

BC → DE

A

B

C

D

E

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)

AB \rightarrow C

A

B

C

D

E

F

DC \rightarrow AE

E \rightarrow F

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)

$AB \rightarrow C$

A

B

C

D

E

F

$C \rightarrow D$

$D \rightarrow BE$

$E \rightarrow F$

$F \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(WXYZ)

$Z \rightarrow W$

W

X

Y

Z

$Y \rightarrow XZ$

$XW \rightarrow Y$

<http://www.knowledgegate.in/gate>

Q R(VWXYZ)

$Z \rightarrow Y$

V

W

X

Y

Z

$Y \rightarrow Z$

$X \rightarrow YV$

$VW \rightarrow X$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)

$A \rightarrow BC$

A

B

C

D

E

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

<http://www.knowledgegate.in/gate>

Q Consider the relation scheme $R = \{E, F, G, H, I, J, K, L, M, N\}$ and the set of functional dependencies $\{\{E, F\} \rightarrow \{G\}, \{F\} \rightarrow \{I, J\}, \{E, H\} \rightarrow \{K, L\}, K \rightarrow \{M\}, L \rightarrow \{N\}\}$ on R . What is the key for R ?

A) $\{E, F\}$

E F G H I J K L M N

B) $\{E, F, H\}$

C) $\{E, F, H, K, L\}$

D) $\{E\}$

Q Given the STUDENTS relation as shown below.

<i>StudentID</i>	<i>StudentName</i>	<i>StudentEmail</i>	<i>StudentAge</i>	<i>CPI</i>
2345	Shankar	shankar@math	X	9.4
1287	Swati	swati@ee	19	9.5
7853	Shankar	shankar@cse	19	9.4
9876	Swati	swati@mech	18	9.3
8765	Ganesh	ganesh@civil	19	8.7

For (StudentName, StudentAge) to be the key for this instance, the value X should not be equal to _____

Q Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values. $F = \{CH \rightarrow G, A \rightarrow BC, B \rightarrow CFH, E \rightarrow A, F \rightarrow EG\}$ is a set of functional dependencies (FDs) so that F^+ is exactly the set of FDs that hold for R. How many candidate keys does the relation R have?

(A) 3

(B) 4

(C) 5

(D) 6

A B C D E F G H

Q Consider a relation scheme $R = (A, B, C, D, E, H)$ on which the following functional dependencies hold: $\{A \rightarrow B, BC \rightarrow D, E \rightarrow C, D \rightarrow A\}$. What are the candidate keys of R ?

(A) AE, BE

A B C D E H

(B) AE, BE, DE

(C) AEH, BEH, BCH

(D) AEH, BEH, DEH

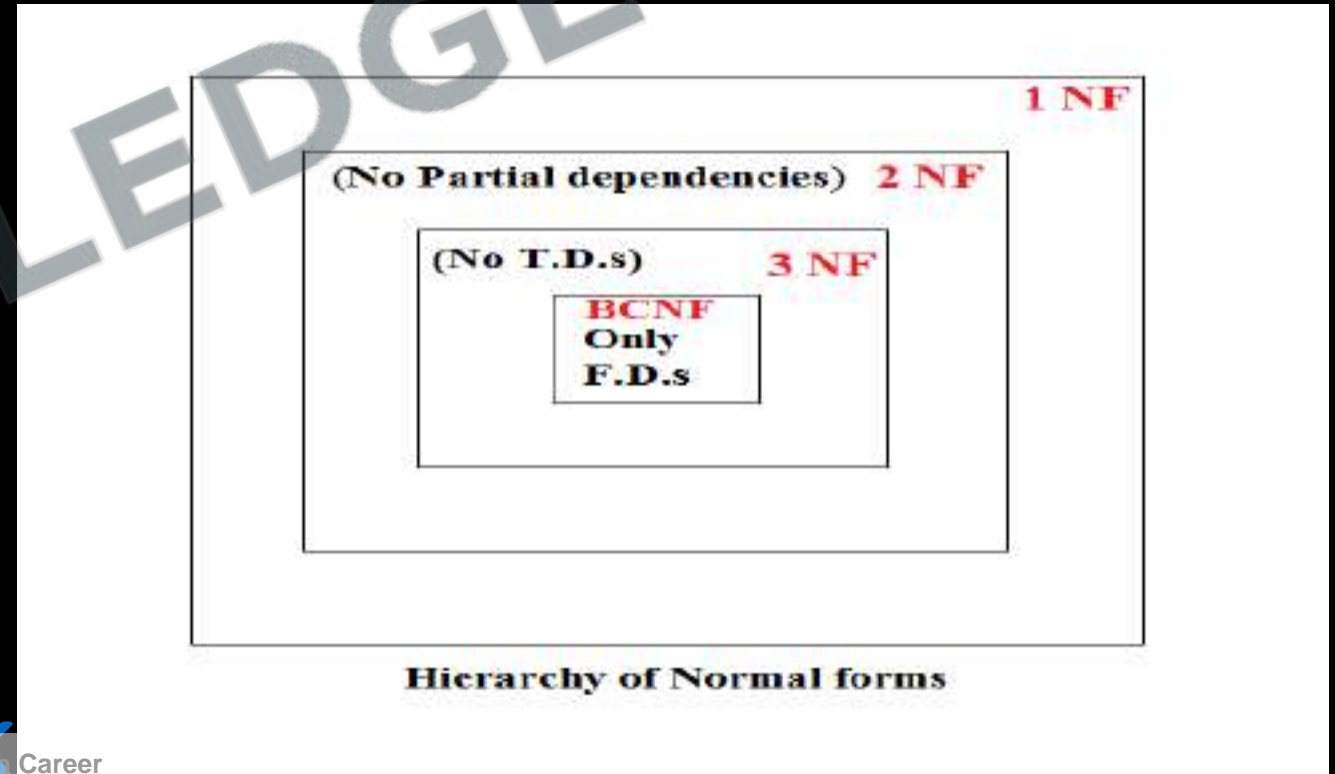
<http://www.knowledgegate.in/gate>

Video chapters

- Chapter-1 (Transactions and concurrency control)
- Chapter-2 (ER-model)
- Chapter-3 (Relational model, Functional Dependencies, Keys)
- Chapter-4 (Normalization)
- Chapter-5 (File organization, indexing (e.g., B and B+ trees))
- Chapter-6 (Relational algebra, tuple calculus, SQL)

<http://www.knowledgegate.in/gate>

- **Normalization of data** (Decomposition of Relation) can be considered a process of analyzing the given relation schema to achieve the desirable properties of minimizing redundancy using Decomposition.
- The tool we use for normalization is functional dependencies and candidate keys.
- Functional dependency can be used only to normalize up to BCNF.
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree.
- 1NF>>>2NF>>3NF>>BCNF



FIRST NORMAL FORM

- A Relation table is said to be in first normal form iff each attribute in each cell have single value(atomic). Means a Relation should not contain any multivalued or composite attributes.
- Other implications of first normal form
 - Every row should be unique, that is no two rows should have the same values of all the attributes.
 - There must be a primary key.
 - Every column should have a unique name
 - Order of row and column is irrelevant

Prime attribute: - A attribute is said to be prime if it is part of any of the candidate key

Non-Prime attribute: - A attribute is said to be non-prime if it is not part of any of the candidate key

Eg R(ABCD)
AB \rightarrow CD

Here candidate key is AB so, A and B are prime attribute, C and D are non-prime attributes.

<http://www.knowledgegate.in/gate>

PARTIAL DEPENDENCY- When a non – prime attribute is dependent only on a part (Proper subset) of candidate key then it is called partial dependency. (PRIME > NON-PRIME)

TOTAL DEPENDENCY- When a non – prime attribute is dependent on the entire candidate key then it is called total dependency.

e.g. R(ABCD) $AB \rightarrow D$, $A \rightarrow C$

<http://www.knowledgegate.in/gate>

SECOND NORMAL FORM

- Relation R is in 2NF if,
 - R should be in 1 NF.
 - R should not contain any Partial dependency. (that is every non-prime attribute should be fully dependent upon candidate key)

Q R(A, B, C) $B \rightarrow C$

A	B	C
a	1	X
b	2	Y
a	3	Z
C	3	Z
D	3	Z
E	3	Z

A	B
A	1
B	2
A	3
C	3
D	3
E	3

B	C
1	X
2	Y
3	Z

<http://www.knowledgegate.in/gate>

TRANSITIVE DEPENDENCY – A functional dependency from non-Prime attribute to non-Prime attribute is called transitive

E.g.- R(A, B, C, D) with A as a candidate key

$A \rightarrow B$

$B \rightarrow C$ [transitive dependency]

$C \rightarrow D$ [transitive dependency]

<http://www.knowledgegate.in/gate>

THIRD NORMAL FORM

- Let R be the relational schema, it is said to be in 3 NF
 - R should be in 2NF
 - It must not contain any transitive dependency

THIRD NORMAL FORM DIRECT DEFINATION

- A relational schema R is said to be 3 NF if every functional dependency in R from $\alpha \rightarrow \beta$, either α is super key or β is the prime attribute

A	B	C
A	1	P
B	2	Q
C	2	Q
D	2	Q
E	3	R
F	3	R
G	4	S

A	B
A	1
B	2
C	2
D	2
E	3
F	3
G	4

B	C
1	P
2	Q
3	R
4	S

<http://www.knowledgegate.in/gate>

$R(A, B, C)$

$AB \rightarrow C$

$C \rightarrow B$

A	B	C
A	C	B
B	B	C
B	A	D
A	A	E
C	C	B
D	C	B
E	C	B
F	C	B

A	B
A	B
B	B
B	A
A	A
C	C
D	C
e	C
f	c

C	B
B	C
C	B
D	A
E	A

BCNF (BOYCE CODD NORMAL FORM)

A relational schema R is said to be BCNF if every functional *dependency* in R from $\alpha \twoheadrightarrow \beta$

α must be a super key

E.g.- R (A, B, C, D)

{

AB \rightarrow C [No violation of 2NF, 3NF, BCNF]

C \rightarrow D [violation of BCNF, C not a candidate/super key]

D \rightarrow A [violation of BCNF, D not a candidate/super key]

} Candidate key = {AB}, {DB}, {CB}

<http://www.knowledgegate.in/gate>

Some important note points on Normalization:

- If a relation R does not contain any non- trivial dependency, then R is in BCNF.
- A Relation with two attributes is always in BCNF.
- A relation schema R consist of only simple candidate key then, R is always in 2NF but may or may not be in 3NF or BCNF.
- A Relation schema R consist of only prime attributes then R is always in 3NF, but may or may not be in BCNF.
- A relation schema R in 3NF and with only simple candidate keys, then R surely in BCNF.

Q R(ABC) (AB, BC)

$AB \rightarrow C$

A

B

C

$C \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCD)(AB)

AB \rightarrow C

A

B

C

D

B \rightarrow D

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(CE)

CE \rightarrow D

A B C D E

D \rightarrow B

C \rightarrow A

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(ACD, BCD, CDE)

$A \rightarrow B$

A B C D E

$BC \rightarrow E$

$DE \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCD)(AB, AD, BC, CD)

$AB \rightarrow CD$

A

B

C

D

$C \rightarrow A$

$D \rightarrow B$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(AB)

AB \rightarrow C

A B C D E

B \rightarrow D

D \rightarrow E

<http://www.knowledgegate.in/gate>

Q R(ABCDEFGH)(AE)

A → BC

A B C D E F G H

ABE → CDGH

C → GD

D → G

E → F

<http://www.knowledgegate.in/gate>

$Q R(WXYZ)(Y, XW, XZ)$

$Z \rightarrow W$

W

X

Y

Z

$Y \rightarrow XZ$

$XW \rightarrow Y$

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)(ABC, ACD)

ABC → D

A B C D E F

ABD → E

CD → F

CDF → B

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(AB)

AB → C

A B C D E

B → D

D → E

<http://www.knowledgegate.in/gate>

$Q \rightarrow R(ABCDE)(A, E, BC, CD)$

$A \rightarrow BC$

A B C D E

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

<http://www.knowledgegate.in/gate>

Q (ABCDE)(ACD, BCD, CDE)

$A \rightarrow B$

A

B

C

D

E

$BC \rightarrow E$

$DE \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(ABD)

A

B

C

D

E

BD \rightarrow E

A \rightarrow C

<http://www.knowledgegate.in/gate>

Q R(ABCDEF) (A, BC, DEF)

A → BCDEF

BC → ADEF

DEF → ABC

A B C D E F

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(ac)

A → B

A

B

C

D

E

B → E

C → D

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(AB, BC, BD)

AB \rightarrow CD

A

B

C

D

E

D \rightarrow A

BC \rightarrow DE

<http://www.knowledgegate.in/gate>

Q R(ABCD)(AB, BD)

AB \rightarrow CD

A

B

C

D

D \rightarrow A

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)(BF)

AB → C

A B C D E F

C → D

B → AE

<http://www.knowledgegate.in/gate>

Q R(ABCDEFGH IJ)(AB)

AB \rightarrow C

A B C D E F G H I J

A \rightarrow DE

B \rightarrow F

F \rightarrow GH

D \rightarrow IJ

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(BC, CD)

BC → ADE

A

B

C

D

E

D → B

<http://www.knowledgegate.in/gate>

Q $R(ABCD)(AD, BD, CD)$

$A \rightarrow B$

A

B

C

D

$B \rightarrow C$

$C \rightarrow A$

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)(ABD, BCD)

AB → C

A B C D E F

DC → AE

E → F

<http://www.knowledgegate.in/gate>

$Q \ R(VWXYZ)(VW, XW)$

$Z \rightarrow Y$

V

W

X

Y

Z

$Y \rightarrow Z$

$X \rightarrow YV$

$VW \rightarrow X$

<http://www.knowledgegate.in/gate>

Q R(ABCDE)(AC)

$A \rightarrow B$

A

B

C

D

E

$B \rightarrow E$

$C \rightarrow D$

<http://www.knowledgegate.in/gate>

Q R(ABCDEF)(C, D, AB, BE, BF)

$AB \rightarrow C$

A

B

C

D

E

F

$C \rightarrow D$

$D > BE$

$E > F$

$F > A$

<http://www.knowledgegate.in/gate>

Q Consider the following four relational schemas. For each schema, all non-trivial functional dependencies are listed. The underlined attributes are the respective primary keys.

Schema I: Registration (rollno, courses)

Field 'courses' is a set-valued attribute containing the set of courses a student has registered for.

Non-trivial functional dependency
 $\text{rollno} \rightarrow \text{courses}$

Schema II: Registration (rollno, coursid, email)

Non-trivial functional dependencies:
 $\text{rollno}, \text{coursid} \rightarrow \text{email}$
 $\text{email} \rightarrow \text{rollno}$

Schema III: Registration (rollno, coursid, marks, grade)

Non-trivial functional dependencies:
 $\text{rollno}, \text{coursid} \rightarrow \text{marks}, \text{grade}$
 $\text{marks} \rightarrow \text{grade}$

Schema IV: Registration (rollno, coursid, credit)

Non-trivial functional dependencies:
 $\text{rollno}, \text{coursid} \rightarrow \text{credit}$
 $\text{coursid} \rightarrow \text{credit}$

Which one of the relational schemas above is in 3NF but not in BCNF?

- (a) Schema 1
- (b) Schema 2
- (c) Schema 3
- (d) Schema 4

Q Which one of the following statements is FALSE?

- a)** Any relation with two attributes is in BCNF
- b)** A relation in which every key has only one attribute is in 2NF
- c)** A prime attribute can be transitively dependent on a key in a 3NF relation
- d)** A prime attribute can be transitively dependent on a key in a BCNF relation

<http://www.knowledgegate.in/gate>

Multivalued Dependency

- Multivalued dependencies Denoted by, $A \twoheadrightarrow B$, Means, for every value of A, there may exist more than one value of B.
- A **trivial multivalued dependency** $X \twoheadrightarrow Y$ is one where either Y is a subset of X, or X and Y together form the whole set of attributes of the relation.
- If there is functional dependency from $A \rightarrow B$, then there will also a multivalued functional dependency from $A \twoheadrightarrow B$.

E.g. let the constraint specified by MVD in relation **Student** as

$S_name \twoheadrightarrow Club_name$

$S_name \twoheadrightarrow P_no$

S_Name	Club_name
Kamesh	Dance
Kamesh	Guitar

S_Name	P_no
Kamesh	123
Kamesh	789

S_Name	Club_name	P_no
Kamesh	Dance	123
Kamesh	Guitar	123
Kamesh	Dance	789
Kamesh	Guitar	789

NOTE: The above Student schema is in BCNF as no functional dependency holds on EMP, but still redundancy due to MVD.

- Each row indicates that a given restaurant can deliver a given variety. The table has no non-key attributes because its only key is {Restaurant, Variety, Delivery Area}. Therefore, it meets all normal forms up to BCNF.
- If we assume, however, that Variety offered by a restaurant are not affected by delivery area (i.e. a restaurant offers all Variety it makes to all areas it supplies), then it does not meet 4NF. The problem is that the table features two non-trivial multivalued dependencies on the {Restaurant} attribute (which is not a super key). The dependencies are:
 - {Restaurant} \twoheadrightarrow {Variety}
 - {Restaurant} \twoheadrightarrow {Delivery Area}

Restaurant Delivery Permutations		
Restaurant	Variety	Delivery Area
Chatora Sweets	Samosa	Hatibagan Market
Chatora Sweets	Samosa	Chandni Chowk
Chatora Sweets	Samosa	Koramangala
Chatora Sweets	Dosa	Hatibagan Market
Chatora Sweets	Dosa	Chandni Chowk
Chatora Sweets	Dosa	Koramangala
Moolchand	Ladoo	Koramangala
Moolchand	Dosa	Koramangala
Thaggu	Samosa	Hatibagan Market
Thaggu	Samosa	Chandni Chowk
Thaggu	Ladoo	Hatibagan Market
Thaggu	Ladoo	Chandni Chowk

- These non-trivial multivalued dependencies on a non-superkey reflect the fact that the varieties a restaurant offers are independent from the areas to which the restaurant delivers. This state of affairs leads to redundancy in the table:
- for example, we are told three times that Chatora Sweets offers Dosa, and if Chatora Sweets starts producing Momo then we will need to add multiple rows, one for each of Chatora Sweets's delivery areas.

Restaurant Delivery Permutations		
Restaurant	Variety	Delivery Area
Chatora Sweets	Samosa	Hatibagan Market
Chatora Sweets	Samosa	Chandni Chowk
Chatora Sweets	Samosa	Koramangala
Chatora Sweets	Dosa	Hatibagan Market
Chatora Sweets	Dosa	Chandni Chowk
Chatora Sweets	Dosa	Koramangala
Moolchand	Ladoo	Koramangala
Moolchand	Dosa	Koramangala
Thaggu	Samosa	Hatibagan Market
Thaggu	Samosa	Chandni Chowk
Thaggu	Ladoo	Hatibagan Market
Thaggu	Ladoo	Chandni Chowk

- If we have two or more multivalued *independent* attributes in the same relation schema, we get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation state consistent and to maintain the independence among the attributes involved. This constraint is specified by a multivalued dependency.

Delivery Areas By Restaurant	
Restaurant	Delivery Area
Chatora Sweets	Hatibagan Market
Chatora Sweets	Chandni Chowk
Chatora Sweets	Koramangala
Moolchand	Koramangala
Thaggu	Hatibagan Market
Thaggu	Chandni Chowk

Varieties By Restaurant	
Restaurant	Pizza Variety
Chatora Sweets	Samosa
Chatora Sweets	Dosa
Moolchand	Ladoo
Moolchand	Dosa
Thaggu	Samosa
Thaggu	Ladoo

- A relation is in 4NF iff
 - It is in BCNF
 - There must not exist any non-trivial multivalued dependency.
- Each MVD is decomposed in separate table, where it becomes trivial MVD.

<http://www.knowledgegate.in/gate>

- A 1992 paper by Margaret S. Wu notes that the teaching of database normalization typically stops short of 4NF, perhaps because of a belief that tables violating 4NF (but meeting all lower normal forms) are rarely encountered in business applications.
- This belief may not be accurate, however. Wu reports that in a study of forty organizational databases, over 20% contained one or more tables that violated 4NF while meeting all lower normal forms.

<http://www.knowledgegate.in/gate>

Lossy/Lossless-Dependency Preserving Decomposition

- Because of a normalization a table is Decomposed into two or more tables, but during this decomposition we must ensure satisfaction of some properties out of which the most important is lossless join property / decomposition.

- if we decompose a table r into two tables r_1 and r_2 because of normalization then at some later stage if we want to join (combine) (natural join) these tables r_1 and r_2 , then we must get back the original table r , without any extra or less tuple. But some information may be lost during retrieval of original relation or table. For e.g.

r		
A	B	C
1	a	p
2	b	q
3	a	r

r ₁	
A	B

r ₂	
B	C

r		
A	B	C
1	a	p
2	b	q
3	a	r

r ₁	
A	B
1	a
2	b
3	a

r ₂	
B	C
a	p
b	q
a	r

A	B	C

- Decomposition is lossy if $R_1 \bowtie R_2 \supset R$
- Decomposition is lossy if $R \supset R_1 \bowtie R_2$

<http://www.knowledgegate.in/gate>

- Decomposition is lossless if $R_1 \bowtie R_2 = R$ "The decomposition of relation R into R_1 and R_2 is lossless when the join of R_1 and R_2 yield the same relation as in R." which guarantees that the spurious (extra or less) tuple generation problem does not occur with respect to the relation schemas created after decomposition.
- This property is extremely critical and must be achieved at any cost.
- lossless Decomposition / NonAdditive Join Decomposition

<http://www.knowledgegate.in/gate>

A	B	C	D	E
A	122	1	W	A
E	236	4	X	B
A	199	1	Y	C
B	213	2	Z	D

<http://www.knowledgegate.in/gate>

How to check for lossless join decomposition using FD set, following conditions must hold:

- Union of Attributes of R_1 and R_2 must be equal to attribute of R . Each attribute of R must be either in R_1 or in R_2 . $\text{Att}(R_1) \cup \text{Att}(R_2) = \text{Att}(R)$
- Intersection of Attributes of R_1 and R_2 must not be NULL. $\text{Att}(R_1) \cap \text{Att}(R_2) \neq \Phi$
- Common attribute must be a key for at least one relation (R_1 or R_2)
 - $\text{Att}(R_1) \cap \text{Att}(R_2) \rightarrow (R_1)$ or $\text{Att}(R_1) \cap \text{Att}(R_2) \rightarrow (R_2)$

Dependency Preserving Decomposition

- Let relation R be decomposed into Relations $R_1, R_2, R_3, \dots, R_N$ with their respective functional Dependencies set as $F_1, F_2, F_3, \dots, F_N$, then the Decomposition is Dependency Preserving iff
 - $\{F_1 \cup F_2 \cup F_3 \cup F_4 \dots \cup F_N\}^+ = F^+$
- Dependency preservation property, although desirable, is sometimes sacrificed.

$Q R (A, B, C)$

$A \rightarrow B, B \rightarrow C, C \rightarrow A$

$R_1(A, B)$ AND $R_2(B, C)$

<http://www.knowledgegate.in/gate>

Q R(ABCDEFG)(NF) $R_1(ABC)$ $R_2(ABDE)$ $R_3(EG)$

$AB \rightarrow C$

$AC \rightarrow B$

$AD \rightarrow E$

$B \rightarrow D$

$BC \rightarrow A$

$E \rightarrow G$

<http://www.knowledgegate.in/gate>

Q Consider the relation R (ABCD) with the FD set $F = \{A \rightarrow BC, B \rightarrow CD, C \rightarrow AD\}$ which is decomposed into set of tables $D = \{(AB), (BC), (CD)\}$. Which of the following is true about the decomposition D?

- a) It is lossless and dependency preserving
- b) It is lossy but dependency preserving
- c) It is lossless but dependencies are not preserved
- d) It is neither lossless nor dependency preserving

Q Let the set of functional dependencies $F = \{QR \rightarrow S, R \rightarrow P, S \rightarrow Q\}$ hold on a relation schema $X = (PQRS)$. X is not in BCNF. Suppose X is decomposed into two schemas Y and Z , where $Y = (PR)$ and $Z = (QRS)$.

Consider the two statements given below.

I. Both Y and Z are in BCNF

II. Decomposition of X into Y and Z is dependency preserving and lossless

Which of the above statements is/are correct? **(GATE- 2019) (1 Marks)**

(a) I only

(b) Neither I nor II

(c) II only

(d) Both I and II

<http://www.knowledgegate.in/gate>

Q Consider a schema $R(A,B,C,D)$ and functional dependencies $A \rightarrow B$ and $C \rightarrow D$. Then the decomposition of R into $R_1(AB)$ and $R_2(CD)$ is **(GATE-2001) (2 Marks)**

- (A) dependency preserving and lossless join
- (B) lossless join but not dependency preserving
- (C) dependency preserving but not lossless join
- (D) not dependency preserving and not lossless join

Q Consider the relation $R(P,Q,S,T,X,Y,Z,W)$ with the following functional dependencies.

$PQ \rightarrow X; P \rightarrow YX; Q \rightarrow Y; Y \rightarrow ZW$

$D_1: R = [(P,Q,S,T); (P,T,X); (Q,Y); (Y,Z,W)]$

$D_2: R = [(P,Q,S); (T,X); (Q,Y); (Y,Z,W)]$

Consider the decomposition of the relation R into the constituent relations according to the following two decomposition schemes. Which one of the following options is correct? **(GATE 2021) (2 MARKS)**

- a) D_1 is a lossless decomposition, but D_2 is a lossy decomposition
- b) D_1 is a lossy decomposition, but D_2 is a lossless decomposition
- c) Both D_1 and D_2 are lossless decompositions
- d) Both D_1 and D_2 are lossy decompositions

<http://www.knowledgegate.in/gate>

5 NF/Project-Join Normal Form

- A Relational table R is said to be in 5th normal form if
 - it is in 4 NF
 - it cannot be further non-loss decomposed

Video chapters

- Chapter-1 (Transactions and concurrency control)
- Chapter-2 (ER-model)
- Chapter-3 (Relational model, Functional Dependencies, Keys)
- Chapter-4 (Normalization)
- Chapter-5 (File organization, indexing (e.g., B and B+ trees))
- Chapter-6 (Relational algebra, tuple calculus, SQL)

<http://www.knowledgegate.in/gate>

Indexing

- Theoretically relational database is derived from set theory, and in a set the order of elements in a set is irrelevant, so does in relations (tables). But in practice implementation we have to specify the order.
- A number of properties, like search, insertion and deletion will depend on the order in which elements are stored in the tables.
- There are only two ways in which elements can be stored in a table ordered (Sorted) or unordered (Unsorted).

File organization/ organization of records in a file

Ordered file organization

- All the records in the file are ordered on some search key field.
- Here binary search is possible. (give example of book page searching)
- Maintenance (insertion & deletion) is costly, as it requires reorganization of entire file.
- Notes that we will get binary search only if we are using that key for searching on which indexing is done, otherwise it will behave as unsorted file

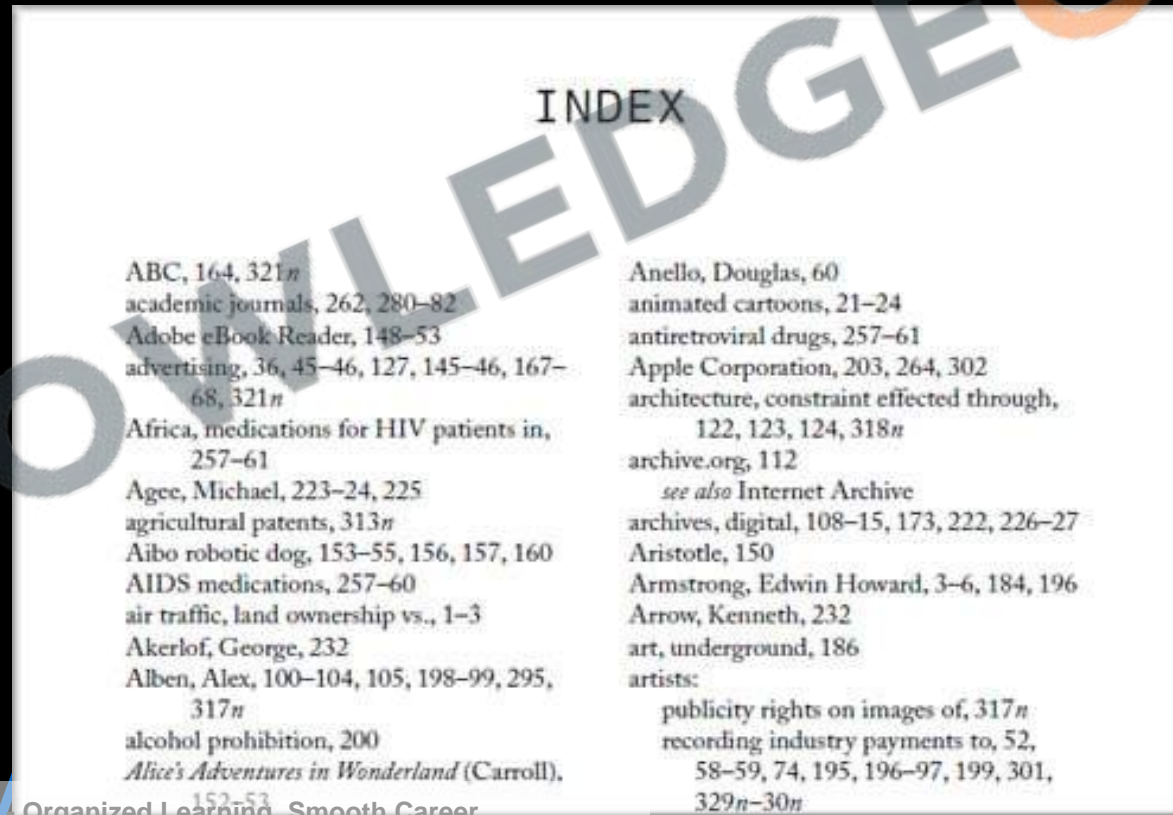


```
-rw-r----- 1 mysql mysql 10M Jul 16 20:10 node__field_meta_tags.ibd
-rw-r----- 1 mysql mysql 10M Jul 15 22:50 comment_field_data.ibd
-rw-r----- 1 mysql mysql 11M Jul 16 20:18 search_total.ibd
-rw-r----- 1 mysql mysql 11M Jul 16 20:21 cache_discovery.ibd
-rw-r----- 1 mysql mysql 12M Jul 16 20:11 node_field_data.ibd
-rw-r----- 1 mysql mysql 14M Jul 16 20:11 url_alias.ibd
-rw-r----- 1 mysql mysql 14M Jul 16 20:11 taxonomy_index.ibd
-rw-r----- 1 mysql mysql 15M Jul 16 19:27 node__tags.ibd
-rw-r----- 1 mysql mysql 15M Jul 15 22:47 comment__comment_body.ibd
-rw-r----- 1 mysql mysql 16M Jul 16 19:27 node_revision__tags.ibd
-rw-r----- 1 mysql mysql 19M Jul 16 20:23 sessions.ibd
-rw-r----- 1 mysql mysql 22M Jul 16 20:18 search_dataset.ibd
-rw-r----- 1 mysql mysql 31M Jul 16 20:11 node__body.ibd
-rw-r----- 1 mysql mysql 32M Jul 16 20:24 watchdog.ibd
-rw-r----- 1 mysql mysql 36M Jul 16 20:11 node_revision__body.ibd
-rw-r----- 1 mysql mysql 108M Jul 16 20:18 search_index.ibd
-rw-r----- 1 mysql mysql 120M Jul 16 20:21 cache_entity.ibd
-rw-r----- 1 mysql mysql 268M Jul 16 20:25 cache_data.ibd
-rw-r----- 1 mysql mysql 1.6G Jul 16 20:25 cache_dynamic_page_cache.ibd
-rw-r----- 1 mysql mysql 3.4G Jul 16 20:25 cache_render.ibd
```

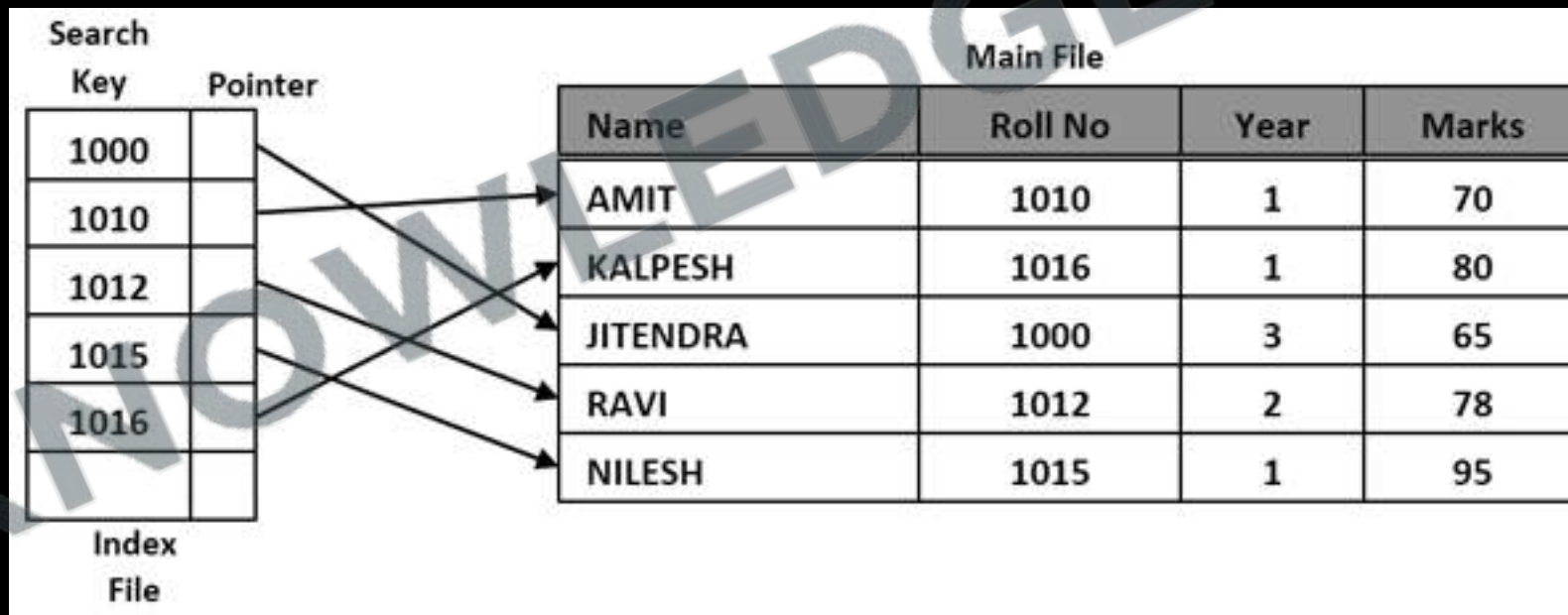

Unordered file organization

- All the records are inserted usually in the end of the file so not ordered according to any field, Because of this only linear search is possible, searching is slow.
- Maintenance (insertion & deletion) is easy, as it does not require re organization of entire file.

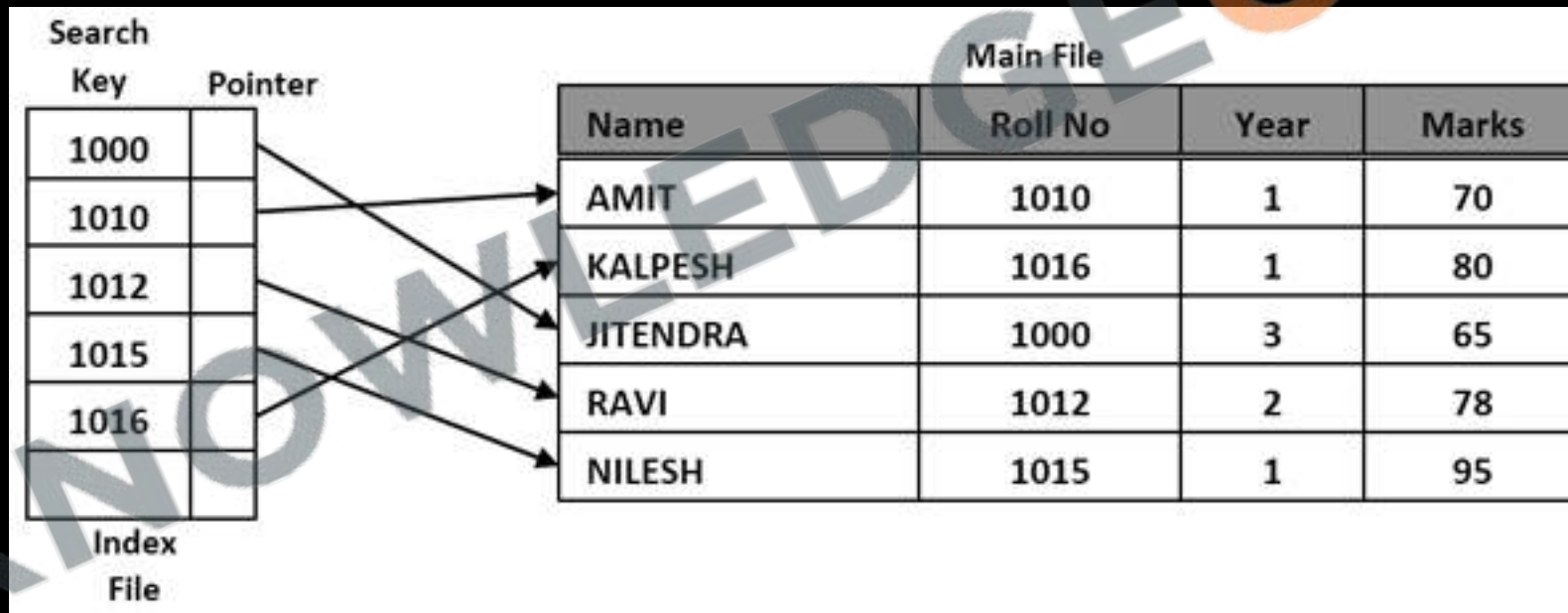
- Reason for indexing - For a large file when it contains a large number of records which will eventually acquire large number of blocks, then its access will become slow.
- Additional auxiliary access structure is called indexes, a data technique to efficiently retrieve records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.



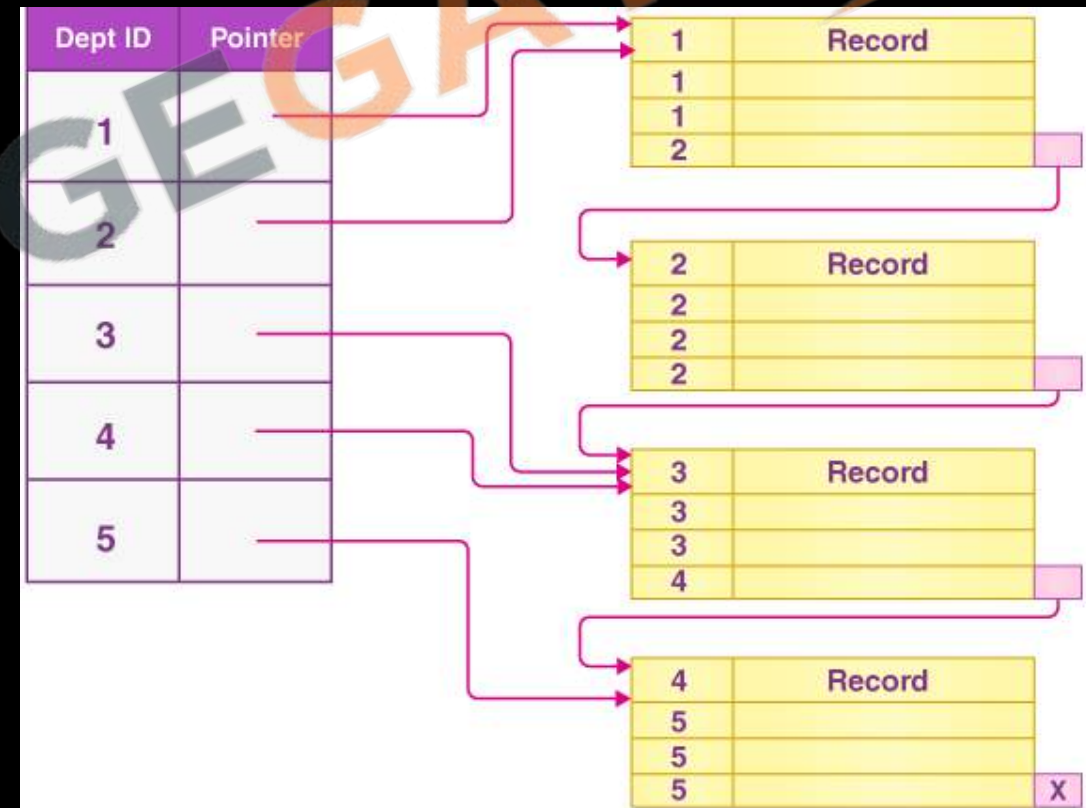
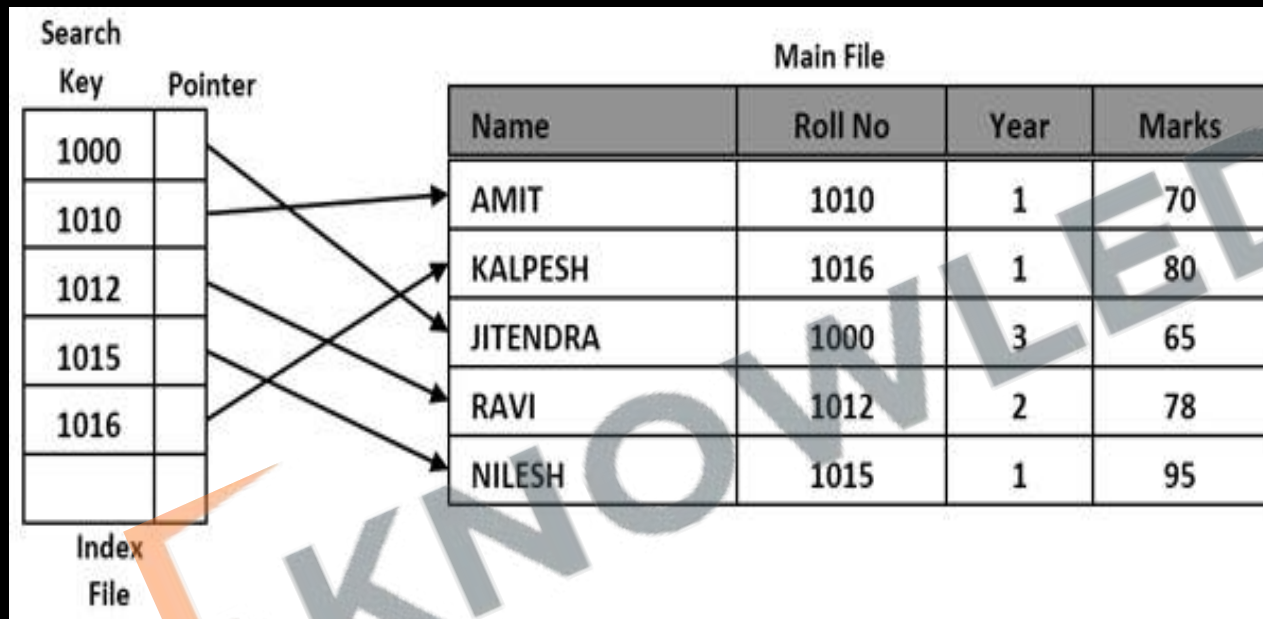
- Index typically provides secondary access path, which provide alternative way to access the records without affecting the physical placement of records in the main file.
- The size of index file is way smaller than that of the main file, as index file record contain only two columns key (attribute in which searching is done) and block pointer (base address of the block of main file which contains the record holding the key), while main file contains all the columns.



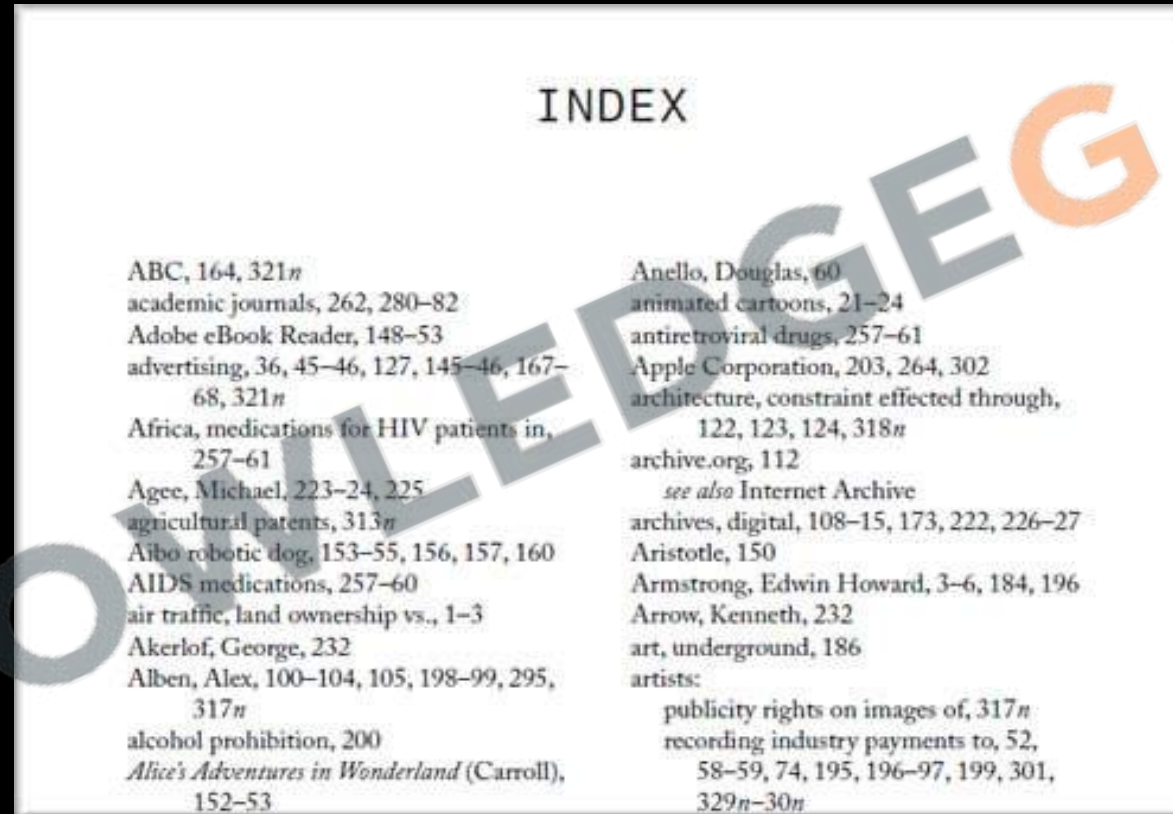
- Index can be created on any field of relation (primary key, non-key)
- One index file is designed according to an attribute, means more than one index file can be designed for a main file.



- Index file is always ordered, irrespective of whether main file is ordered or unordered. So that we can take the advantage of binary search.
- Indexing gives the advantage of faster time, but space taken by index file will be an overhead.



- Number of access required to search the correct block of main file is $\log_2(\text{number of blocks in index file}) + 1$



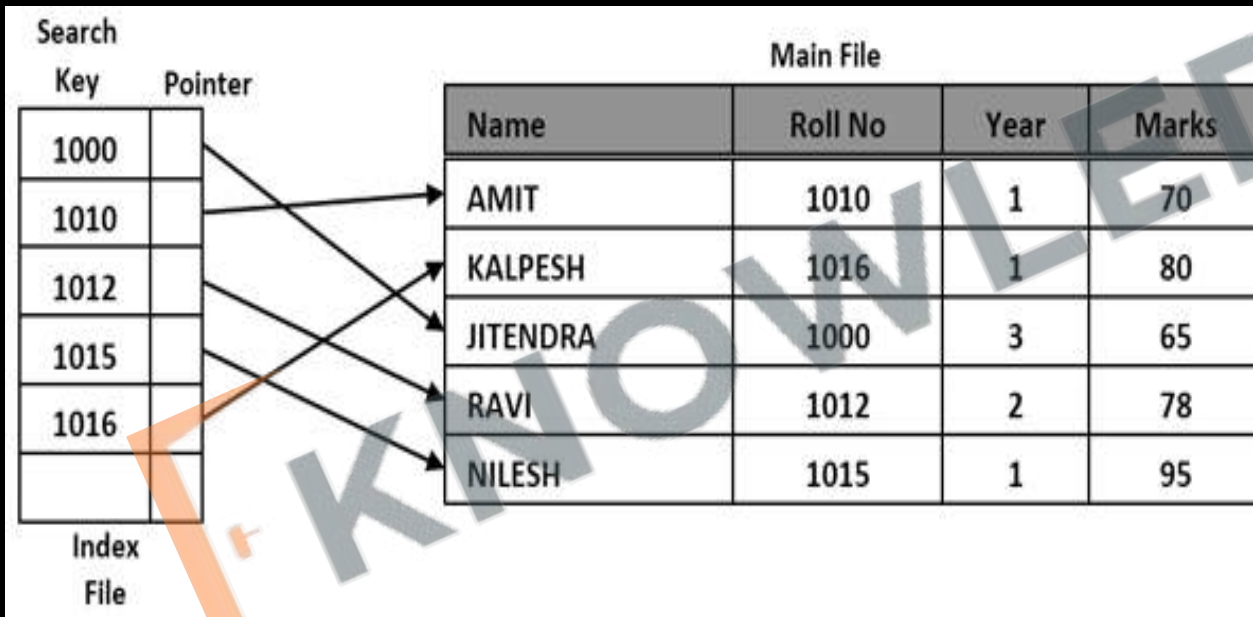
Indexing can be classified on number of criteria's one of them could be

- Dense Index
- Sparse Index

<http://www.knowledgegate.in/gate>

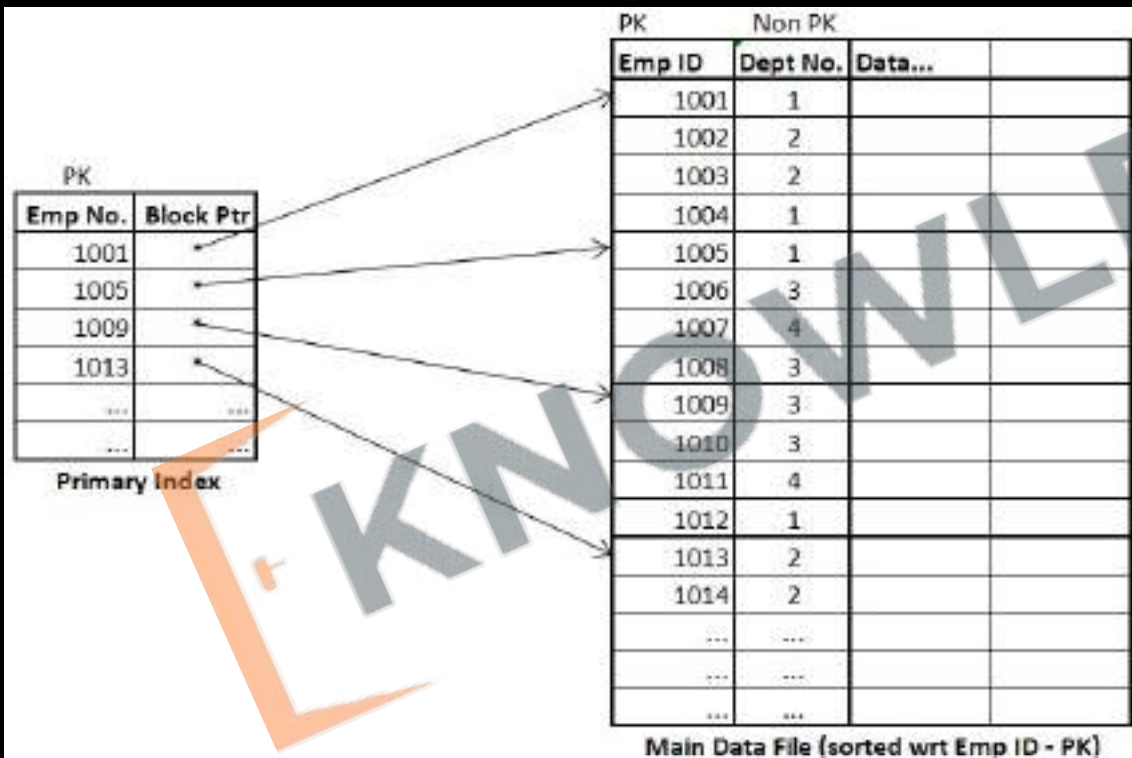
DENSE INDEX

- In dense index, there is an entry in the index file for every search key value in the main file. This makes searching faster but requires more space to store index records itself.
- Note that it is not for every record, it is for every search key value. Sometime number of records in the main file > number of search keys in the main file, for example if search key is repeated.



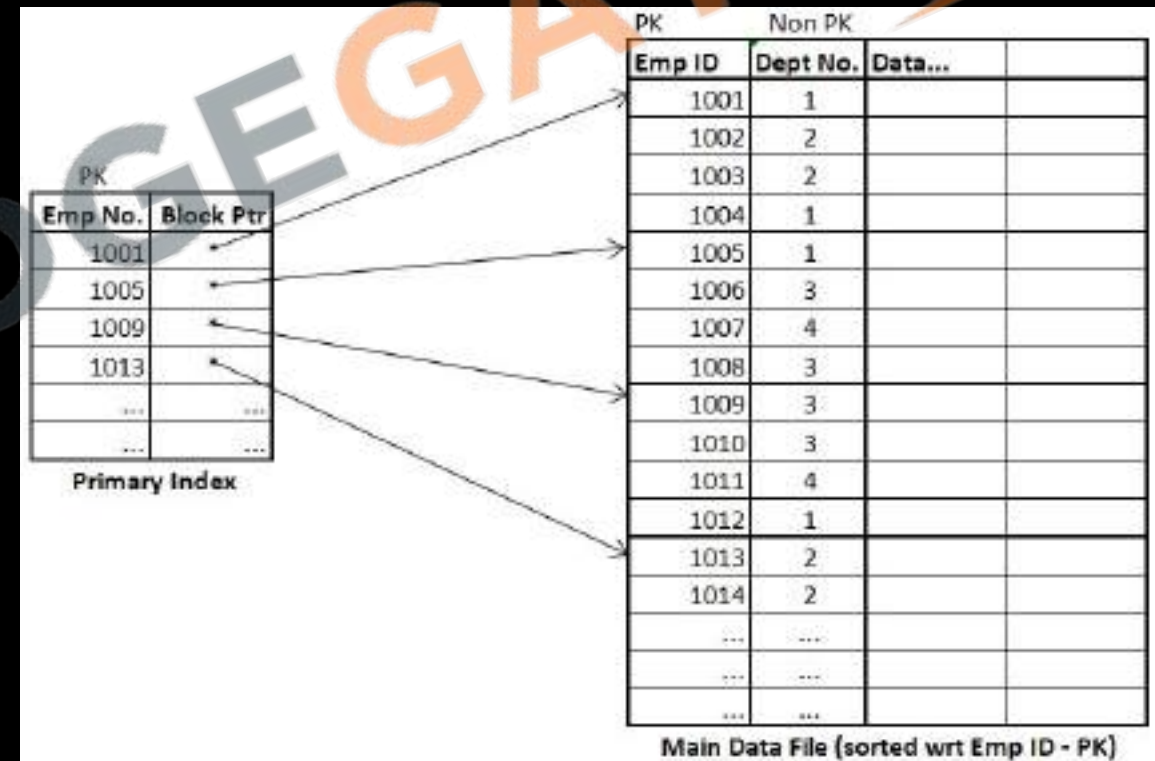
SPARSE INDEX

- If an index entry is created only for some records of the main file, then it is called sparse index.
- No. of index entries in the index file < No. of records in the main file.
- Note: - dense and sparse are not complementary to each other, sometimes it is possible that a record is both dense and sparse.



Basic term used in Indexing

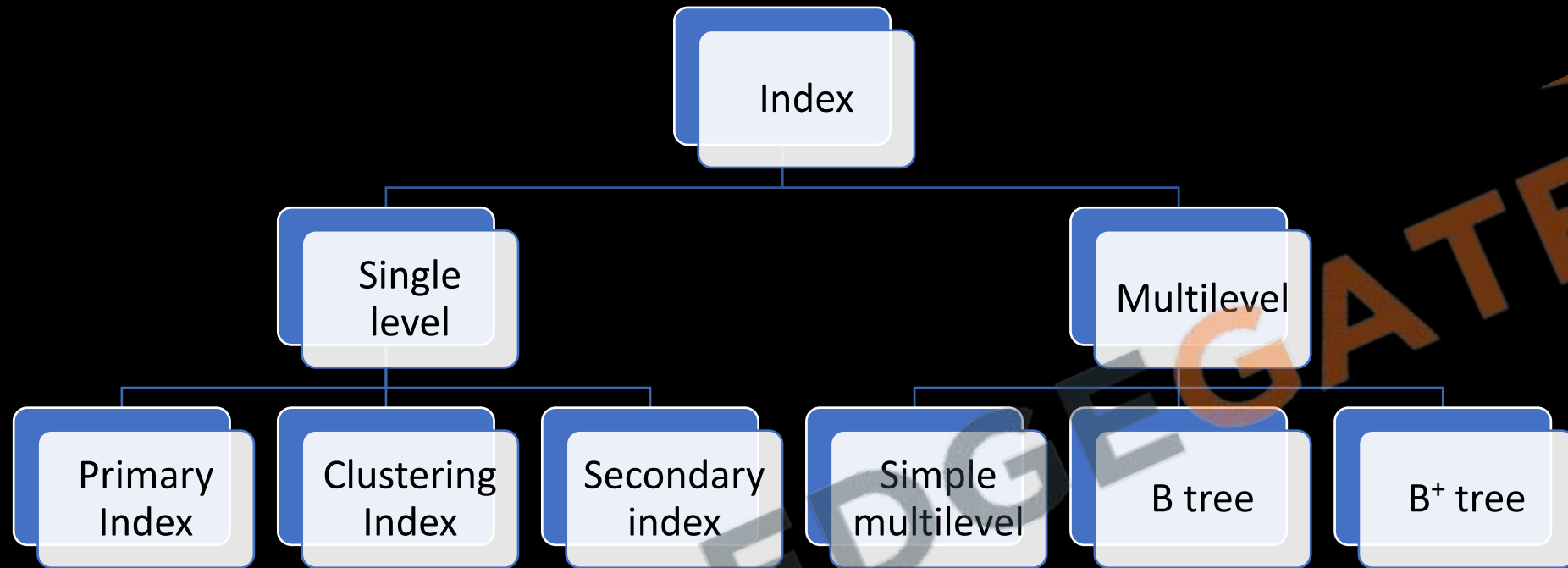
- BLOCKING FACTOR = No. of Records per block = $\lceil \text{block size} / \text{record size} \rceil$
- No of blocks required by file = $\lceil \text{no of records} / \text{blocking factor} \rceil$



- If file is unordered then no of block assesses required to reach correct block which contain the desired record is $O(n)$, where n is the number of blocks.
- if file is ordered then no of block assesses required to reach correct block which contain the desired record is $O(\log_2 n)$, where n is the number of blocks.

<http://www.knowledgegate.in/gate>

TYPES OF INDEXING



PRIMARY INDEXING

- Main file is always sorted according to primary key.
- Indexing is done on Primary Key, therefore called as primary indexing
- Index file have two columns, first primary key and second anchor pointer (base address of block)

- It is an example of Sparse Indexing.
- Here first record (anchor record) of every block gets an entry in the index file
- No. of entries in the index file = No of blocks acquired by the main file.

<http://www.knowledgegate.in/gate>

Q Suppose we have ordered file with records stored $r = 30,000$ on a disk with Block Size $B = 1024$ B. File records are of fixed size and are unspanned with record length $R = 100$ B. Suppose that ordering key field of file is 9 B long and a block pointer is 6 B long, Implement primary indexing?

<http://www.knowledgegate.in/gate>

CLUSTERED INDEXING

- Main file will be ordered on some non-key attributes
- No of entries in the index file = no of unique values of the attribute on which indexing is done.
- It is the example of Sparse as well as dense indexing



Q An index is clustered, if **(GATE-2013) (1 Marks)**

(a) it is on a set of fields that form a candidate key

(b) it is on a set of fields that include the primary key

(c) the data records of the file are organized in the same order as the data entries of the index

(d) the data records of the file are organized not in the same order as the data entries of the index

<http://www.knowledgegate.in/gate>

Q A clustering index is defined on the fields which are of type (GATE-2008) (1 Marks)

a) non-key and ordering

b) non-key and non-ordering

c) key and ordering

d) key and non-ordering

<http://www.knowledgegate.in/gate>

SECONDARY INDEXING

- Most common scenarios, suppose that we already have a primary indexing on primary key, but there is frequent query on some other attributes, so we may decide to have one more index file with some other attribute.
- Main file is ordered according to the attribute on which indexing is done (unordered).

- Secondary indexing can be done on key or non-key attribute.
- No of entries in the index file is same as the number of entries in the main file.
- It is an example of dense indexing.

<http://www.knowledgegate.in/gate>

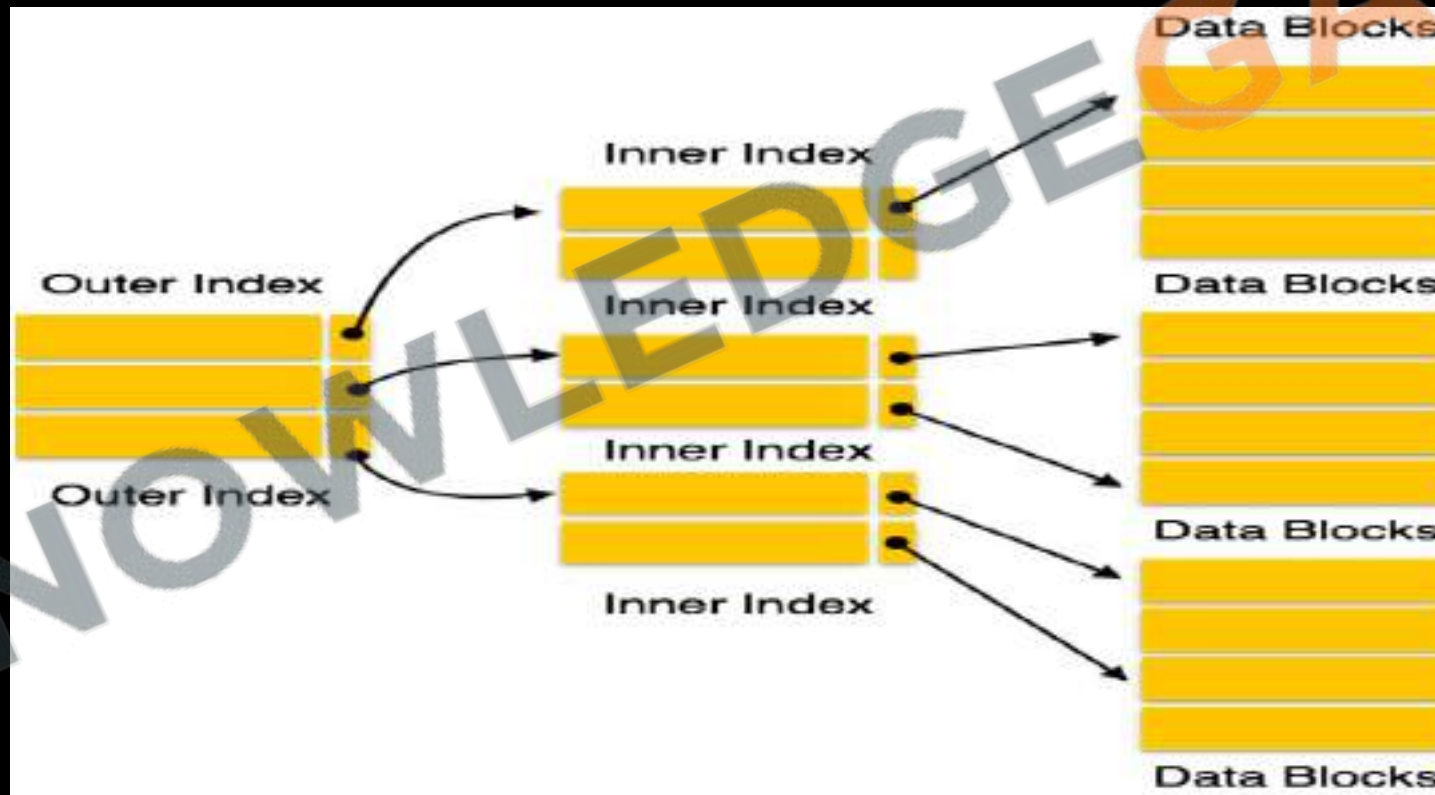
Q Suppose we have ordered file with records stored $r = 30,000$ on a disk with Block Size $B = 1024$ B. File records are of fixed size and are unspanned with record length $R = 100$ B. Suppose that ordering key field of file is 9 B long and a block pointer is 6 B long, Implement Secondary indexing?

<http://www.knowledgegate.in/gate>

Q A data file consisting of 1,50,000 student-records is stored on a hard disk with block size of 4096 bytes. The data file is sorted on the primary key RollNo. The size of a record pointer for this disk is 7 bytes. Each student-record has a candidate key attribute called ANum of size 12 bytes. Suppose an index file with records consisting of two fields, ANum value and the record pointer the corresponding student record, is built and stored on the same disk. Assume that the records of data file and index file are not split across disk blocks. The number of blocks in the index file is _____. **.(GATE 2021) (1 MARKS)**

MULTILEVEL INDEXING

- Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block-0, which can easily be accommodated anywhere in the main memory.



Q Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is ordered on a non-key field, and the file organization is unpanned. The file is stored in a file system with block size 1024 bytes, and the size of a block pointer is 10 bytes. If the secondary index is built on the key field of the file, and a multi-level index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multi-level index are respectively **(GATE-2008) (1 Marks)**

a) 8 and 0

b) 128 and 6

c) 256 and 4

d) 512 and 5

<http://www.knowledgegate.in/gate>

Reason to have B tree and B+ tree

- After studying indexing in detail now we understand that an index file is always sorted in nature and will be searched frequently, and sometimes index files can be so large that even we want to index the index file (Multilevel index), therefore we must search best data structure to meet our requirements.
- There are number of options in data structure like array, stack, link list, graph, table etc. but we want a data structure which support frequent insertion deletion, and modify it self accordingly but at the same time also provide speed search and give us the advantage of having a sorted data.

- If we look at the data structures option then tree seem to be the most appropriate but every kind of tree in the available option have some problems either simple tree or binary search tree or AVL tree, so we end up on designing new data structure called B-tree which are kind of specially designed for sorted stored index files in databases.
- In general, with multilevel indexing, we require dynamic structure, b and b⁺ tree is generalized implementation of multilevel indexing, which are dynamic in nature, that is increasing and decreasing number of records. In the first level index file can be easily supported by other level index.
- B tree and B⁺ tree also provides efficient search time, as the height of the structure is very less and they are also perfectly balanced.

B tree

- A B-tree of order m if non-empty is an m -way search tree in which.
 - The root has at least zero child nodes and at most m child nodes.
 - The internal nodes except the root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes.
 - The number of keys in each internal node is one less than the number of child nodes and these keys partition the subtrees of the nodes in a manner similar to that of m -way search tree.
 - All leaf nodes are on the same level(perfectly balanced).

Root

Rules	MAX	MIN
CHILD	m	0
DATA	$m-1$	1

Internal except Root

Rules	MAX	MIN
CHILD	m	$\lceil m/2 \rceil$
DATA	$m-1$	$\lceil m/2 \rceil - 1$

Leaf

Rules	MAX	MIN
CHILD	0	0
DATA	$m-1$	$\lceil m/2 \rceil - 1$

Insertion in B-TREE

- A B-tree starts with a single root node (which is also a leaf node) at level 0 (zero). Once the root node is full with $m - 1$ search key values and we attempt to insert another entry in the tree, the root node splits into two nodes at level 1.
- Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes. When a non-root node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes.
- If the parent node is full, it is also split. Splitting can propagate all the way to the root node, creating a new level if the root is split.

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 3, 4
insert them into an empty b-tree of order = 3.

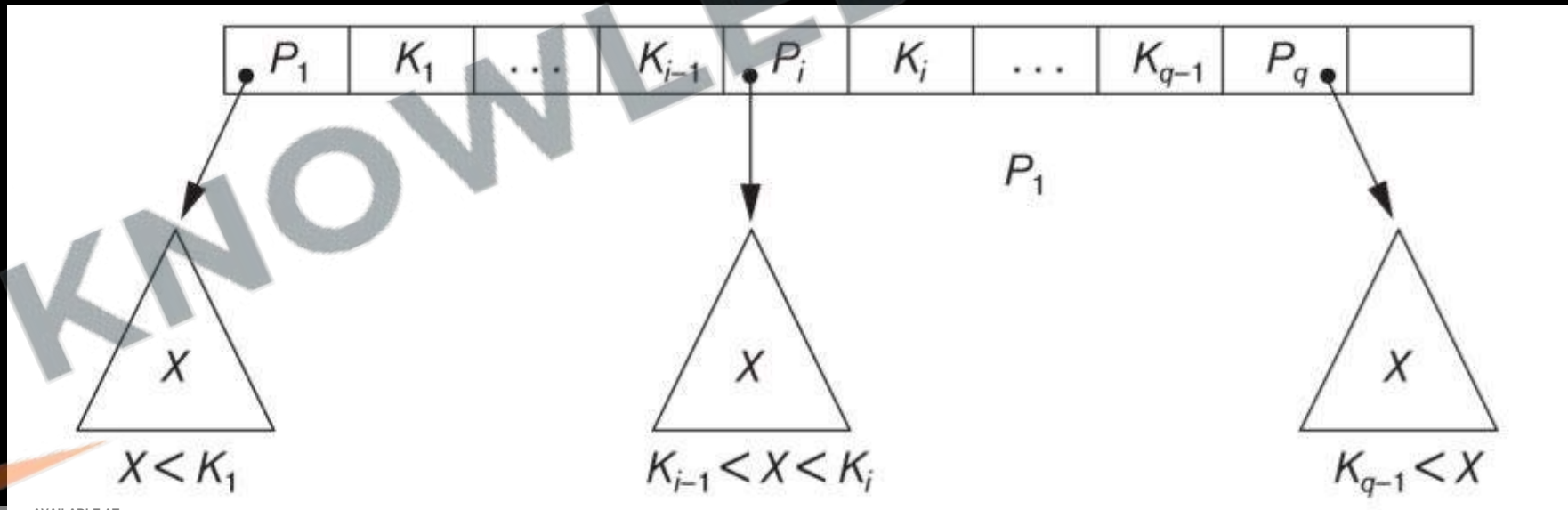
<http://www.knowledgegate.in/gate>

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 4, 20, 18, 19, 17, 16, 15, 25, 23, 24 insert them into an empty b-tree of order = 5.

<http://www.knowledgegate.in/gate>

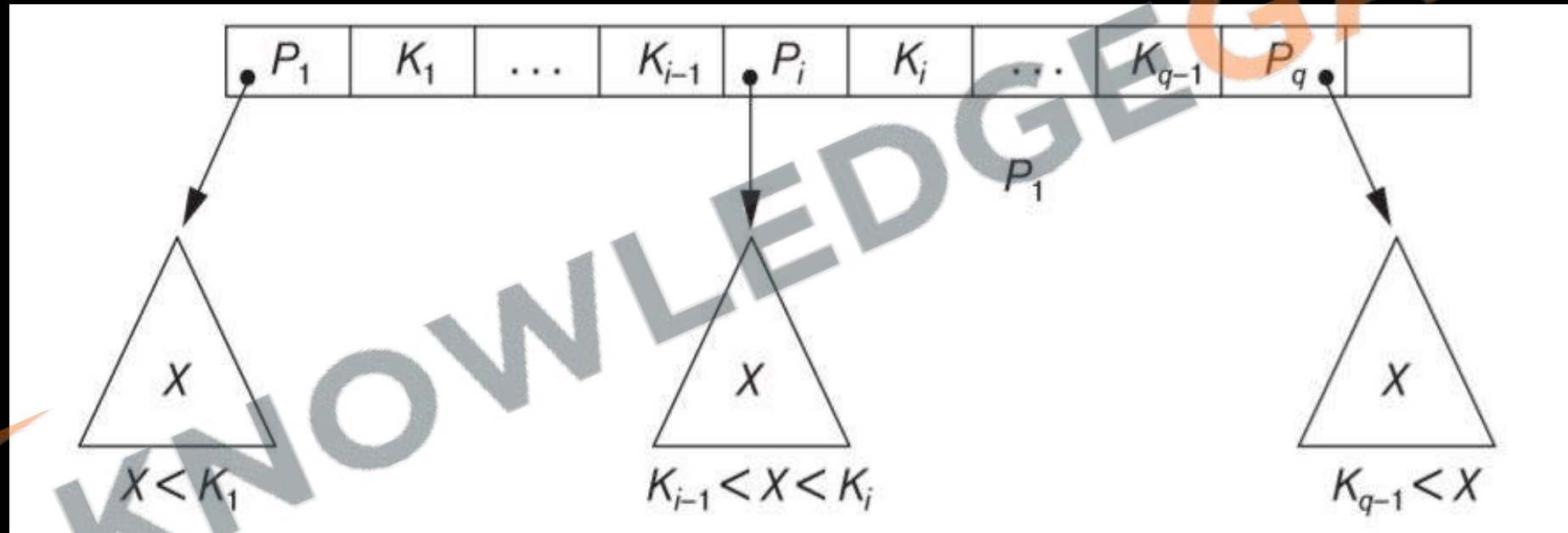
Analysis

- B- TREE- In computer science, a B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time.
- A **search tree of order p** is a tree such that each node contains *at most* $p - 1$ search values and p pointers in the order $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$, where $q \leq p$. Each P_i is a pointer to a child node (or a NULL pointer), and each K_i is a search value from some ordered set of values. All search values are assumed to be unique.



AVAILABLE AT:

- Two constraints must hold at all times on the search tree:
 - Within each node, $K_1 < K_2 < \dots < K_{q-1}$.
 - For all values X in the subtree pointed at by P_i , we have $K_{i-1} < X < K_i$ for $1 < i < q$; $X < K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$.



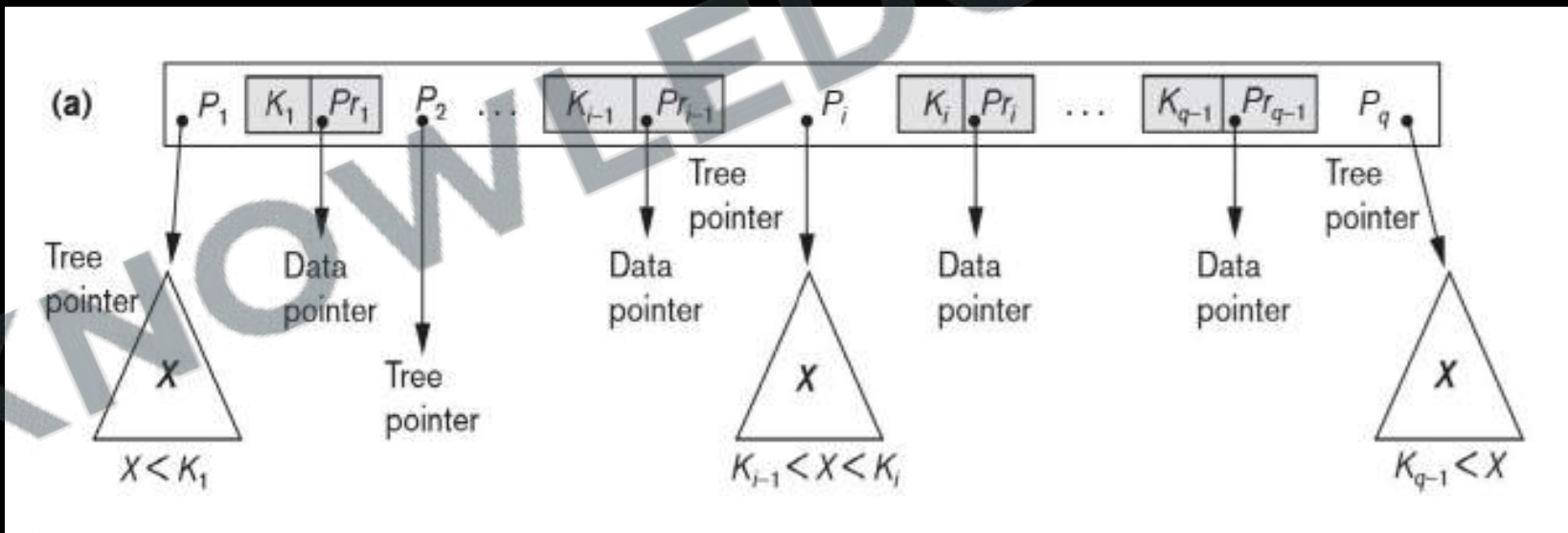
- We can use a search tree as a mechanism to search for records stored in a disk file. The values in the tree can be the values of one of the fields of the file, called the **search field** (which is the same as the index field if a multilevel index guides the search).
- Each key value in the tree is associated with a pointer to the record in the data file having that value.
- To guarantee that nodes are evenly distributed, so that the depth of the tree is minimized for the given set of keys and that the tree does not get skewed with some nodes being at very deep levels.

- To make the search speed uniform, so that the average time to find any random key is roughly the same
- While minimizing the number of levels in the tree is one goal, another implicit goal is to make sure that the index tree does not need too much restructuring as records are inserted into and deleted from the main file.
- Thus, we want the nodes to be as full as possible and do not want any nodes to be empty if there are too many deletions. Record deletion may leave some nodes in the tree nearly empty, thus wasting storage space and increasing the number of levels.
- The B-tree addresses both of these problems by specifying additional constraints on the search tree.

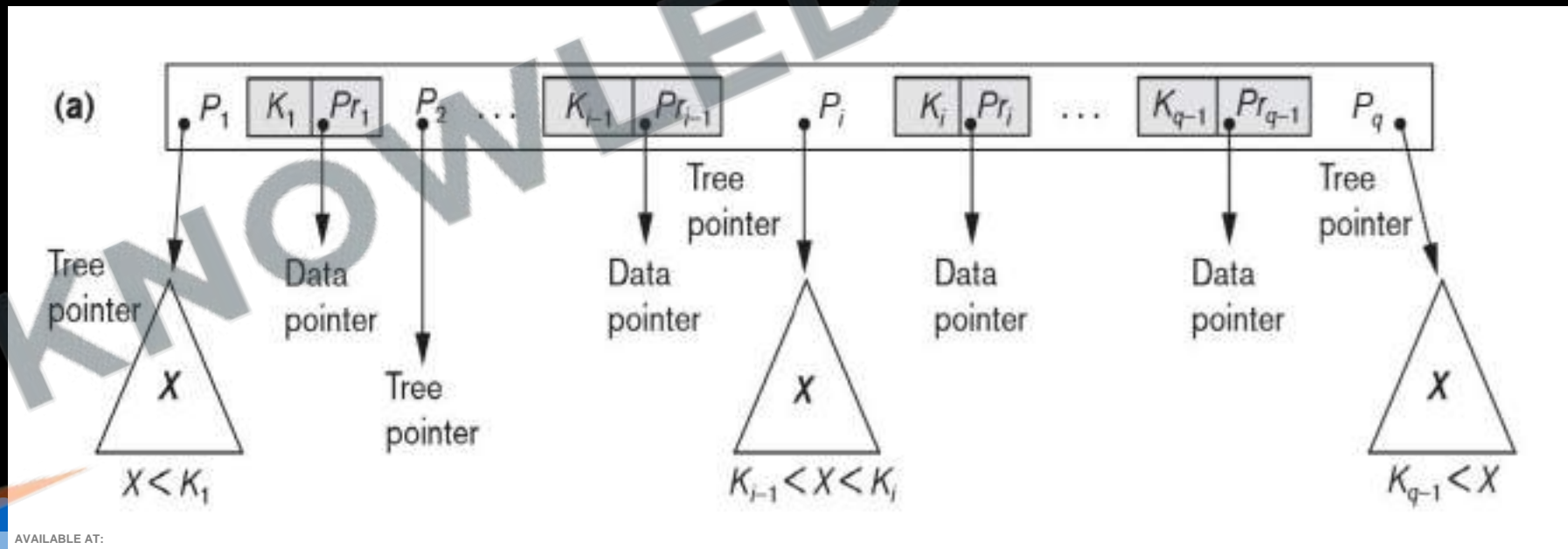
- The B-tree has additional constraints that ensure that the tree is always balanced and that the space wasted by deletion, if any, never becomes excessive.
- The algorithms for insertion and deletion, though, become more complex in order to maintain these constraints. Nonetheless, most insertions and deletions are simple processes; they become complicated only under special circumstances—namely, whenever we attempt an insertion into a node that is already full or a deletion from a node that makes it less than half full.
- More formally, a **B-tree of order p** , when used as an access structure on a *key field* to search for records in a data file, can be defined as follows:

<http://www.knowledgegate.in/gate>

- Each internal node in the B-tree is of the form
 - $\langle P_1, \langle K_1, P_{r1} \rangle, P_2, \langle K_2, P_{r2} \rangle, \dots, \langle K_{q-1}, P_{rq-1} \rangle, P_q \rangle$ where $q \leq p$. Each P_i is a **tree pointer**—a pointer to another node in the B-tree. Each P_{ri} is a **data pointer**—a pointer to the record whose search key field value is equal to K_i (or to the data file block containing that record).
- Within each node, $K_1 < K_2 < \dots < K_{q-1}$.
- For all search key field values X in the subtree pointed at by P_i (the i th sub-tree), we have: $K_{i-1} < X < K_i$ for $1 < i < q$; $X < K_1$ for $i = 1$; and $K_{q-1} < X$ for $i = q$.



- Each node has at most p tree pointers.
- Each node, except the root and leaf nodes, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers unless it is the only node in the tree.
- A node with q tree pointers, $q \leq p$, has $q - 1$ search key field values (and hence has $q - 1$ data pointers).
- All leaf nodes are at the same level. Leaf nodes have the same structure as internal nodes except that all of their *tree pointers* P_i are NULL.



AVAILABLE AT:

Conclusion

- Very less internal fragmentation, memory utilization is very good.
- Less number of nodes(blocks) are used and height is also optimized, so access will be very fast.
- Difficulty of traversing the key sequentially. Means B-TREE do not hold good for range-based queries of database.

B⁺ Tree

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 3, 4 insert them into an empty b⁺ tree of order = 3.

<http://www.knowledgegate.in/gate>

B⁺ Tree

Q Consider the following elements 5, 10, 12, 13, 14, 1, 2, 4, 20, 18, 19, 17, 16, 15, 25, 23, 24, 22 insert them into an empty b⁺ tree of order = 5

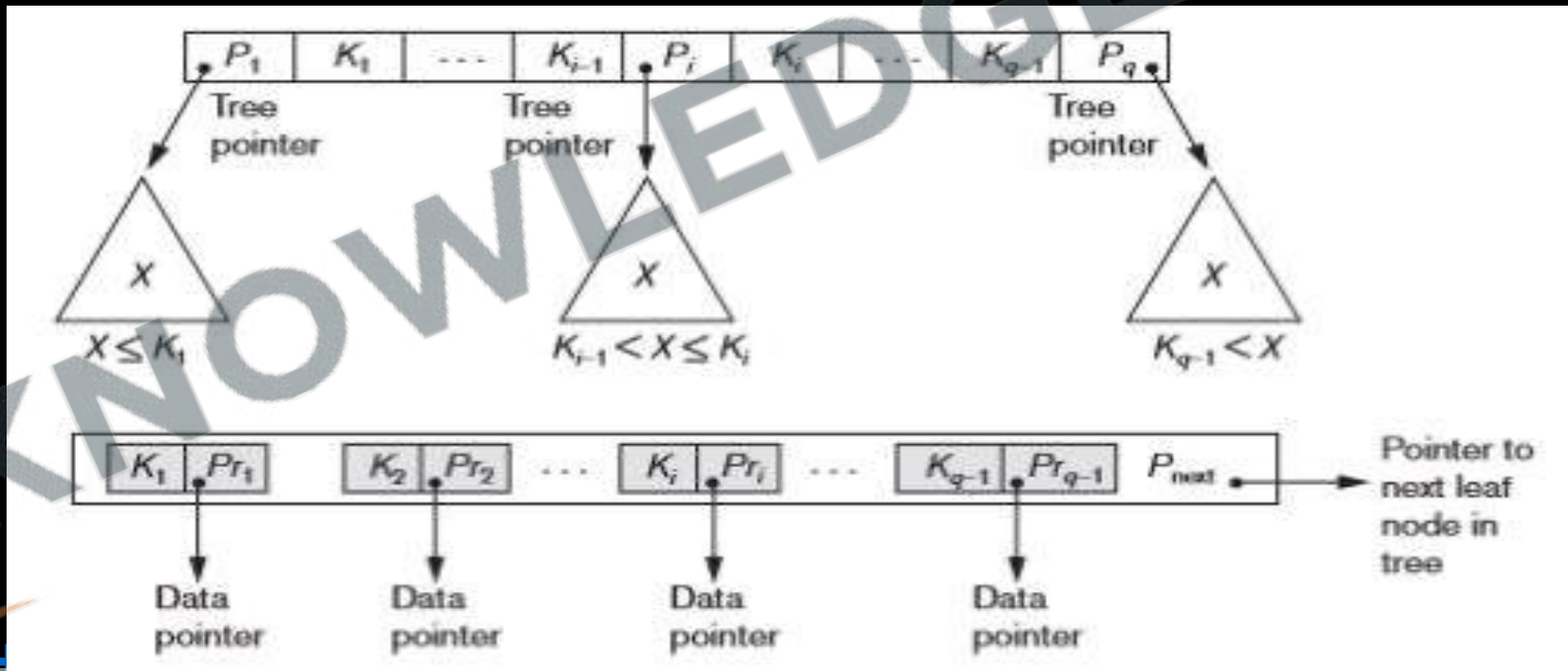
<http://www.knowledgegate.in/gate>

Insertion in B⁺ Tree

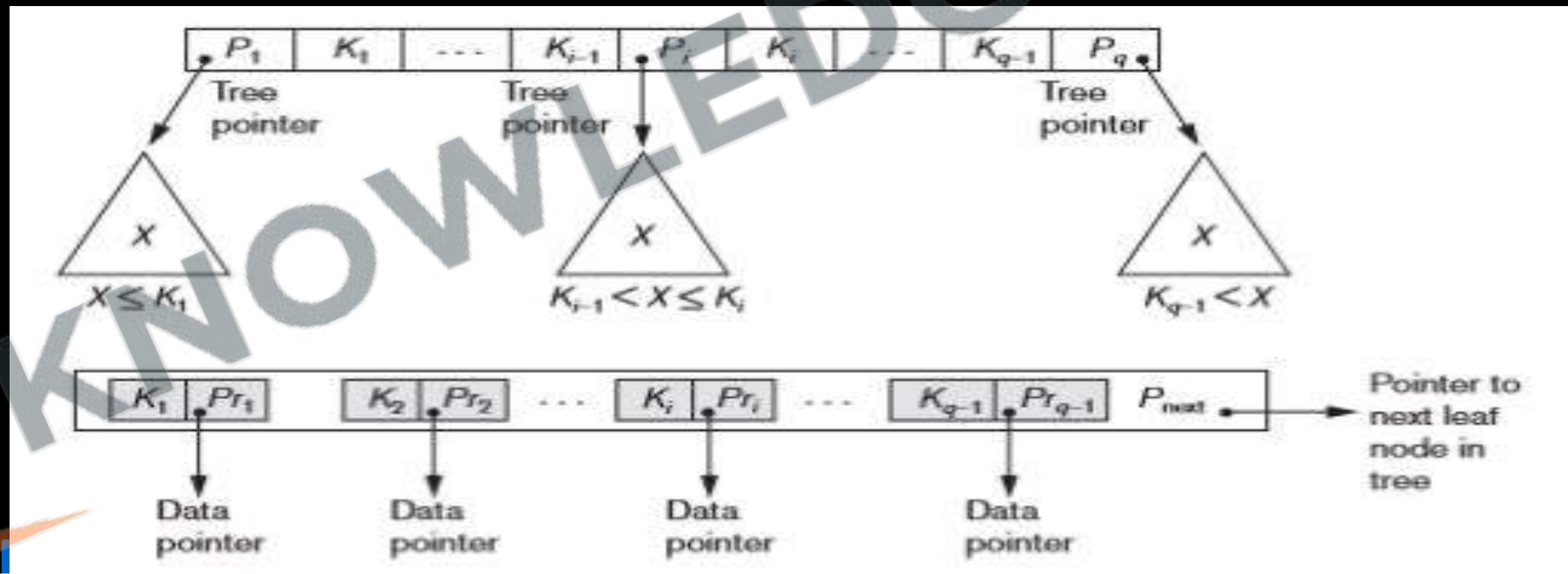
- Start from root node and proceed towards leaf using the logic of binary search tree. Value is inserted in the leaf.
- If overflow condition occurs pick the median and push it into the parent node. Also copy the median or key inserted in parent node to the left or right child node.
- Repeat this procedure until tree is maintained.

Analysis

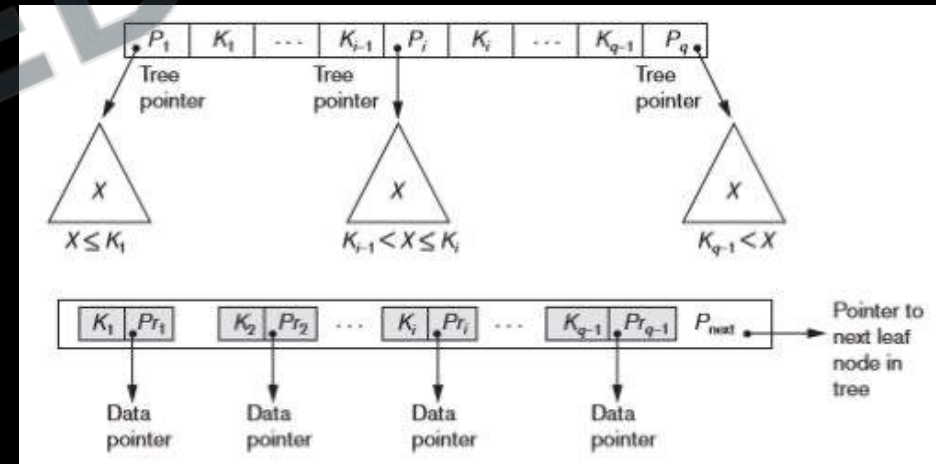
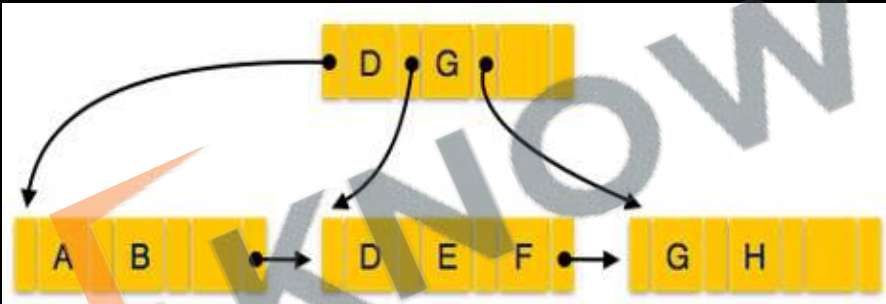
- Most implementations of a dynamic multilevel index use a variation of the B-tree data structure called a **B+-tree**. In a B-tree, every value of the search field appears once at some level in the tree, along with a data pointer.
- In a B+-tree, data pointers are stored *only at the leaf nodes* of the tree; hence, the structure of leaf nodes differs from the structure of internal nodes.
- The leaf nodes have an entry for *every* value of the search field, along with a data pointer to the record (or to the block that contains this record) if the search field is a key field.



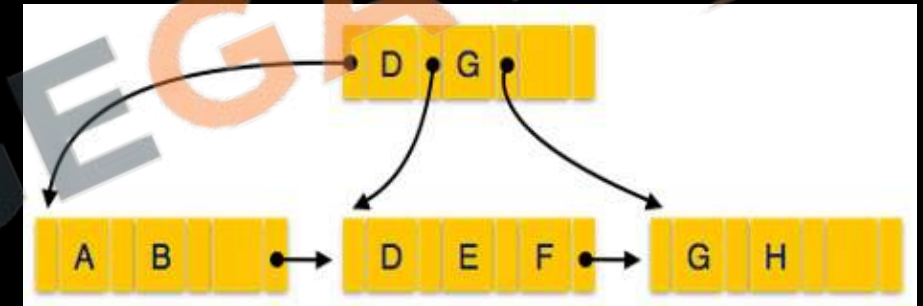
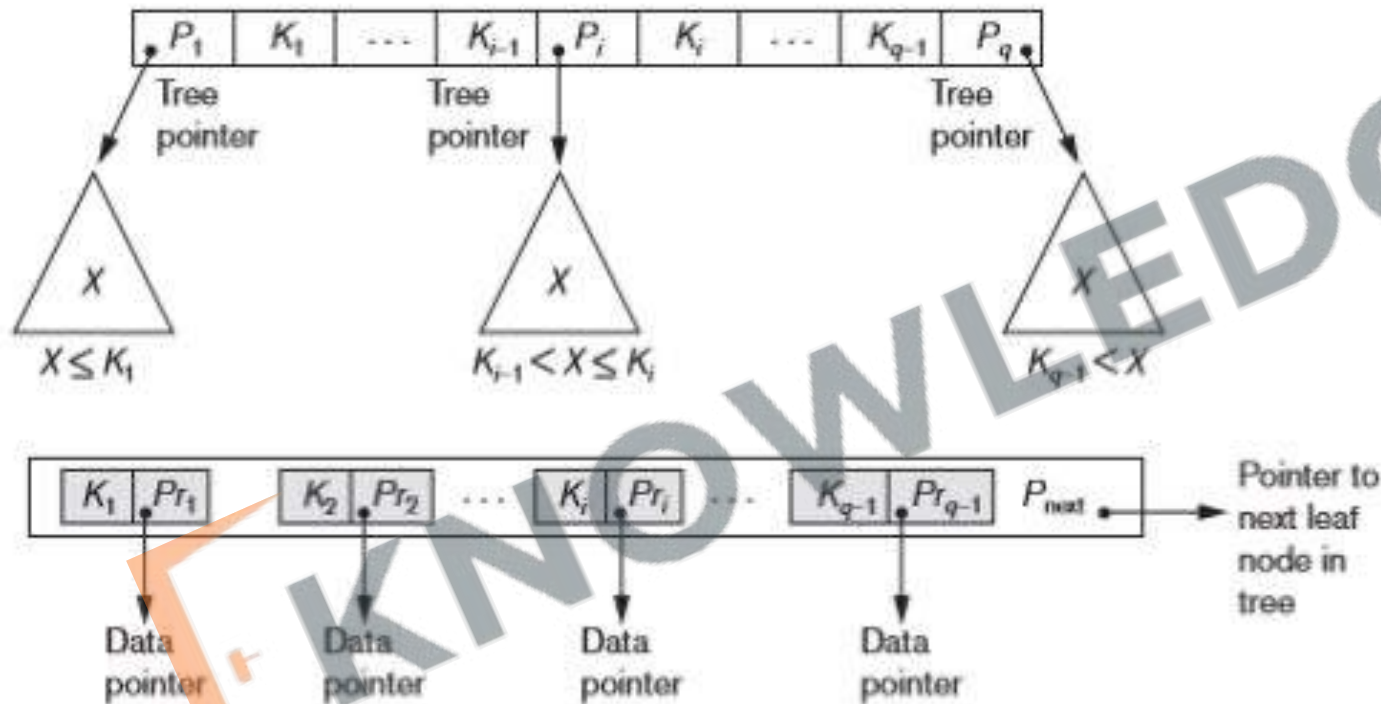
- The leaf nodes of the B+-tree are usually linked to provide ordered access on the search field to the records.
- Each internal node is of the form $\langle P_1, K_1, P_2, K_2, \dots, P_{q-1}, K_{q-1}, P_q \rangle$
- Within each internal node, $K_1 < K_2 < \dots < K_{q-1}$.
- For all search field values X in the subtree pointed at by P_i , we have $K_{i-1} < X \leq K_i$ for $1 < i < q$; $X \leq K_i$ for $i = 1$; and $K_{i-1} < X$ for $i = q$.



- Each internal node has at most p tree pointers.
- Each internal node, except the root, has at least $\lceil (p/2) \rceil$ tree pointers. The root node has at least two tree pointers if it is an internal node.
- An internal node with q pointers, $q \leq p$, has $q - 1$ search field values.
- The structure of the *leaf nodes* of a B+-tree of order p is as follows:
- Each leaf node is of the form $\langle \langle K_1, Pr_1 \rangle, \langle K_2, Pr_2 \rangle, \dots, \langle K_{q-1}, Pr_{q-1} \rangle, P_{\text{next}} \rangle$ where $q \leq p$, each Pr_i is a data pointer, and P_{next} points to the next *leaf node* of the B+-tree.



- Within each leaf node, $K_1 \leq K_2 \dots, K_{q-1}$, $q \leq p$.
- Each P_{r_i} is a **data pointer** that points to the record whose search field value is K_i or to a file block containing the record (or to a block of record pointers that point to records whose search field value is K_i if the search field is not a key).
- Each leaf node has at least $\lceil (p/2) \rceil$ values.
- All leaf nodes are at the same level.



KNOWLEDGEGATE

<http://www.knowledgegate.in/gate>

The following key values are inserted into a B⁺-tree in which order of the internal nodes is 3, and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B⁺-tree is initially empty.

10, 3, 6, 8, 4, 2, 1

The maximum number of times leaf nodes would get split up as a result of these insertions is

- a) 2 b) 3 c) 4 d) 5

(GATE-2009) (2 Marks)

<http://www.knowledgegate.in/gate>

Q In a B+ tree, if the search-key value is 8 bytes long, the block size is 512 bytes and the block pointer is 2 bytes, then the maximum order of the B+ tree is. (GATE-2017) (2 Marks)

<http://www.knowledgegate.in/gate>

Q Consider B+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is **(Gate-2015) (2 Marks)**

<http://www.knowledgegate.in/gate>

Q The order of a leaf node in a B⁺- tree is the maximum number of (value, data record pointer) pairs it can hold. Given that the block size is 1K bytes, data record pointer is 7 bytes long, the value field is 9 bytes long and a block pointer is 6 bytes long, what is the order of the leaf node?
(GATE-2007) (1 Marks)

a) 63

b) 64

c) 67

d) 68

<http://www.knowledgegate.in/gate>

Q Which one of the following statements is NOT correct about the B⁺ tree data structure used for creating an index of a relational database table? **(GATE-2019) (1 Marks)**

(a) Each leaf node has a pointer to the next leaf node

(b) Non-leaf nodes have pointers to data records

(c) B+ Tree is a height-balanced tree

(d) Key values in each node are kept in sorted order

<http://www.knowledgegate.in/gate>

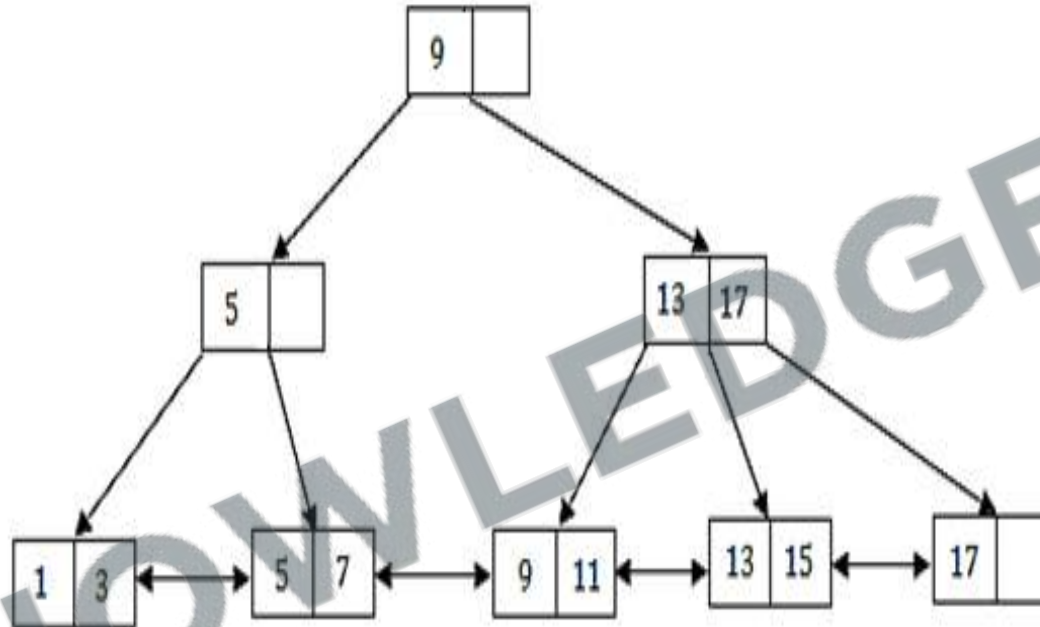
Q B⁺ Trees are considered **BALANCED** because (GATE-2016) (1 Marks)

- a) the lengths of the paths from the root to all leaf nodes are all equal
- b) the lengths of the paths from the root to all leaf nodes differ from each other by at most 1
- c) the number of children of any two non-leaf sibling nodes differ by at most 1
- d) the number of records in any two leaf nodes differ by at most 1

<http://www.knowledgegate.in/gate>

Q (GATE-2015) (1 Marks)

With reference to the B⁺ tree index of order 1 shown below, the minimum number of nodes (including the Root node) that must be fetched in order to satisfy the following query: "Get all records with a search key greater than or equal to 7 and less than 15" is _____.



B tree

- A B-tree of order m if non-empty is an m -way search tree in which.
 - The root has at least zero child nodes and at most m child nodes.
 - The internal nodes except the root have at least $\lceil m/2 \rceil$ child nodes and at most m child nodes.
 - The number of keys in each internal node is one less than the number of child nodes and these keys partition the subtrees of the nodes in a manner similar to that of m -way search tree.
 - All leaf nodes are on the same level(perfectly balanced).

Root

Rules	MAX	MIN
CHILD	m	0
DATA	$m-1$	1

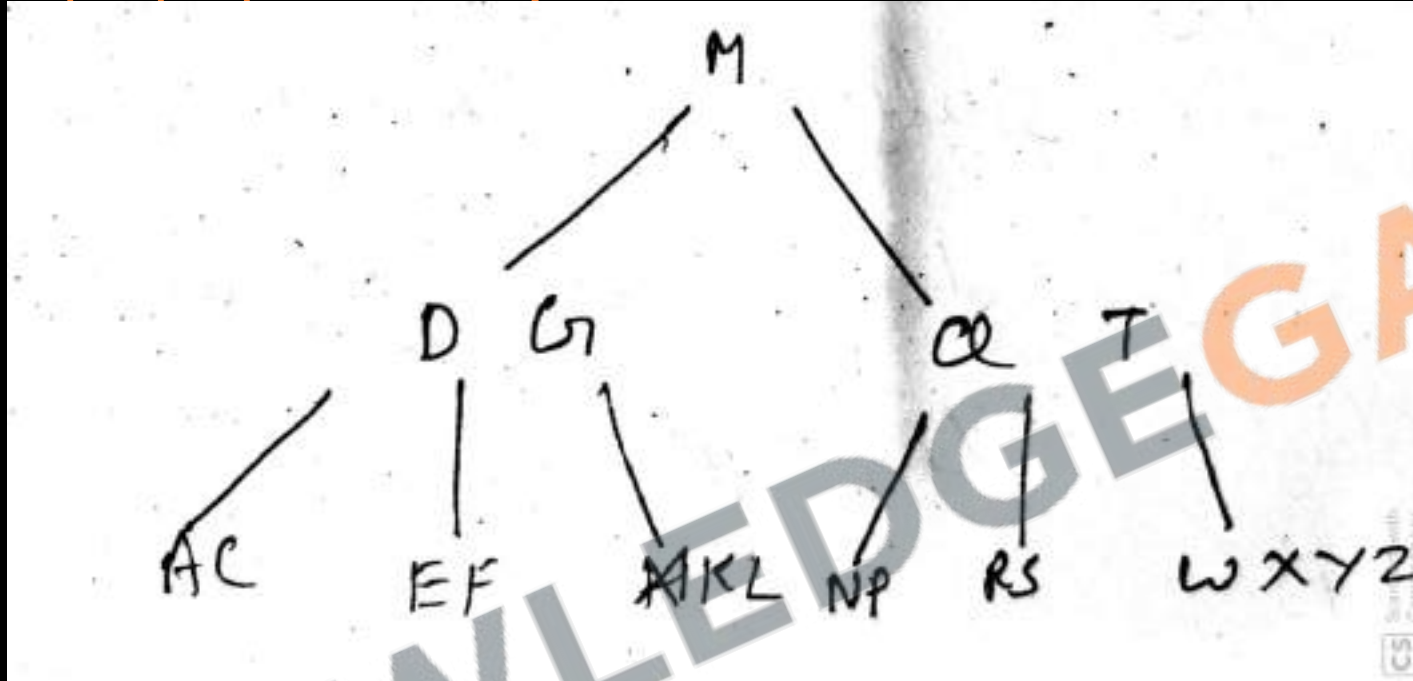
Internal except Root

Rules	MAX	MIN
CHILD	m	$\lceil m/2 \rceil$
DATA	$m-1$	$\lceil m/2 \rceil - 1$

Leaf

Rules	MAX	MIN
CHILD	0	0
DATA	$m-1$	$\lceil m/2 \rceil - 1$

Q Consider the Following B-tree of order $m=6$, delete the following nodes H, T, R, E, A, C, S in sequence?



Deletion in B-TREE

- If the deletion is from the leaf node and leaf node is satisfying the minimal condition even after the deletion, then delete the value directly.
- If deletion from leaf node renders leaf node in minimal condition, then first search the extra key in left sibling and then in the right sibling. Largest value from left sibling or smallest value from right sibling is pushed into the root node and corresponding value can be fetched from parent node to leaf node.
- If the deletion is to be from internal node, then first we check for the extra key in the left and then in the right child. If we find one, we fetch the value in the required node. And delete the key.

- If deletion of a value causes a node to be less than half full, it is combined with its neighboring nodes, and this can also propagate all the way to the root. Hence, deletion can reduce the number of tree levels.
- It has been shown by analysis and simulation that, after numerous random insertions and deletions on a B-tree, the nodes are approximately 69 percent full when the number of values in the tree stabilizes. This is also true of B⁺ trees.
- If this happens, node splitting and combining will occur only rarely, so insertion and deletion become quite efficient. If the number of values grows, the tree will expand without a problem—although splitting of nodes may occur, so some insertions will take more time.

Q Consider the following elements 5, 8, 1, 7, 3, 12, 9, 6 insert them into an empty b^+ tree of order = 3. and then delete following nodes in sequence 9, 8, 12?

<http://www.knowledgegate.in/gate>

Q Consider a B⁺-tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node? (GATE-2010) (1 Marks)

<http://www.knowledgegate.in/gate>

Chapter-6

(Relational algebra, tuple calculus, SQL)

<http://www.knowledgegate.in/gate>

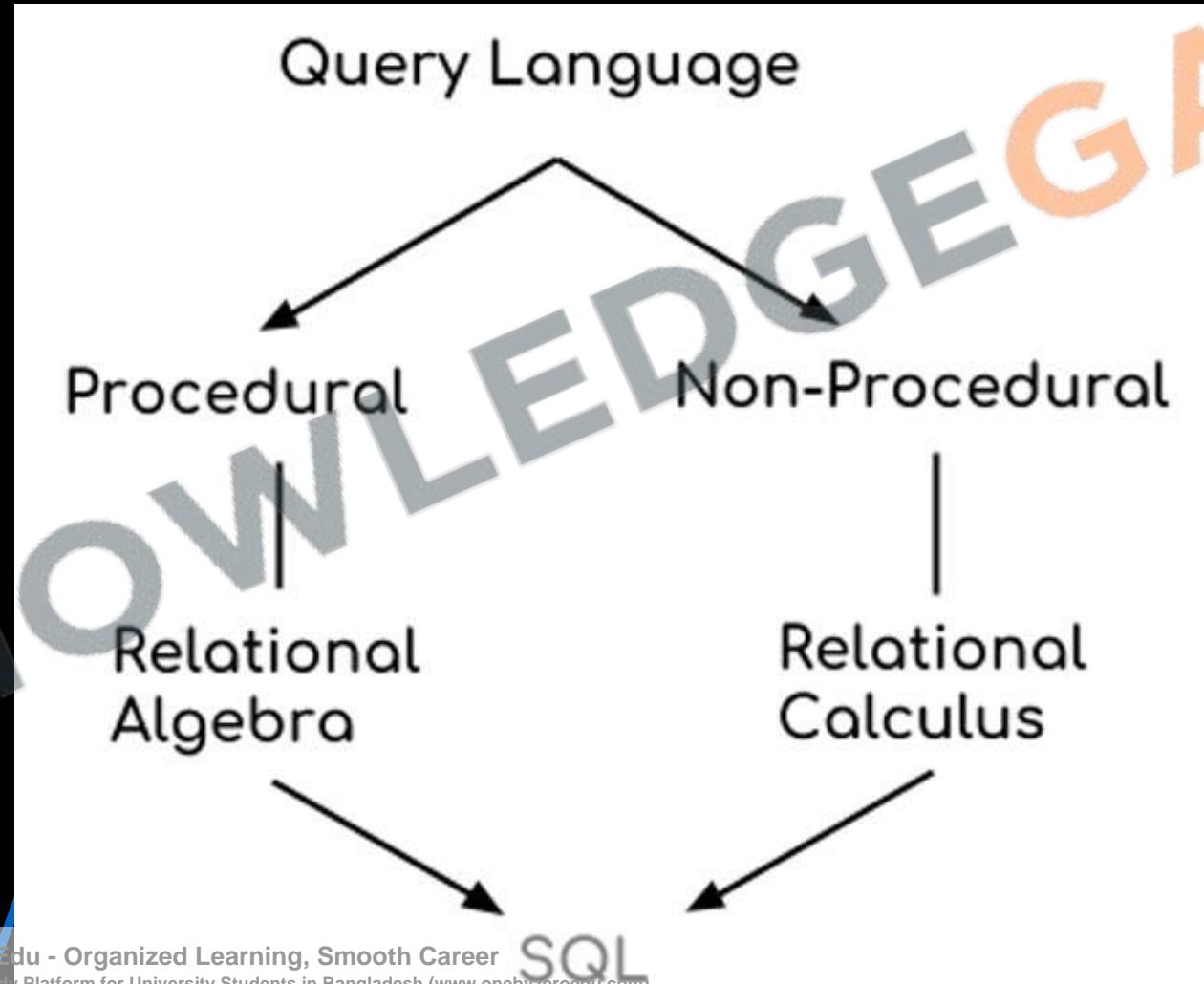
Query Language

- After designing a data base, that is ER diagram followed by conversion in relational model followed by normalization and indexing, now next task is how to store, retrieve and modify data in the data base.
- Thought here we will be concentrating more on the retrieval part. Query languages are used for this purpose.

ER diagram → relational model → Normalization → Indexing →

<http://www.knowledgegate.in/gate>

- **Query languages, data query languages or database query languages (DQLs)** are computer languages using which user request some information from the database. A well known example is the **Structured Query Language (SQL)**.



- **Atomese**, the graph query language for the **OpenCog** graph database, the **AtomSpace**.
- **Attempto Controlled English** is a query language that is also a **controlled natural language**.^[1]
- **AQL** is a query language for the **ArangoDB** native multi-model database system.
- **.QL** is a proprietary object-oriented query language for querying **relational databases**; successor of Datalog;
- **Contextual Query Language** (CQL) a formal language for representing queries to **information retrieval** systems such as web indexes or bibliographic catalogues.
- **CQLF** (CODYASYL Query Language, Flat) is a query language for **CODASYL**-type databases;
- **Concept-Oriented Query Language** (COQL) is used in the concept-oriented model (COM). It is based on a novel **data modeling** construct, concept, and uses such operations as projection and de-projection for multi-dimensional analysis, analytical operations and inference;
- **Cypher** is a query language for the **Neo4j** graph database;
- **DMX** is a query language for **data mining** models;
- **Datalog** is a query language for **deductive databases**;
- **Discovery Query Language** is a query language for accessing Watson Discovery Services on **IBM Cloud**.^[2]
- **F-logic** is a declarative object-oriented language for **deductive databases** and **knowledge representation**;
- **FQL** enables you to use a **SQL**-style interface to query the data exposed by the **Graph API**. It provides advanced features not available in the **Graph API**.^[3]
- **Gellish English** is a language that can be used for queries in Gellish English Databases, for dialogues (requests and responses) as well as for information modeling and knowledge modeling;^[4]
- **Gremlin** is an **Apache Software Foundation** graph traversal language for OLTP and OLAP graph systems.
- **GraphQL** is a data query language developed by **Facebook** as an alternate to **REST** and ad-hoc **webservice** architectures.
- **HTSQL** is a query language that translates **HTTP** queries to **SQL**;
- **ISBL** is a query language for **PRTV**, one of the earliest relational database management systems;
- **Jaql** is a functional data processing and query language most commonly used for **JSON** query processing;
- **JSONiq** is a declarative query language designed for collections of **JSON** documents;
- **KQL** is a query language used in **Azure Data Explorer (Kusto)** and the CMPivot tool in **Microsoft System Center Configuration Manager**
- **LINQ** query-expressions is a way to query various data sources from **.NET** languages
- **LDAP** is an **application protocol** for querying and modifying **directory services** running over **TCP/IP**;
- **LogiQL** is a variant of Datalog and is the query language for the **LogicBlox** system.

LING query-expressions is a way to query various data sources from .NET languages

LDAP is an application protocol for querying and modifying directory services running over TCP/IP;

LogiQL is a variant of Datalog and is the query language for the LogicBlox system.

MQL is a cheminformatics query language for a substructure search allowing beside nominal properties also numerical properties;

MDX is a query language for OLAP databases;

N1QL is a Couchbase's query language finding data in Couchbase Servers;

OQL is Object Query Language;

OCL (Object Constraint Language). Despite its name, OCL is also an object query language and an OMG standard;

OPath, intended for use in querying WinFS Stores;

OttoQL, intended for querying tables, XML, and databases;

Poliqarp Query Language is a special query language designed to analyze annotated text. Used in the PoliQarp search engine;

PQL is a special-purpose programming language for managing process models based on information about scenarios that these models describe;

PTQL based on relational queries over program traces, allowing programmers to write expressive, declarative queries about program behavior.

QUEL is a relational database access language, similar in most ways to SQL;

RDQL is a RDF query language;

Rego is a query language inspired by Datalog;

ReQL is a query language used in RethinkDB;

SMARTS is the cheminformatics standard for a substructure search;

SPARQL is a query language for RDF graphs;

SPL is a search language for machine-generated big data, based upon Unix Piping and SQL.

SCL is the Software Control Language to query and manipulate Endeavor objects

SQL is a well known query language and data manipulation language for relational databases;

SuprTool is a proprietary query language for SuprTool, a database access program used for accessing data in Image/SQL (formerly TurboIMAGE) and Oracle databases;

TMQL Topic Map Query Language is a query language for Topic Maps;

TQL is a language used to query topology for HP products;

Tutorial D is a query language for truly relational database management systems (TRDBMS);

U-SQL is a data processing language invented at Microsoft

XQuery is a query language for XML data sources;

XPath is a declarative language for navigating XML documents;

XSPARQL is an integrated query language combining XQuery with SPARQL to query both XML and RDF data sources at once;

YQL is an SQL-like query language

Search engine query languages, e.g., as used by Google or Bing

AVAILABLE AT:

Onebyzero Edu | Organized Learning, Smooth Career
The Comprehensive Academic Study Platform for University Students in Bangladesh (www.onebyzeroedu.com)

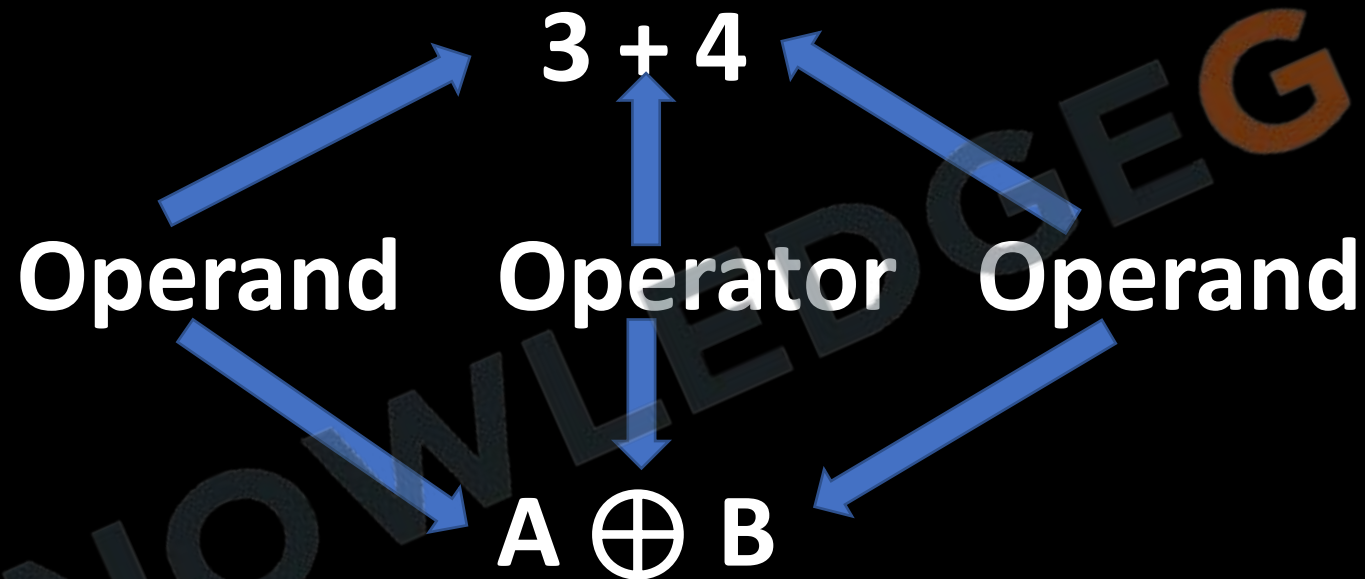
- **Procedural Query Language**
 - Here users instruct the system to performs a sequence of operations on the data base in order to compute the desired result. Means user provides both what data to be retrieved and how data to be retrieved. e.g. Relational Algebra.
- **Non-Procedural Query Language**
 - In nonprocedural language, the user describes the desired information without giving a specific procedure for obtaining that information. What data to be retrieved e.g. Relational Calculus. **Tuple relational calculus, Domain relational calculus** are declarative query languages based on mathematical logic

- Relational Algebra (Procedural) and Relational Calculus (non-procedural) are mathematical system/ query languages which are used for query on relational model.
- RA and RC are not executed in any computer they provide the fundamental mathematics on which SQL is based.
- SQL (structured query language) works on RDBMS, and it includes elements of both procedural or non-procedural query language.

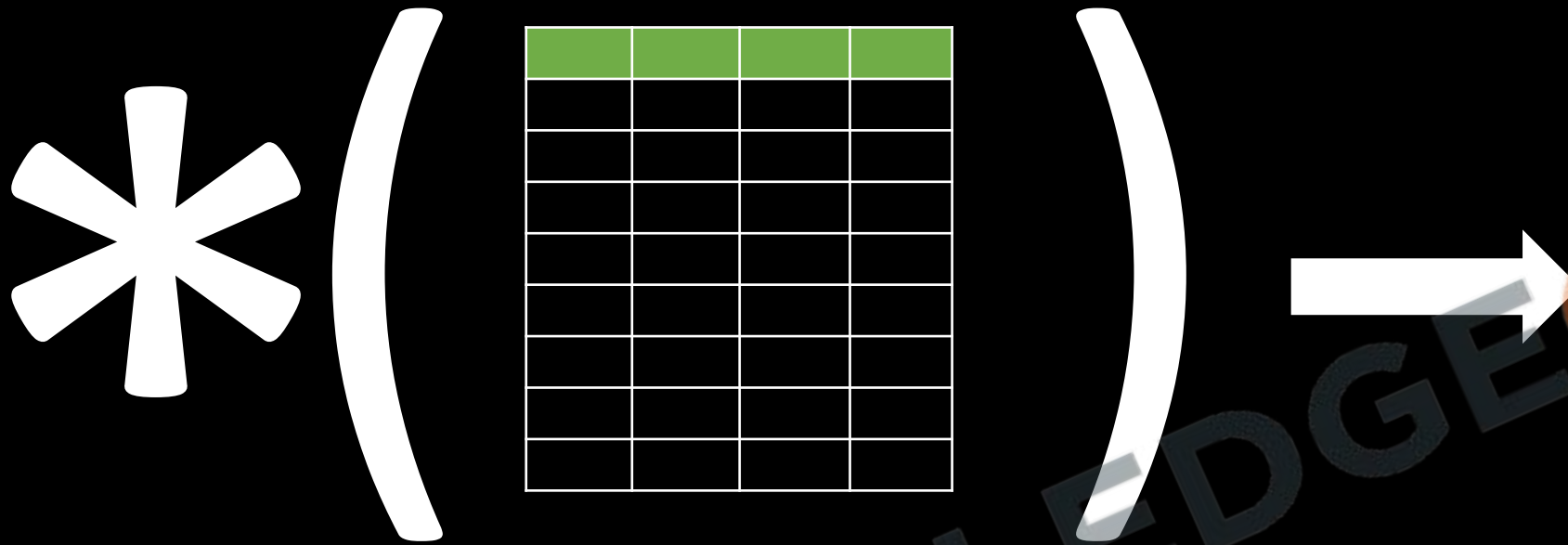
Relational model	RDBMS
RA, RC	SQL
Algo	Code
Conceptual	Reality
Theoretical	Practical
Chess	Battle Field

RELATIONAL ALGEBRA

- RA like any other mathematical system provides a number of operators and use relations (tables) as operands and produce a new relation as their result.



- Every operator in the RA accepts (one or two) relation/table as input arguments and returns always a single relation instance as the result without a name.



<http://www.knowledgegate.in/gate>

- It also does not consider duplicity by default as it is based on set theory. Same query is written in RA and SQL the result may be different as SQL considers duplication.
- As it is pure mathematics no use of English keywords. Operators are represented using symbols.

<http://www.knowledgegate.in/gate>

BASIC / FUNDAMENTAL OPERATORS

- The fundamental operations in the relational algebra are **select**, **project**, **union**, **set difference**, **Cartesian product**, and **Rename**.

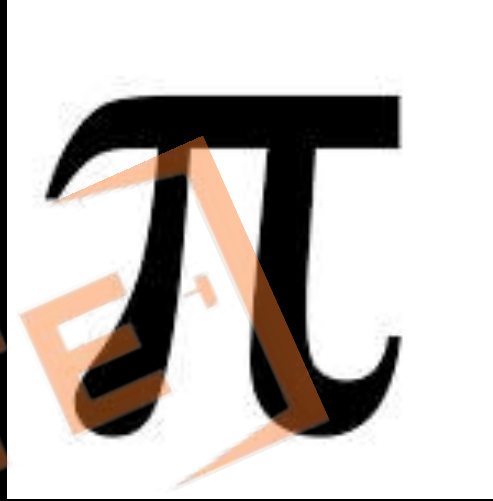
Name	Symbol
Select	(σ)
Project	(π)
Union	(\cup)
Set difference	$(-)$
Cross product	(\times)
Rename	(ρ)

- The **select, project, and rename** operations are called **unary operations**, because they operate on one relation.
- **Union, Cartesian product and set difference** operate on pairs of relations and are, therefore, called **binary operations**.

<http://www.knowledgegate.in/gate>

The Project Operation (Vertical Selection)

- Main idea behind project operator is to select desired columns.
- The project operation is a unary operation that returns its argument relation, with certain attributes left out.
- Projection is denoted by the uppercase Greek letter pi (Π).
- $\Pi_{\text{column_name}}(\text{table_name})$



<http://www.onebyzeroedu.com>

[in/gate](http://www.onebyzeroedu.com)

- We list those attributes that we wish to appear in the result as a subscript to Π , argument relation follows in parentheses.
- Minimum number of columns selected can be 1, Maximum selected Columns can be $n - 1$.

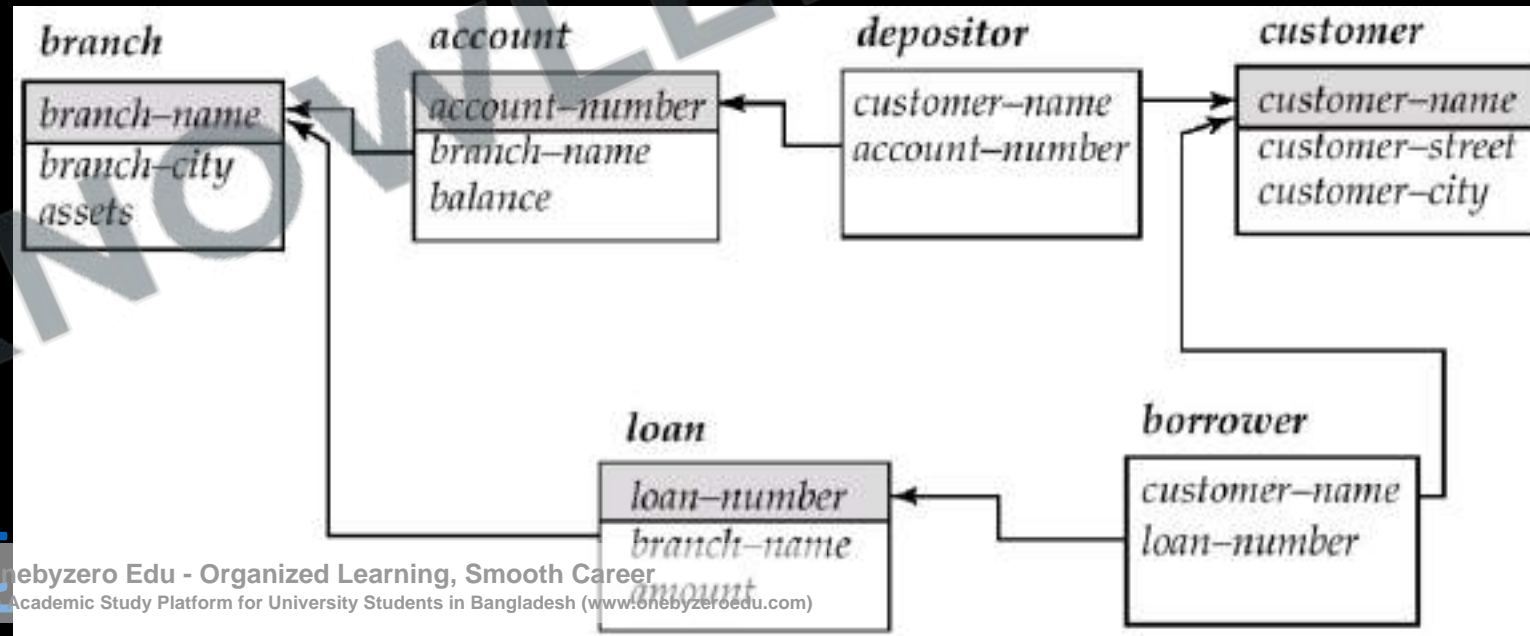


- Some points to remember
 - Eliminates duplicate rows in a result relation by default.
 - $\Pi_{A_1, A_2, \dots, A_n}(r)$, A_1, A_2, \dots, A_n refers to the set of attributes to be projected.

<http://www.knowledgegate.in/gate>

Q Write a RELATIONAL ALGEBRA query to find the name of all customer without duplication having bank account?

Q Write a RELATIONAL ALGEBRA query to find all the details of bank branches?



Q Suppose $R_1(A, B)$ and $R_2(C, D)$ are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2 . If data in r_1 and r_2 satisfy referential integrity constraints, which of the following is ALWAYS TRUE? (Gate-2012) (2 Marks)

a) $\Pi_B(r_1) - \Pi_C(r_2) = \emptyset$

b) $\Pi_C(r_2) - \Pi_B(r_1) = \emptyset$

c) $\Pi_B(r_1) = \Pi_C(r_2)$


d) $\Pi_B(r_1) - \Pi_C(r_2) \neq \emptyset$

Roll no(A)	Br_code(B)
1	101
2	101
3	101
4	102

Br_code(C)	Br_name(D)
101	CS
102	EC
103	ME

The Select Operation (Horizontal Selection)

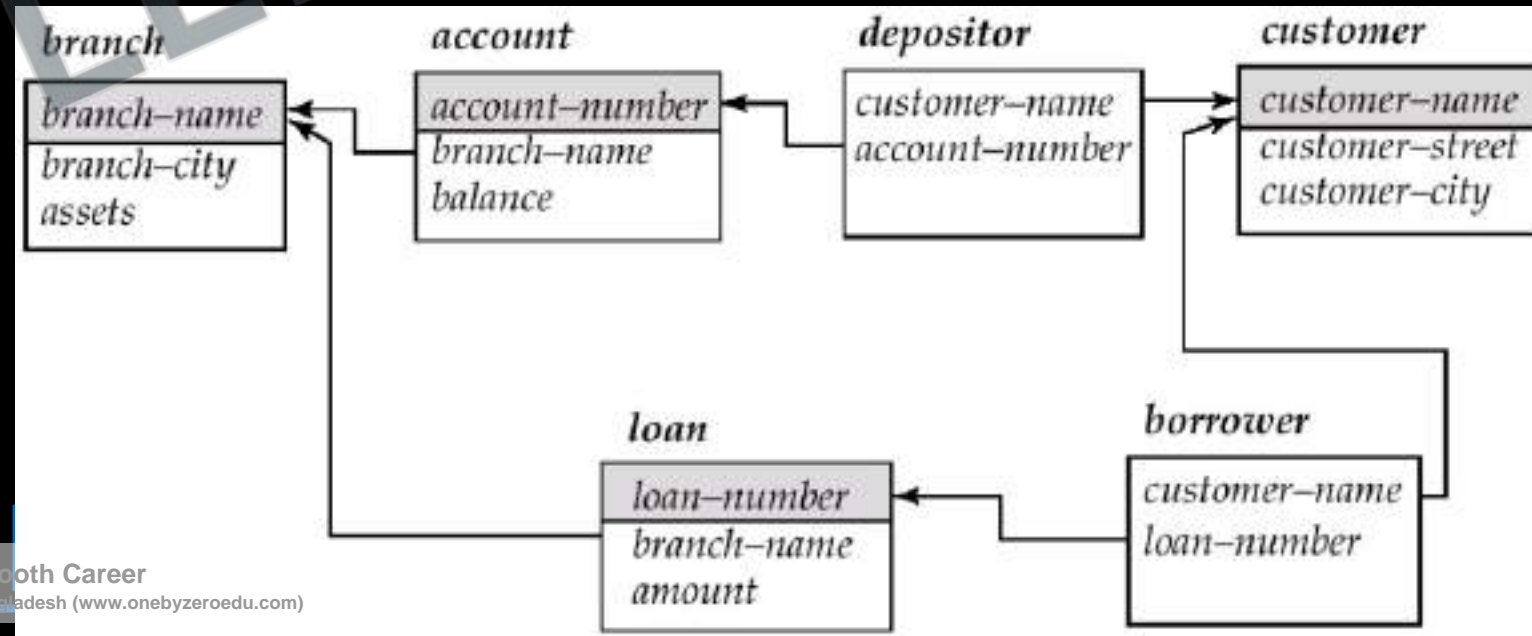
- The select operation selects tuples that satisfy a given predicate/Condition p.
- Lowercase Greek letter sigma (σ) is used to denote selection.
- It is a unary operator.
- Eliminates only tuples/rows.
- $\sigma_{\text{condition}}(\text{table_name})$




Q Write a RELATIONAL ALGEBRA query to find all account_no where balance is less the 1000?

Q Write a RELATIONAL ALGEBRA query to find branch name which is situated in Delhi and having assets less than 1,00,000?

Q Write a RELATIONAL ALGEBRA query to find branch name and account_no which has balance greater than equal to 1,000 but less than equal to 10,000?



- Predicate appears as a subscript to σ , the argument relation is in parentheses after the σ .
- Commutative in Nature, $\sigma_{p_2 \wedge p_1}(r) = \sigma_{p_1 \wedge p_2}(r) = \sigma_{p_1}(\sigma_{p_2}(r)) = \sigma_{p_2}(\sigma_{p_1}(r))$



<http://www.knowledgegate.in/gate>

- Some points to remember

- We allow comparisons using $=$, \neq , $<$, $>$, \leq and \geq in the selection predicate.
- Using the connectives and (\wedge), or (\vee), and not (\neg), we can combine several predicates into a larger predicate.
- Minimum number of tuples selected can be 0, Maximum selected tuples can be all.
- Degree (Result relation) = degree (parent relation), where degree refers to no. of attributes.
- $0 \leq \text{cardinality (result relation)} \leq \text{cardinality (parent relation)}$, where cardinality refers to no. of tuples.

Q What is the optimized version of the relation algebra expression $\pi_{A_1}(\pi_{A_2}(\sigma_{F_1}(\sigma_{F_2}(r))))$, where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expressions based on the attributes in r ? **(Gate-2014) (2 Marks)**

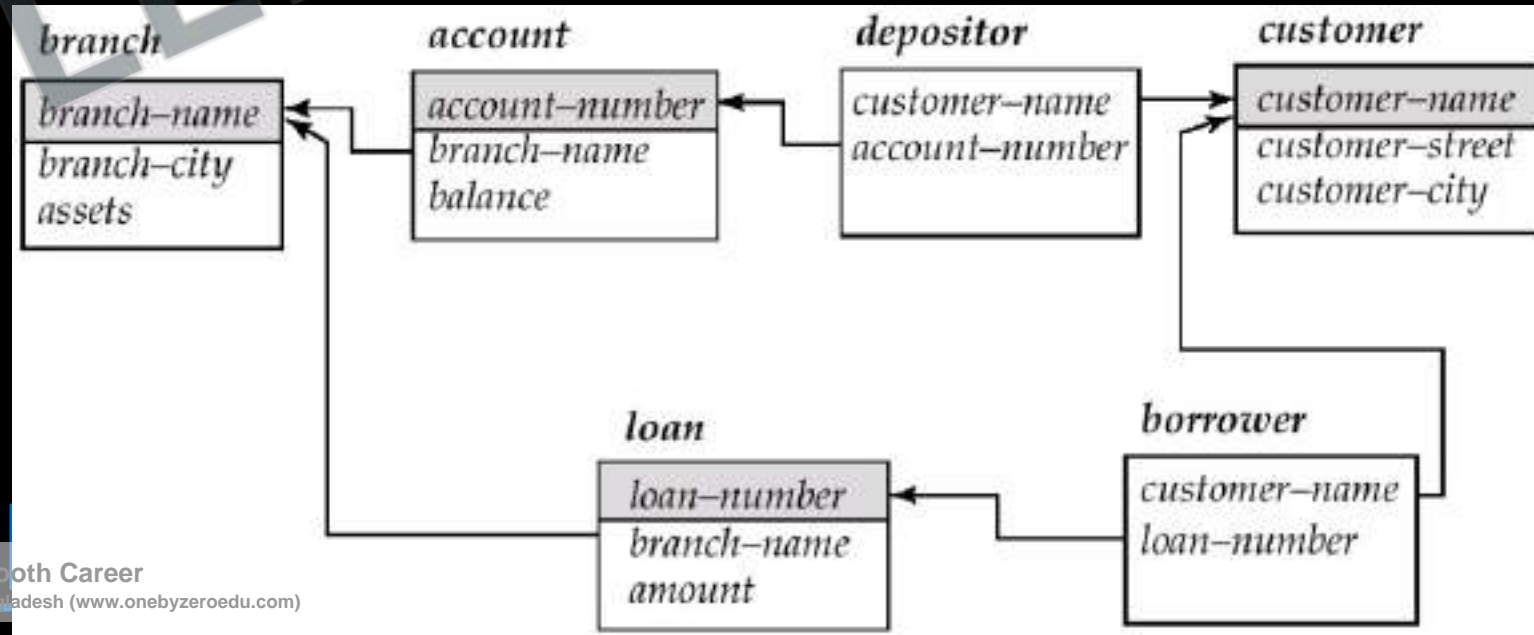
a) $\pi_{A_1}(\sigma_{(F_1 \wedge F_2)}(r))$

b) $\pi_{A_1}(\sigma_{(F_1 \vee F_2)}(r))$

c) $\pi_{A_2}(\sigma_{(F_1 \wedge F_2)}(r))$

d) $\pi_{A_2}(\sigma_{(F_1 \vee F_2)}(r))$

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have a loan or an account or both?



The Union Operation

- It is a binary operation, denoted, as in set theory, by \cup .
- Written as, $\text{Expression}_1 \cup \text{Expression}_2$, $r \cup s = \{t \mid t \in r \text{ or } t \in s\}$
- For a union operation $r \cup s$ to be valid, we require that two conditions hold:
 - The relations r and s must be of the same arity. That is, they must have the same number of attributes, the domains of the i th attribute of r and the i th attribute of s must be the same, for all i .
 - Mainly used to fetch data from different relations.



The Cartesian-Product Operation

- The Cartesian-product operation, denoted by a cross (\times), allows us to combine information from any two relations.
- It is a binary operator; we write the Cartesian product of relations R_1 and R_2 as $R_1 \times R_2$.
- Cartesian-product operation associates every tuple of R_1 with every tuple of R_2 .
 - $R_1 \times R_2 = \{rs \mid r \in R_1 \text{ and } s \in R_2\}$, contains one tuple $\langle r, s \rangle$ (concatenation of tuples r and s) for each pair of tuples $r \in R_1, s \in R_2$.

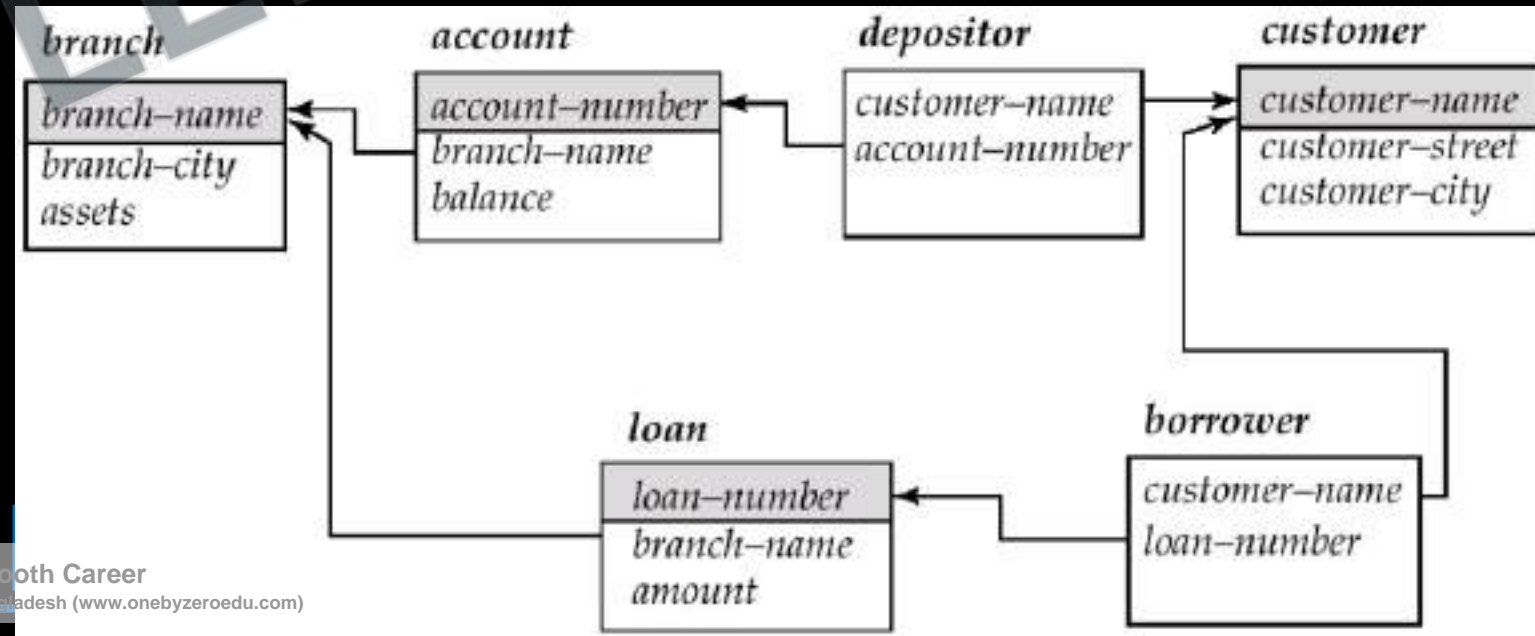
R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \times R_2$			
A	$R_1.B$	$R_2.B$	C

- $R_1 \times R_2$ returns a relational instance whose schema contains all the fields of R_1 (in order as they appear in R_1) and all fields of R_2 (in order as they appear in R_2).
- If R_1 has m tuples and R_2 has n tuples the result will be having $= m * n$ tuples.
- Same attribute name may appear in both R_1 and R_2 , we need to devise a naming schema to distinguish between these attributes.

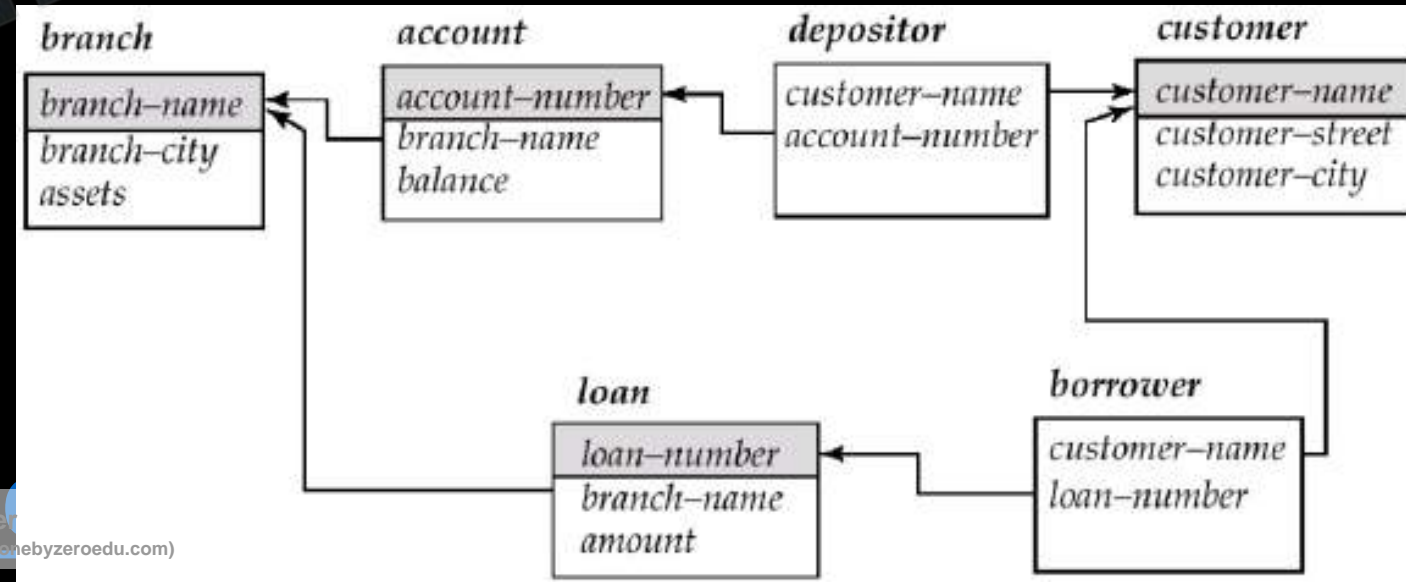
Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?



Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with loan amount, who have a loan in the bank?

Q Write a RELATIONAL ALGEBRA query to find all loan_no along with amount and branch_name, which is situated in Delhi?

Q Write a RELATIONAL ALGEBRA query to find the name of the customer who have an account in the branch situated in Delhi and balance greater than 1000?



Rename Operation

- The results of relational algebra are also relations but without any name.
- The rename operation allows us to rename the output relation. It is denoted with small Greek letter rho ρ . Where the result of expression E is saved with name of x .

- $\rho_{x(A1, A2, A3, A4, \dots, AN)}(E)$
- $\rho_{\text{Learner}}(\text{Student})$
- $\rho_{\text{Learner}(\text{Stu_ID, User_Name, Age})}(\text{Student}(\text{Roll_No, Name, Age}))$

<http://www.knowledgegate.in/gate>

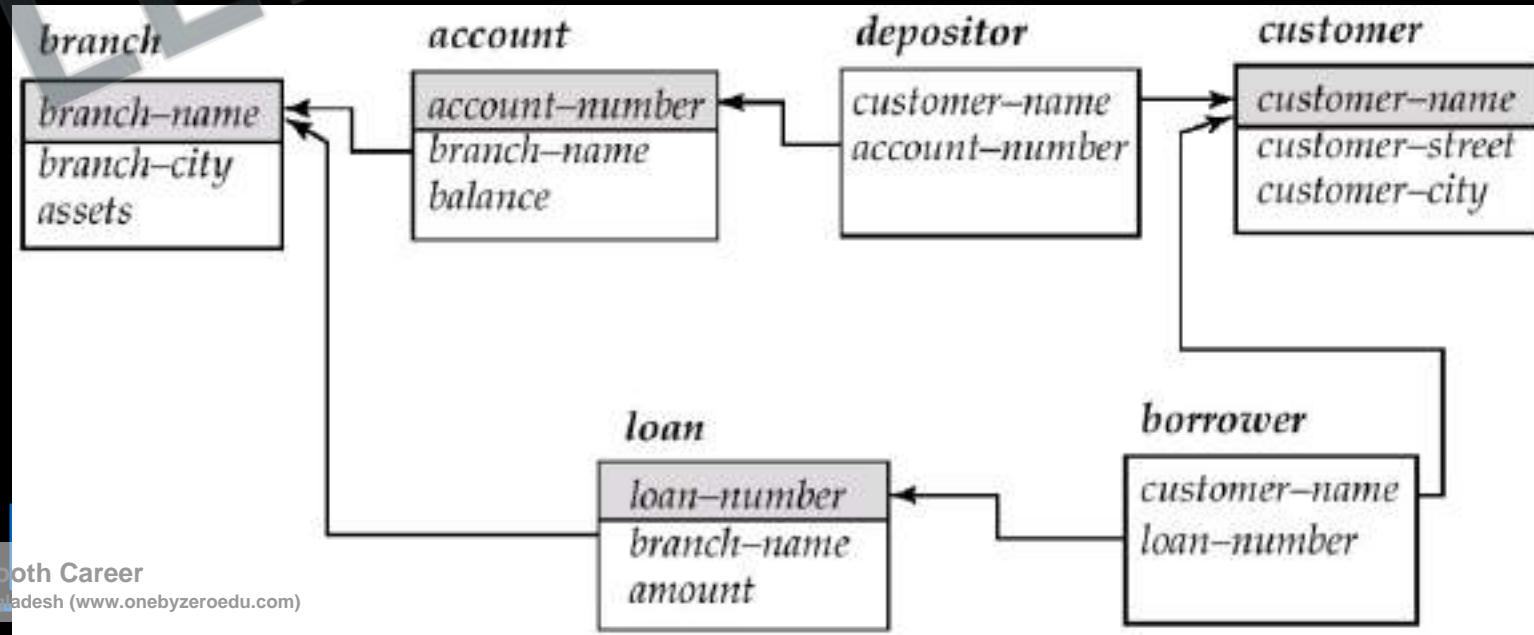
- Deposit = (Branch-name, Account- number, Customer-name, Balance)
- $\rho_{\text{Plus_Cust(Name)}}(\pi_{\text{customer-name}}(\sigma_{\text{balance} > 10000}(\text{Deposit})))$

Important Point

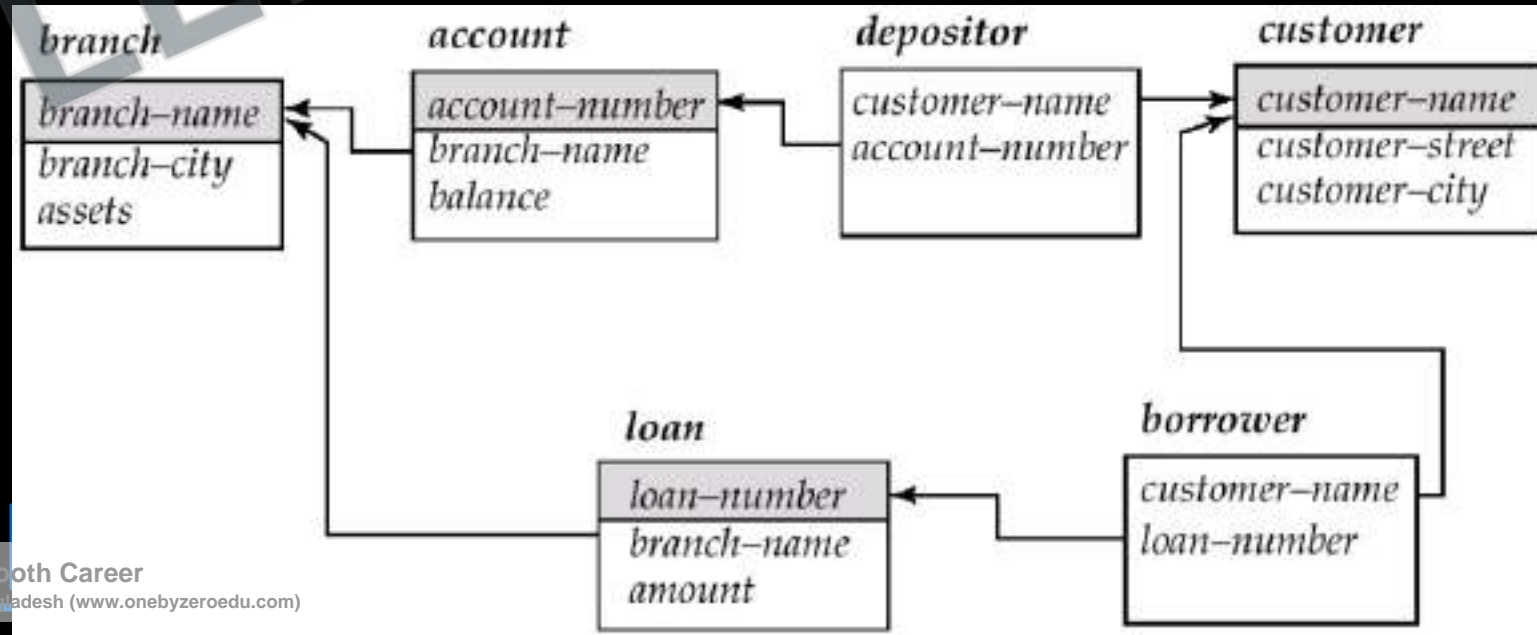
- $\rho_{\text{Learner}}(\text{Student})$
 - This Query do not change the name of the table in the original data base, but create a new copy of the table student with name Learner

<http://www.knowledgegate.in/gate>

Q Write a RELATIONAL ALGEBRA query to find the account_no along with balance with 8% interest as total amount, with table name as balance sheet?



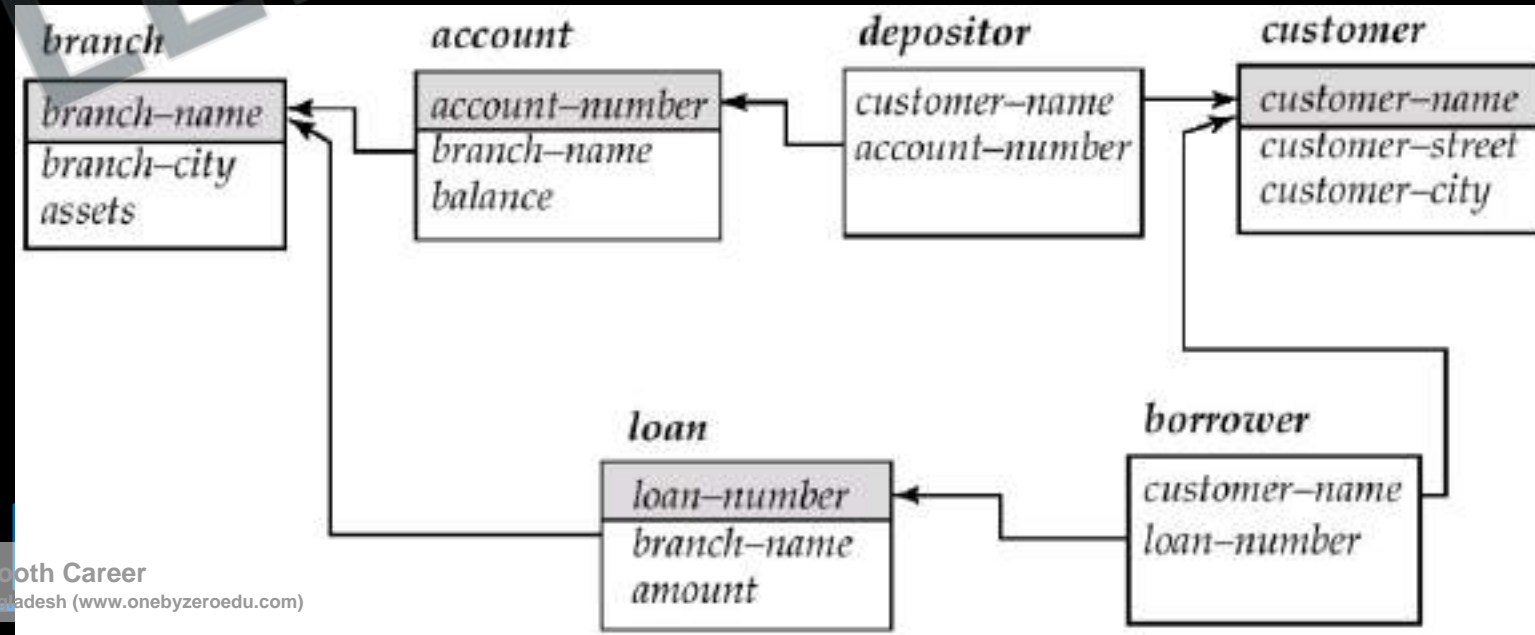
Q Write a RELATIONAL ALGEBRA query to find the loan_no with maximum loan amount?



Additional/Derived Relational-Algebra Operations

- If we restrict ourselves to just the fundamental operations, certain common queries are lengthy to express. Therefore, we use additional operations.
- These additional operations do not add any power to the algebra.
- They are used to simplify the queries.

Q Write a RELATIONAL ALGEBRA query to find all the customer name who have both a loan and an account?



Set-Intersection Operation

- We will be using \cap symbol to denote set intersection.
- $r \cap s = r - (r - s)$
- Set intersection is not a fundamental operation and does not have any power to the relational algebra.
- $r \cap s = \{t \mid t \in r \text{ and } t \in s\}$
- $0 \leq |R \cap S| \leq \min(|R|, |S|)$



Q Let R1 (A, B, C) and R2 (D, E) be two relation schema, where the primary keys are shown underlined, and let C be a foreign key in R1 referring to R2. Suppose there is no violation of the above referential integrity constraint in the corresponding relation instances r1 and r2. Which one of the following relational algebra expressions would necessarily produce an empty relation? (Gate-2004) (1 Marks)

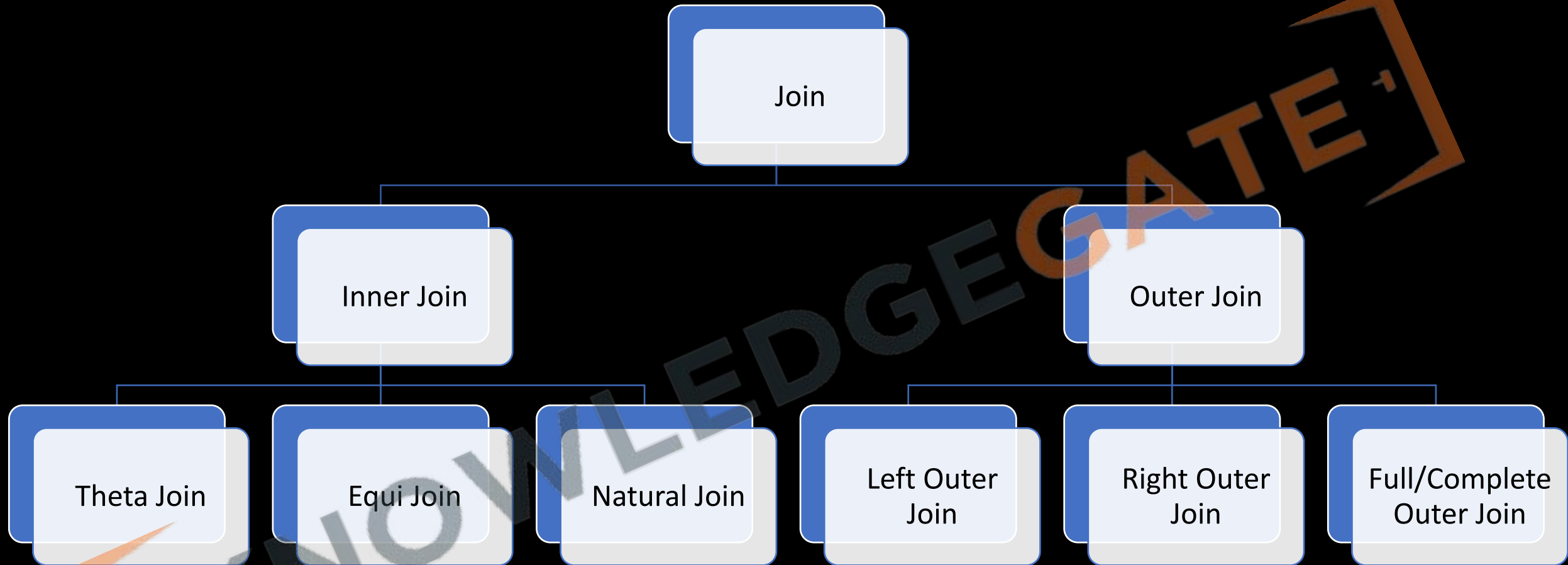
a) $\Pi_D(r_2) - \Pi_C(r_1)$

b) $\Pi_C(r_1) - \Pi_D(r_2)$

c) $\Pi_D(r_1 \bowtie_{C \neq D} r_2)$

d) $\Pi_C(r_1 \bowtie_{C=D} r_2)$

Join Operation



Theta join

- The theta join/ Conditional join operation is a variant of the Cartesian product operation that allows us to combine a selection and a Cartesian product into a single operation.
- The theta join operation $r \bowtie_{\theta} s$ is defined as follows: $r \bowtie_{\theta} s = \sigma_{\theta}(r \times s)$

- The general form of a Theta Join operation on two relations $R(A_1, A_2, \dots, A_n)$ and $S(B_1, B_2, \dots, B_m)$ is

$$R \bowtie_{\theta} S$$

- The result of the JOIN is a relation Q with $n + m$ attributes $Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$ in that order; Q has one tuple for each combination of tuples—one from R and one from S —whenever the combination satisfies the join condition.

- A general join condition is of the form
 - $\langle \text{Condition} \rangle \text{AND} \langle \text{Condition} \rangle \text{AND} \dots \text{AND} \langle \text{Condition} \rangle$
 - where each $\langle \text{Condition} \rangle$ is of the form $A_i \theta B_j$, A_i is an attribute of R , B_j is an attribute of S , A_i and B_j have the same domain, and θ (theta) is one of the comparison operators $\{=, <, \leq, \neq, >, \geq\}$.
- A JOIN operation with such a general join condition is called a THETA JOIN. Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

Equi Join

- The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN.

<http://www.knowledgegate.in/gate>

Q Consider the following relations A, B and C:

A			B			C		
ID	Name	Age	ID	Name	Age	ID	Phone	Area
12	Arun	60	15	Shreya	24	10	2200	02
15	Shreya	24	25	Hari	40	99	2100	01
99	Rohit	11	98	Rohit	20			
			99	Rohit	11			

How many tuples does the result of the following relational algebra expression contain? Assume that the schema of AUB is the same as that of A. (Gate-2007) (2 Marks)

$(A \cup B) \bowtie_{A.Id > 40 \vee C.Id < 15} C$

a) 7

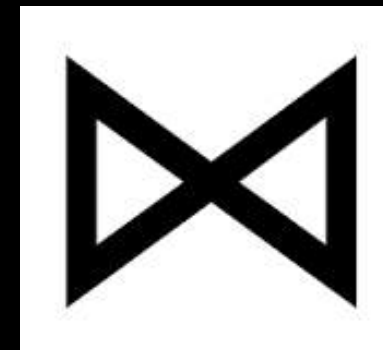
b) 4

c) 5

d) 9

The Natural-Join Operation

- The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.
- The natural join of r and s , denoted by $r \bowtie s$
- The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first.



- The natural join of r and s is a relation on schema $R \cup S$ formally defined as follows:

$$r \bowtie s = \Pi_{R \cup S} (\sigma_{r.A_1=s.A_1 \wedge r.A_2=s.A_2 \wedge \dots \wedge r.A_n=s.A_n} (r \times s))$$

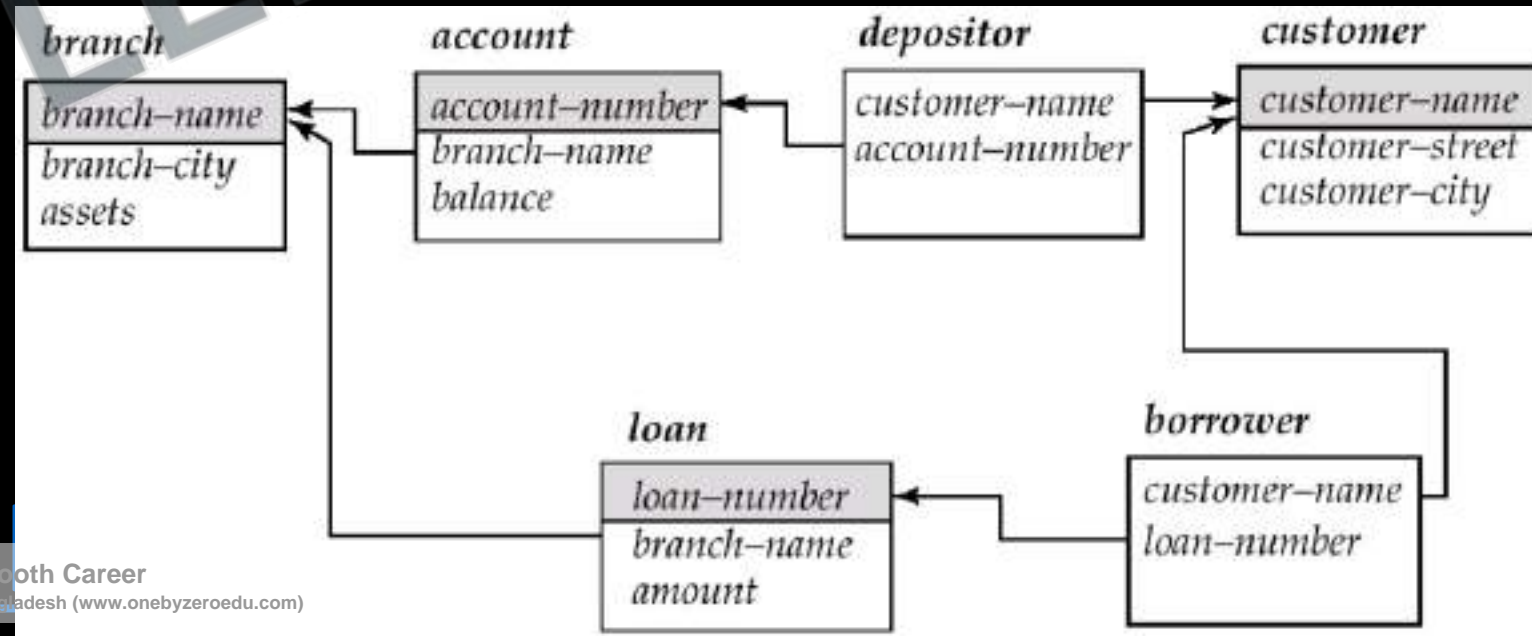
R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \bowtie R_2$		

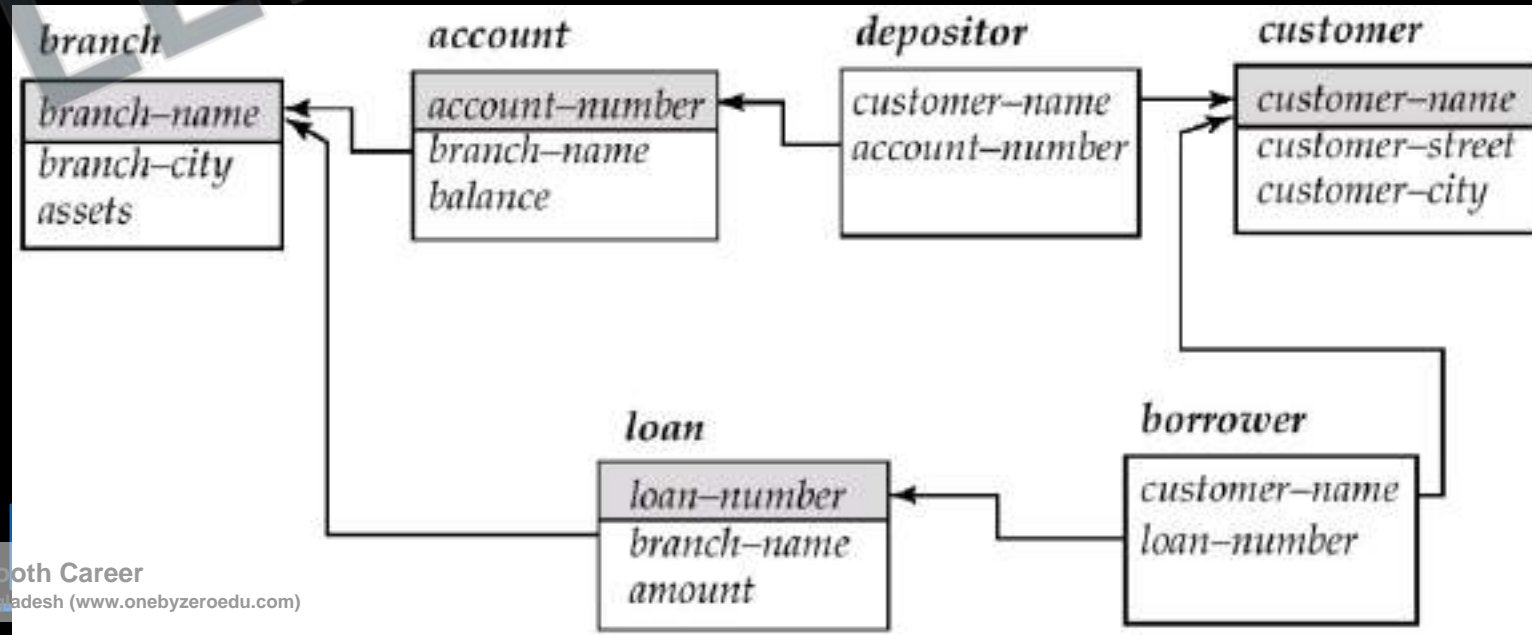
Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with account balance, who have an account in the bank?

Q Write a RELATIONAL ALGEBRA query to find the name of all the customers along with loan amount, who have a loan in the bank?



Q Write a RELATIONAL ALGEBRA query to find all loan_no along with amount and branch_name, which is situated in Delhi?

Q Write a RELATIONAL ALGEBRA query to find the name of the customer who have an account in the branch situated in Delhi and balance greater than 1000?



Notes

$$(instructor \bowtie teaches) \bowtie course = instructor \bowtie (teaches \bowtie course)$$

- The natural join is a Lossy operator

R ₁	
A	B
1	P
2	Q
3	R

R ₂	
B	C
Q	X
R	Y
S	Z

R ₁ ⋈ R ₂		
A	B	C

Q The following functional dependencies hold for relations R(A, B, C) and S(B, D, E).

$B \rightarrow A$

$A \rightarrow C$

The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural join $R \bowtie S$? **(Gate-2010) (2 Marks)**

a) 100

b) 200

c) 300

d) 2000

<http://www.knowledgegate.in/gate>

Q Consider the join of a relation R with a relation S. If R has m tuples and S has n tuples, then the maximum and minimum sizes of the join respectively are: (Gate-1999) (1 Marks)

(A) $m+n$ and 0

(B) mn and 0

(C) $m+n$ and $m-n$

(D) mn and $m+n$

<http://www.knowledgegate.in/gate>

Outer join Operations

- The outer-join operation is an extension of the join operation to deal with missing information.
- The outer join operation works in a manner similar to the natural join operation, but preserves those tuples that would be lost in a join by creating tuples in the result containing null values.
- We can use the outer-join operation to avoid this loss of information.

- There are actually three forms of the operation:
 - left outer join, denoted $\bowtie\leftarrow$
 - right outer join, denoted $\rightarrow\bowtie$
 - full outer join, denoted $\bowtie\rightleftarrows$

Left Outer Join

- The left outer join (\bowtie) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \bowtie R_2$		
A	B	C

Q Consider the relations $r(A, B)$ and $s(B, C)$, where $s.B$ is a primary key and $r.B$ is a foreign key referencing $s.B$. Consider the query

Q: $r \bowtie (\sigma_{B < 5}(s))$

Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values. Which of the following is NOT equivalent to Q ? **(Gate-2018) (2 Marks)**

a) $\sigma_{B < 5}(r \bowtie s)$

b) $\sigma_{B < 5}(r \text{ LOJ } s)$

c) $r \text{ LOJ } (\sigma_{B < 5}(s))$

d) $\sigma_{B < 5}(r) \text{ LOJ } s$

Right Outer Join

- The right outer join (\bowtie_r) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \bowtie_r R_2$		
A	B	C
1	P	X
2	Q	Y
3	R	Z

Full Outer Join

- The full outer join(\bowtie) does both the left and right outer join operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 \bowtie R_2$		
A	B	C

Q Consider two relations $R_1(A, B)$ with the tuples $(1, 5)$, $(3, 7)$ and $R_2(A, C) = (1, 7)$, $(4, 9)$. Assume that $R(A, B, C)$ is the full natural outer join of R_1 and R_2 . Consider the following tuples of the form (A, B, C)

$a = (1, 5, \text{null})$,

$b = (1, \text{null}, 7)$,

$c = (3, \text{null}, 9)$,

$d = (4, 7, \text{null})$,

$e = (1, 5, 7)$,

$f = (3, 7, \text{null})$,

$g = (4, \text{null}, 9)$.

Which one of the following statements is correct? **(Gate-2015) (1 Marks)**

(A) R contains a, b, e, f, g but not c, d

(B) R contains a, b, c, d, e, f, g

(C) R contains e, f, g but not a, b

(D) R contains e but not f, g

DIVISION

- In general, the DIVISION operation is applied to two relations $R(Z) \div S(X)$, where the attributes of R are a subset of the attributes of S ; that is, $X \subseteq Z$.
- $Z = \{A, B\}$
- $X = \{A\}$
- $Y = Z - X = \{B\}$
- This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S .
- Note that in the formulation of the DIVISION operation, the tuples in the denominator relation S restrict the numerator relation R by selecting those tuples in the result that match all values present in the denominator.

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

S
A
a1
a2
a3

T
B
b1
b4

- The semantics of the division is defined as follows:
 - $R \div S = \{ t[a_1, \dots, a_n] : t \in R \wedge \forall s \in S ((t[a_1, \dots, a_n] \cup s) \in R) \}$
 - where $\{a_1, \dots, a_n\}$ is the set of attribute names unique to R and $t[a_1, \dots, a_n]$ is the restriction of t to this set. It is usually required that the attribute names in the header of S are a subset of those of R because otherwise the result of the operation will always be empty.

Completed		DBProject	Completed \div DBProject
Student	Task	Task	Student
Fred	Database1	Database1	Fred
Fred	Database2	Database2	Sarah
Fred	Compiler1		
Eugene	Database1		
Eugene	Compiler1		
Sarah	Database1		
Sarah	Database2		

AVAILABLE AT:

$$\pi_{\text{Student}}(R) - \{\pi_{\text{Student}}[(\pi_{\text{Student}}(R) \times S) - \pi_{\text{Student,Task}}(R)]\}$$

Completed		DBProject	Completed
Student	Task	Task	÷ DBProject
Fred	Database1	Database1	Student
Fred	Database2	Database2	Fred
Fred	Compiler1		Sarah
Eugene	Database1		
Eugene	Compiler1		
Sarah	Database1		
Sarah	Database2		

<http://www.knowledgegate.in/gate>

$$\pi_{\text{Student}}(R) - \{\pi_{\text{Student}}[(\pi_{\text{Student}}(R) \times S) - \pi_{\text{Student,Task}}(R)]\}$$

Completed		DBProject	Completed
Student	Task	Task	÷ DBProject
Fred	Database1	Database1	Student
Fred	Database2	Database2	Fred
Fred	Compiler1		Sarah
Eugene	Database1		
Eugene	Compiler1		
Sarah	Database1		
Sarah	Database2		

$$T := \pi_{\text{Student}}(R) \times S$$

$$U := T - \pi_{\text{Student,Task}}(R)$$

$$V := \pi_{\text{Student}}(U)$$

$$W := \pi_{\text{Student}}(R) - V$$

Q Consider the given table R and S find the number of elements retrieved by the query

Table R

A	B
1	Dog
1	Cat
1	Cow
2	Cat
4	Cat
3	Dog
4	Dog
2	Dog
4	Cow

Table S

B
Cat
Dog

$\Pi_{A,B}(R) \div \Pi_B(S)$ _____

<http://www.knowledgegate.in/gate>

Q Consider a database that has the relation schema CR (StudentName, CourseName). An instance of the schema CR is as given below. The following query is made on the database.

$$T_1 \leftarrow \pi_{\text{CourseName}} (\sigma_{\text{StudentName}=\text{SA}}(\text{CR}))$$

$$T_2 \leftarrow \text{CR} \div T_1$$

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The number of rows in T_2 is _____. (Gate-2017) (1 Marks)

<http://www.knowledgegate.in/gate>

Q A relation $r(A,B)$ in a relational database has 1200 tuples. The attribute A has integer values ranging from 6 to 20, and the attribute B has integer values ranging from 1 to 20. Assume that the attributes A and B are independently distributed. The estimated number of tuples in the output of $\sigma_{(A>10) \vee (B=18)}(r)$ is _____. **(GATE 2021)**

- (a)** 820
- (b)** 1200
- (c)** 960
- (d)** 1000

Q The following relation records the age of 500 employees of a company, where empNo (indicating the employee number) is the key:

$empAge(\underline{empNo}, age)$

Consider the following relational algebra expression:

$\Pi_{empNo}(empAge \bowtie_{(age > age1)} \rho_{empNo1, age1}(empAge))$

What does the above expression generate? (**GATE 2021**) (2 MARKS)

- (A) Employee numbers of only those employees whose age is the maximum
- (B) Employee numbers of only those employees whose age is more than the age of exactly one other employee
- (C) Employee numbers of all employees whose age is not the minimum
- (D) Employee numbers of all employees whose age is the minimum

<http://www.knowledgegate.in/gate>

Q Consider the relational schema given below, where *eld* of the relation *dependent* is a foreign key referring to *empId* of the relation *employee*. Assume that every employee has at least one associated dependent in the *dependent* relation. **(Gate-2014) (2 Marks)**

employee (*empId*, *empName*, *empAge*)

dependent (*depId*, *eld*, *depName*, *depAge*)

Consider the following relational algebra query:

$\Pi_{empId}(employee) - \Pi_{empId}(employee \bowtie_{(empId=eld) \wedge (empAge \leq depAge)} dependent)$

The above query evaluates to the set of *empIds* of employees whose age is greater than that of
(A) some dependent.

(B) all dependents.

(C) some of his/her dependents

(D) all of his/her dependents.

<http://www.knowledgegate.in/gate>

Q Information about a collection of students is given by the relation *studInfo*(*studId*, *name*, *sex*). The relation *enroll* (*studId*, *courseId*) gives which student has enrolled for (or taken) that course(s). Assume that every course is taken by at least one male and at least one female student. What does the following relational algebra expression represent? (Gate-2007) (2 Marks)

$$\pi_{\text{courseId}}((\pi_{\text{studId}}(\sigma_{\text{sex}=\text{"female"}}(\text{studInfo})) \times \pi_{\text{courseId}}(\text{enroll})) - \text{enroll})$$

- (A) Courses in which all the female students are enrolled.
- (B) Courses in which a proper subset of female students are enrolled.
- (C) Courses in which only male students are enrolled.
- (D) None of the above

Q Consider the relation Student (name, sex, marks), where the primary key is shown underlined, pertaining to students in a class that has at least one boy and one girl. What does the following relational algebra expression produce? (Note: ρ is the rename operator). **(Gate-2004) (2 Marks)**

$\pi_{\text{name}}\{\sigma_{\text{sex=female}}(\text{Student})\} - \pi_{\text{name}}(\text{Student} \bowtie_{(\text{sex=female} \wedge \text{x=male} \wedge \text{marks} \leq \text{m})} \rho_{\text{n,x,m}}(\text{Student}))$

- a) names of girl students with the highest marks
- b) names of girl students with more marks than some boy student
- c) names of girl students with marks not less than some boy student
- d) names of girl students with more marks than all the boy students

<http://www.knowledgegate.in/gate>

Introduction to SQL

- There are a number of database query languages in use, either commercially or experimentally, around 50 are used popularly. We will study the most widely used query language 'SQL'.
- Structured Query Language is a domain-specific language (not general purpose) used in programming and design for managing data held in a relational database management system (RDBMS).
- Although we refer to the SQL language as a “query language,” it can do much more than just query a database. It can define the structure of the data base, modify data in the database, specify security constraints and number of other tasks.
- Originally based upon relational algebra(procedural) and tuple relational calculus (Non-procedural) mathematical model.

<http://www.knowledgegate.in/gate>

Overview of the SQL Query Language

1. IBM developed the original version of SQL, originally called Sequel (*Structured English Query Language*), as part of the System R project in the early 1970s.
2. The Sequel language has evolved since then, and its name has changed to SQL (*Structured Query Language*) (some other company has trademark on the word sequel). SQL has clearly established itself as *the* standard relational database language.
3. In 1986, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) published an SQL standard, called SQL-86.
4. The next version of the standard was SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011, SQL: 2016 and most recently SQL:2023.

Parts of SQL

- **Data-definition language (DDL).** The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
 - **Integrity.** The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed. e.g. not null which means Primary key value cannot be null.
 - **View definition.** The SQL DDL includes commands for defining views. e.g. sorting result in ascending or descending order with order by clause.
 - **Authorization.** The SQL DDL includes commands for specifying access rights to relations and views like read only, read/write. E.g. grant etc.

- **Data-manipulation language** (DML). The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database. e.g. insert, delete command etc.
- **Transaction control**. SQL includes commands for specifying the beginning and ending of transactions. E.g. commit, rollback, savepoint etc.
- **Embedded SQL** and **dynamic SQL**. Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java.

<http://www.knowledgegate.in/gate>

Q Which of the following is/are correct? (GATE-1999) (1 Marks)

- (a) An SQL query automatically eliminates duplicates**
- (b) An SQL query will not work if there are no indexes on the relations**
- (c) SQL permits attribute names to be repeated in the same relation**
- (d) None of the above**

<http://www.knowledgegate.in/gate>

Basic Structure of SQL Queries

- For any SQL as query, input and output both are relations.
- Number of relations inputs to a query will be at least one, but output will always be a single relation without any name unless specified, but columns will have names from input tables.

- The basic structure of an SQL query consists of three clauses: select, from, and where.
- The query takes its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result without any name unless specified.
- A typical SQL query has the form.

Select A_1, A_2, \dots, A_n (Column name)
from r_1, r_2, \dots, r_m (Relation/table name)
Where P; (Condition)

Select A_1, A_2, \dots, A_n

(Column name)

from r_1, r_2, \dots, r_m

(Relation/table name)

Where P ;

(Condition)

1. It is to be noted only select and from are mandatory clauses, and if not required then it is not essential to write where. If the **where** clause is omitted, the predicate P is **true**.
2. SQL in general is not case sensitive i.e. it doesn't matter whether we write query in upper or lower case.
3. In the formal, mathematical definition of the relational model, a relation is a set. Thus, duplicate tuples would never appear in relations.
4. In practice, duplicate elimination is time-consuming. Therefore, SQL allows duplicates in relations as well as in the results of SQL expressions. In those cases where we want to force the elimination of duplicates, we insert the keyword **distinct** after **select**, will discuss in detail later.
5. SQL allows us to use the keyword **all** to specify explicitly that duplicates are not removed, Since duplicate retention is the default, we shall not use **all** in our examples

<http://www.knowledgegate.in/gate>

Select Clause

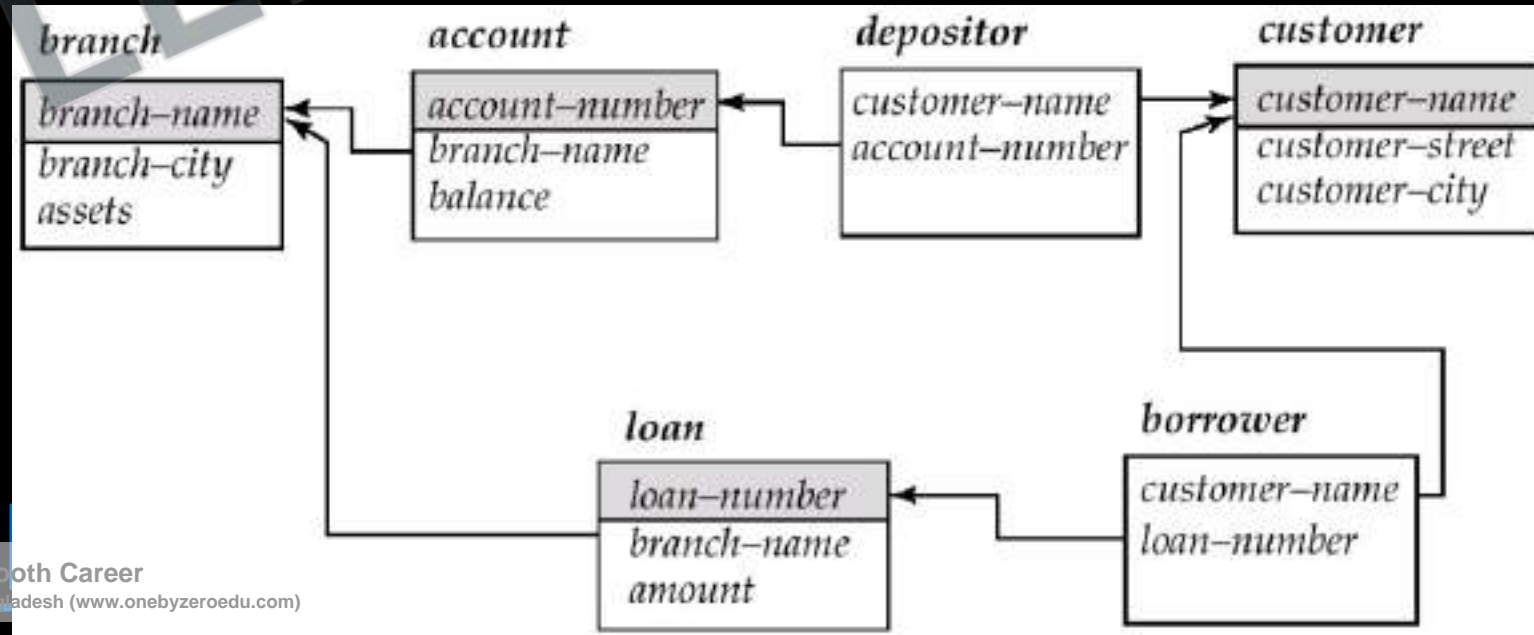
- The function of Select clause in SQL is more or less same as that of ' Π ' projection in the relational algebra. It is used to pick the column required in result of the query out of all the columns in relation/table. (Vertical filtering)
 - **Select A_1, A_2, \dots, A_n (Column name)**
- The order in which columns are represented in the select clause, will be same the column will appear in the relation.
- It is to be noted that in Relational Algebra ' Π ' projection is not mandatory and should be used only when we require less than all the column available in the table, but in SQL even if we need all the column we must use Select, the argument is it makes SQL query more readable.

- But to make things easy we can use '*' to specify that we need all columns

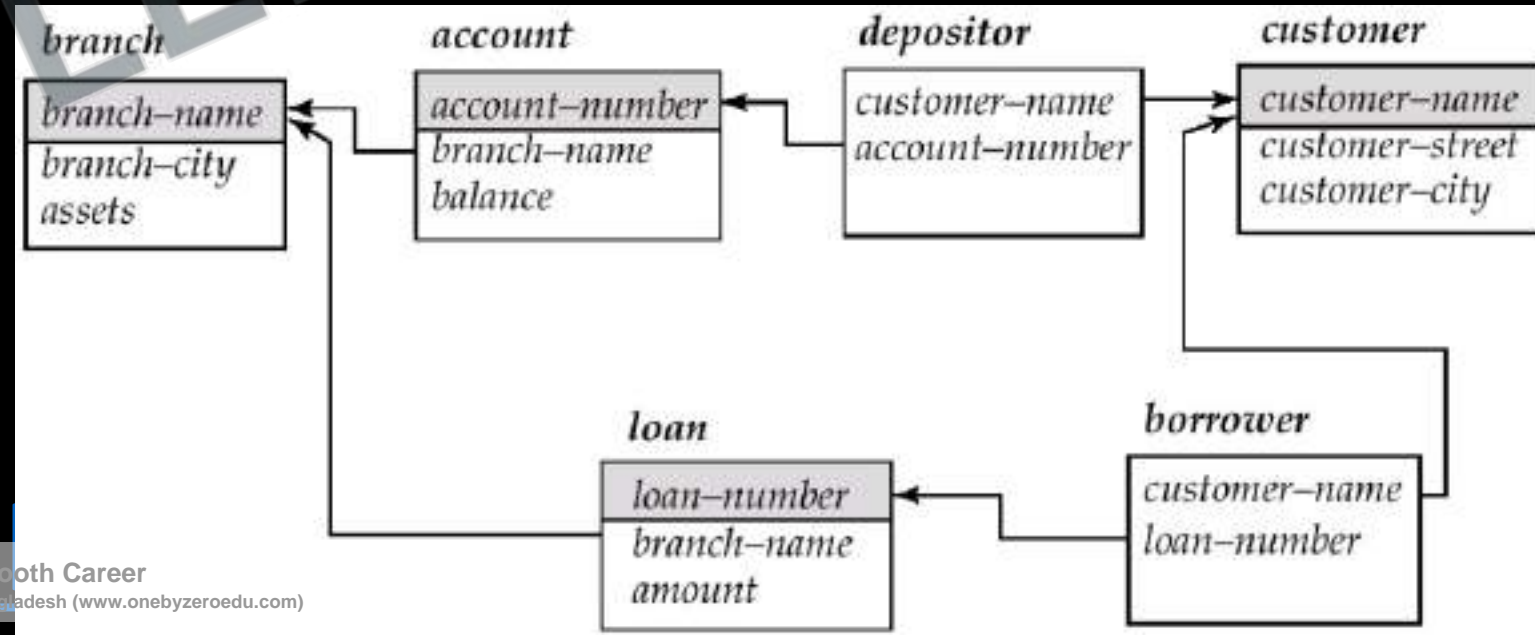
Select *

- The **select** clause may also contain arithmetic expressions involving the operators +, -, / and * operating on constants or attributes of tuples. however, that it does not result in any change to the relation/table.

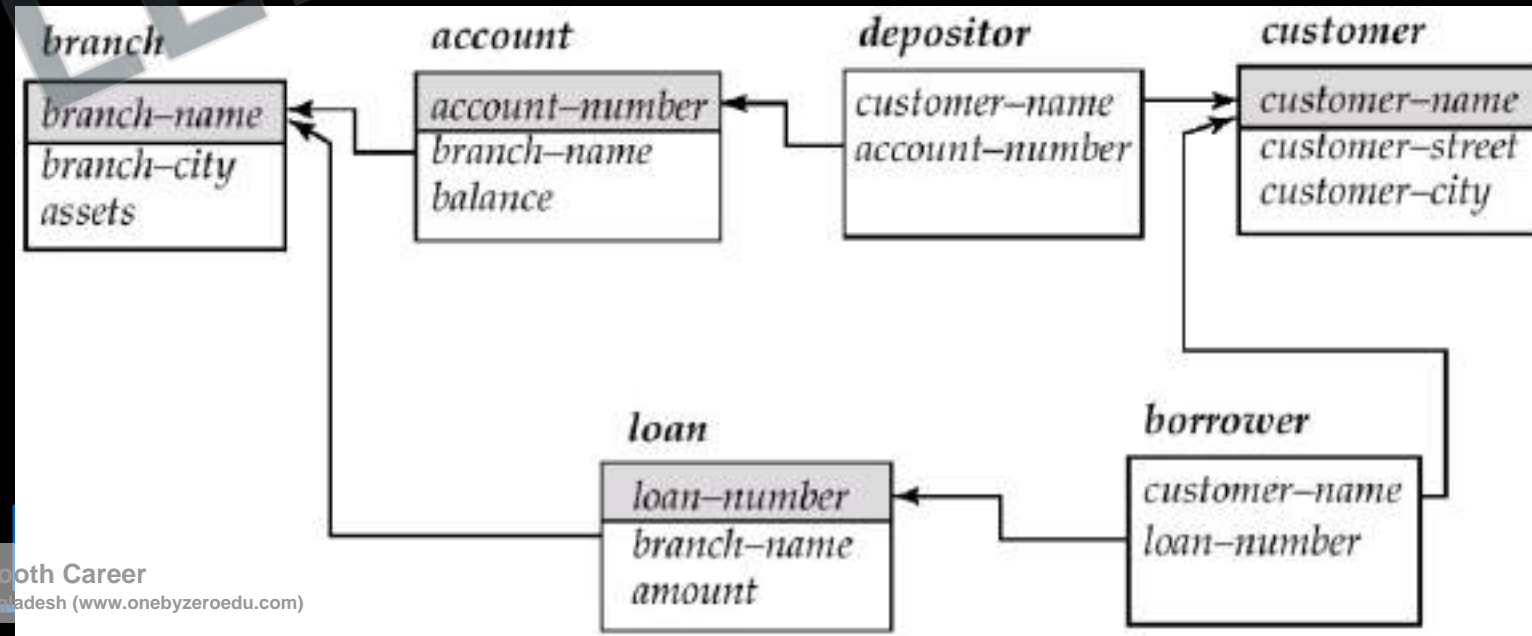
Q Write a SQL query to find all the details of bank branches?



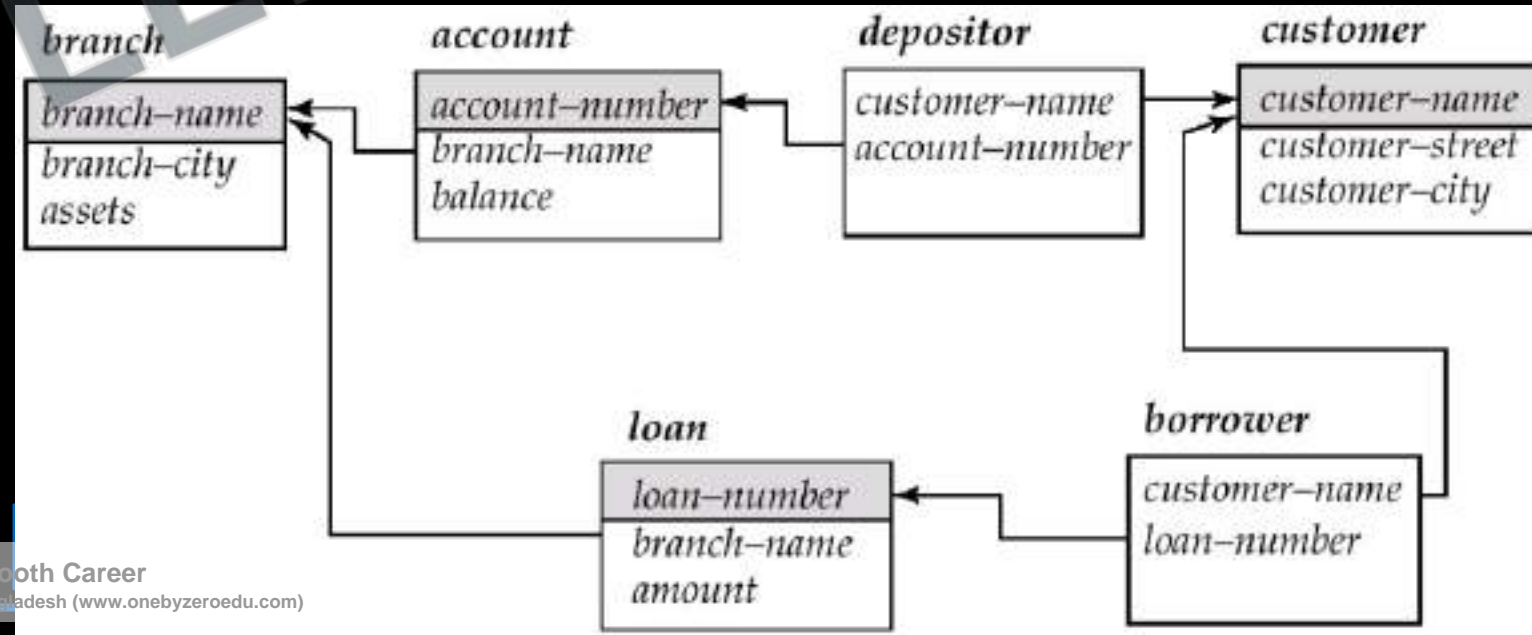
Q Write a SQL query to find each loan number along with loan amount?



Q Write a SQL query to find the name of all customer without duplication having bank account?



Q Write a SQL query to find all account_no and balance with 6% yearly interest added to it?



Q Select operation in SQL is equivalent to (Gate-2015) (1 Marks)

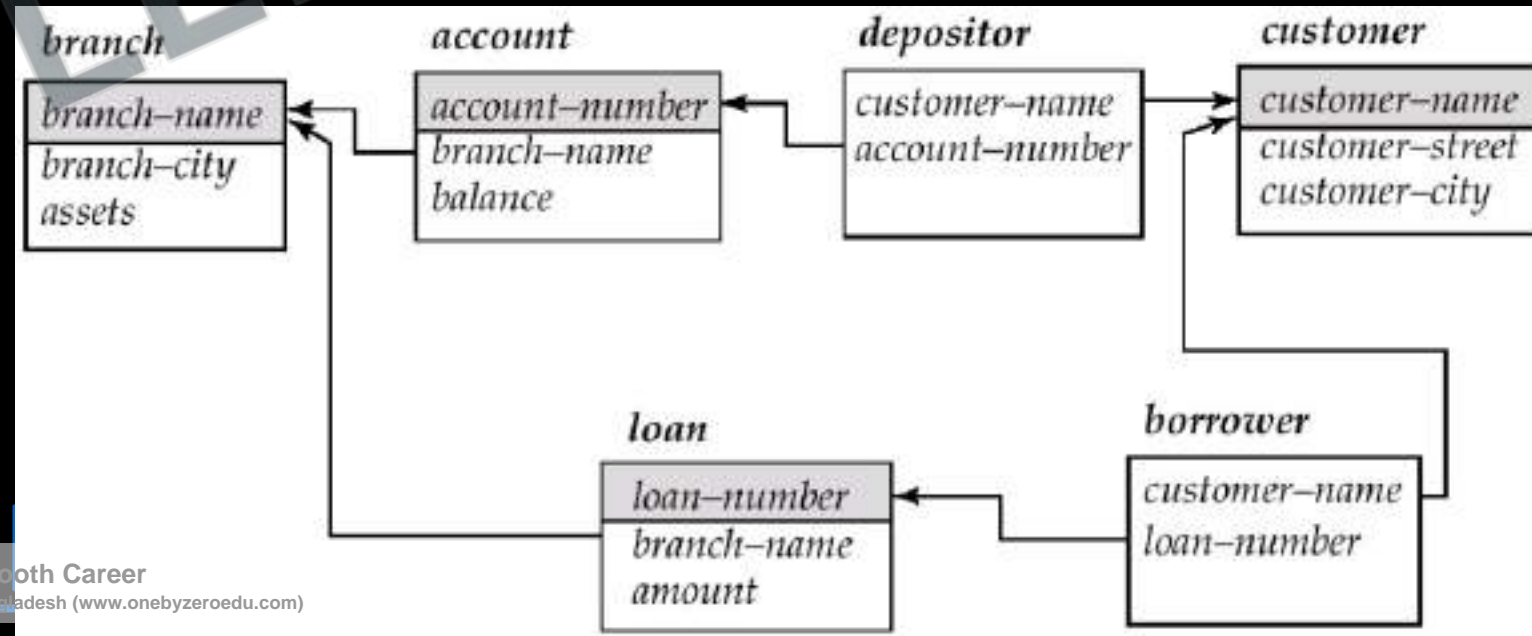
- (A)** the selection operation in relational algebra
- (B)** the selection operation in relational algebra, except that select in SQL retains duplicates
- (C)** the projection operation in relational algebra, except that select in SQL retains duplicates
- (D)** the projection operation in relational algebra

<http://www.knowledgegate.in/gate>

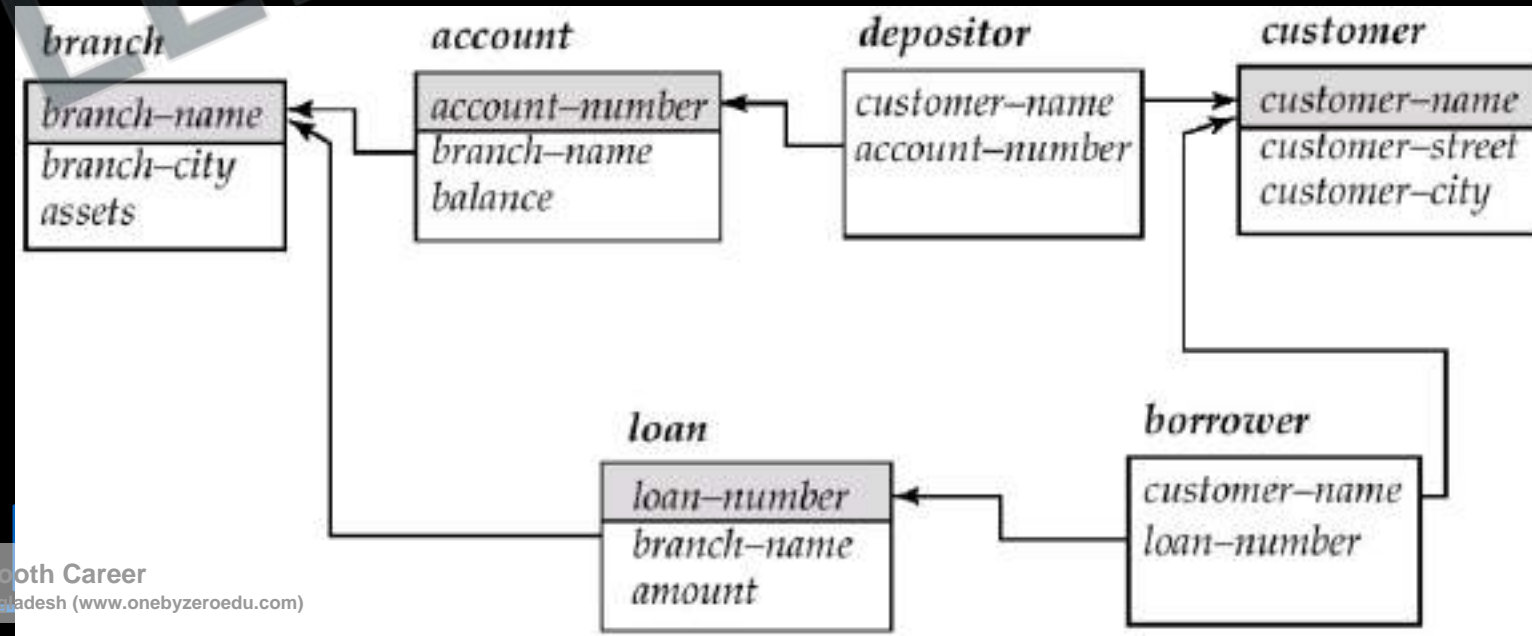
Select Clause with where clause

1. Where clause in SQL is same as ' σ ' sigma of relational algebra where we specify the conditions/Predicate (horizontal filtering)
2. Where clause can have expressions involving the comparison operators $<$, $<=$, $>$, $>=$, $=$ and $<>$. SQL allows us to use the comparison operators to compare strings and arithmetic expressions.
3. SQL allows the use of the logical connectives **and**, **or**, and **not** in the **where** clause.
4. SQL includes a **between** comparison operator to simplify **where** clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.
5. Similarly, we can use the **not between** comparison operator.

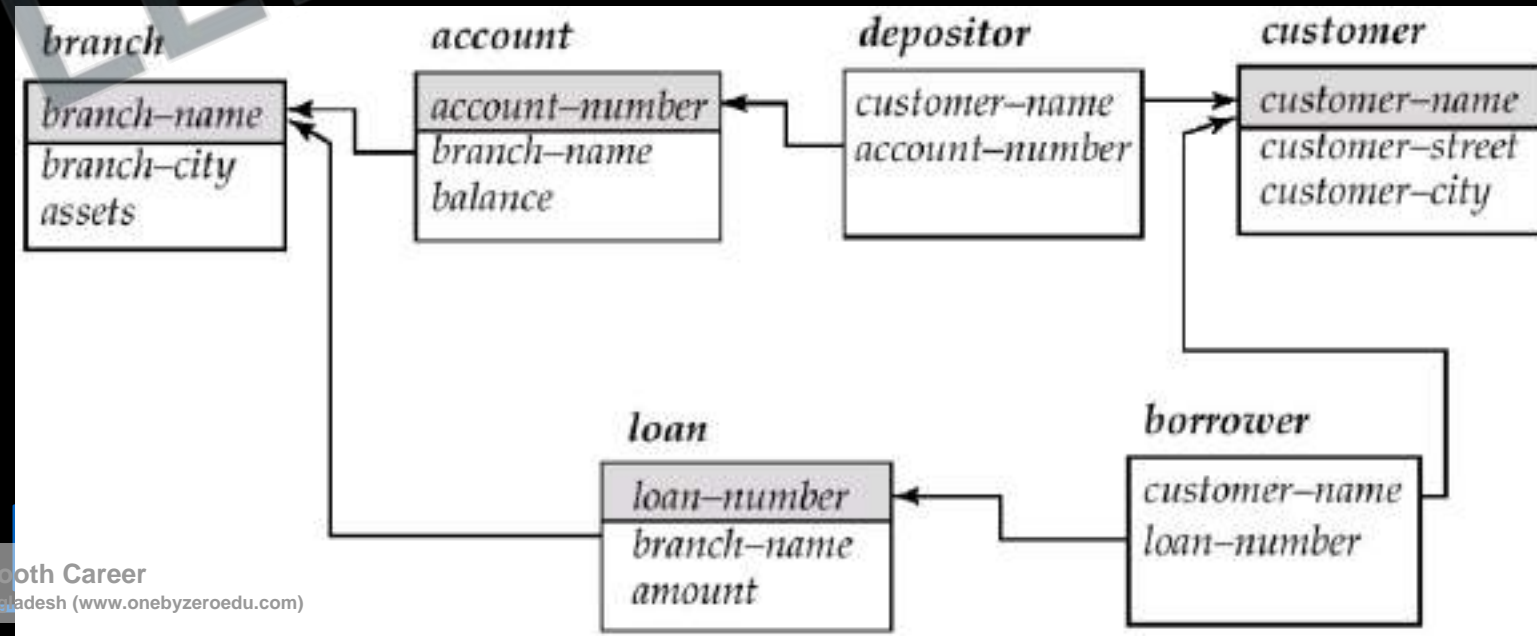
Q Write a SQL query to find all account_no where balance is less the 1000?



Q Write a SQL query to find branch name which is situated in Delhi and having assets less than 1,00,000?



Q Write a SQL query to find branch name and account_no which has balance greater than equal to 1,000 but less than equal to 10,000?



(A) 127 (B) 255 (C) 129 (D) 257

<http://www.knowledgegate.in/gate>

Set Operation

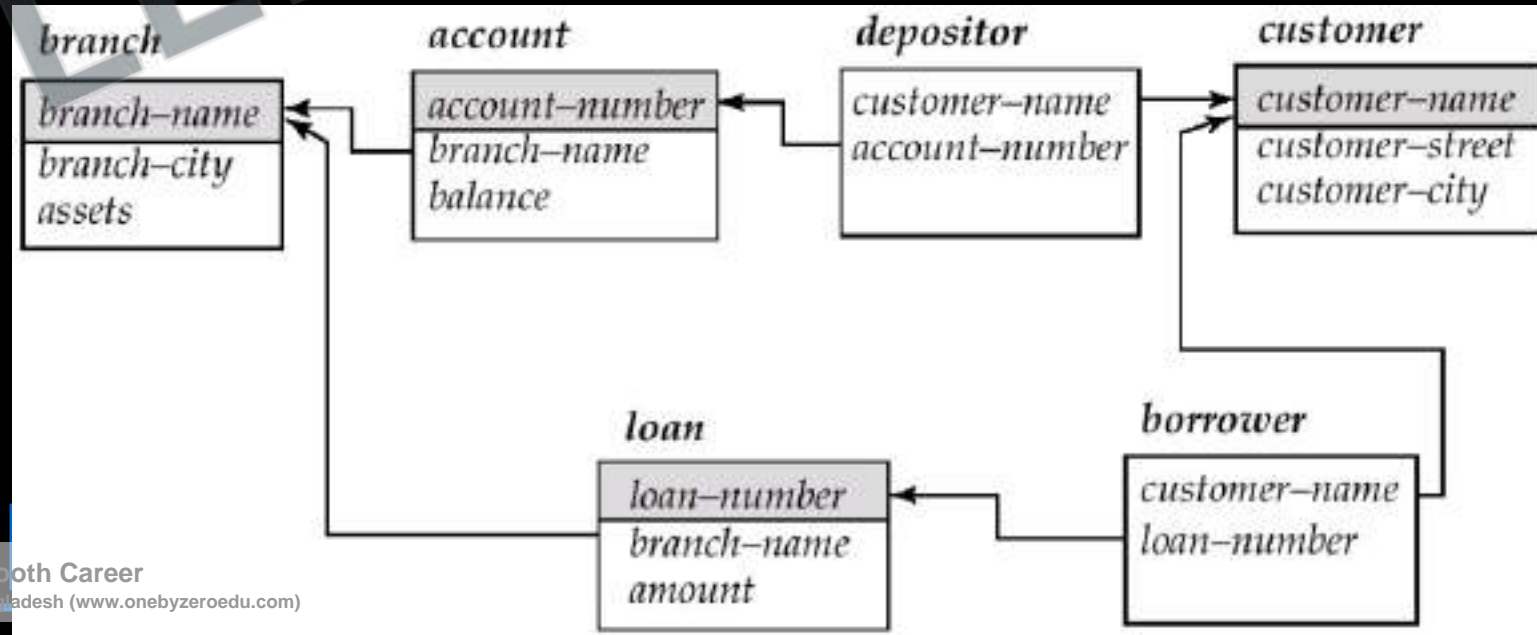
1. The SQL operations **union**, **intersect**, and **except/minus** operate on relations and corresponds to the mathematical set-theory operations \cup , \cap and $-$ respectively.
2. The **union** operation automatically eliminates duplicates, unlike the **select** clause, If we want to retain all duplicates, we must write **union all** in place of **union**.
3. The **intersect** operation automatically eliminates duplicates. If we want to retain all duplicates, we must write **intersect all** in place of intersect.
4. If we want to retain duplicates, we must write **except all** in place of **except**.

Q Write a SQL query to find all the customer name

a) who have a loan or an account or both ?

b) who have both a loan and an account?

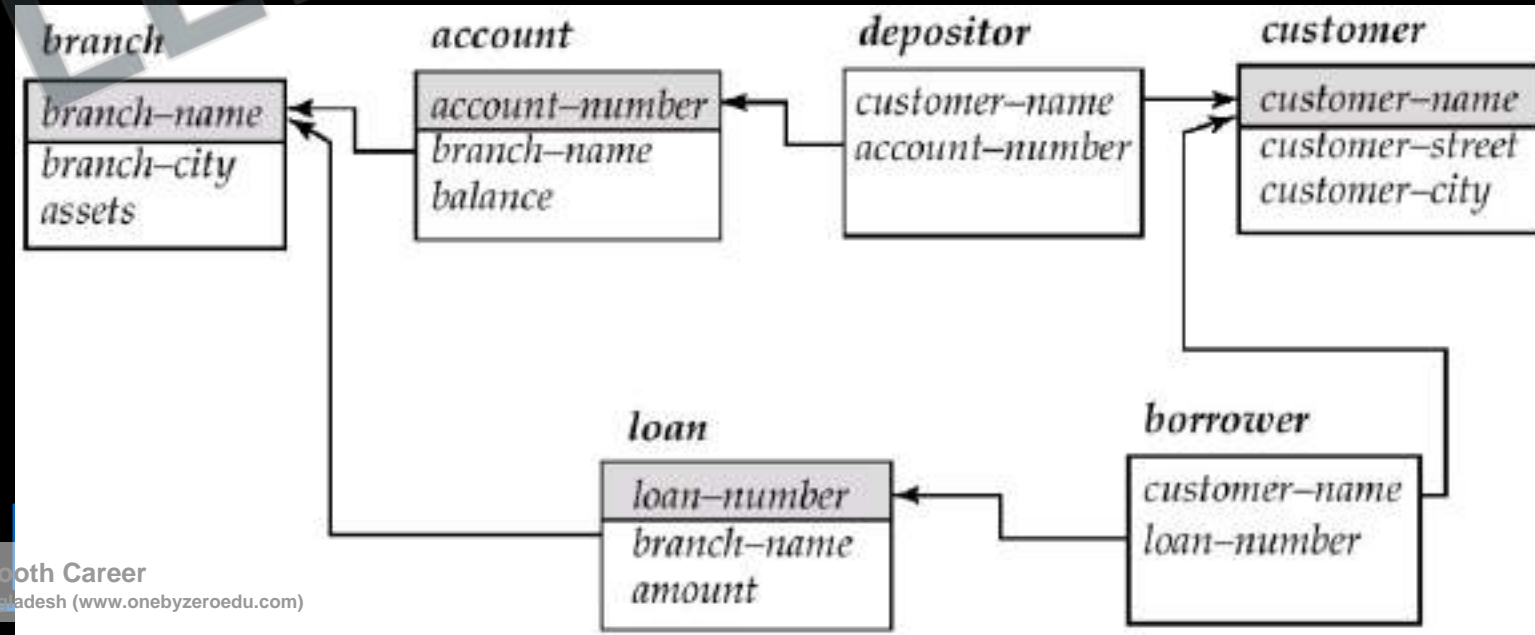
c) who have a loan but do not have an account?



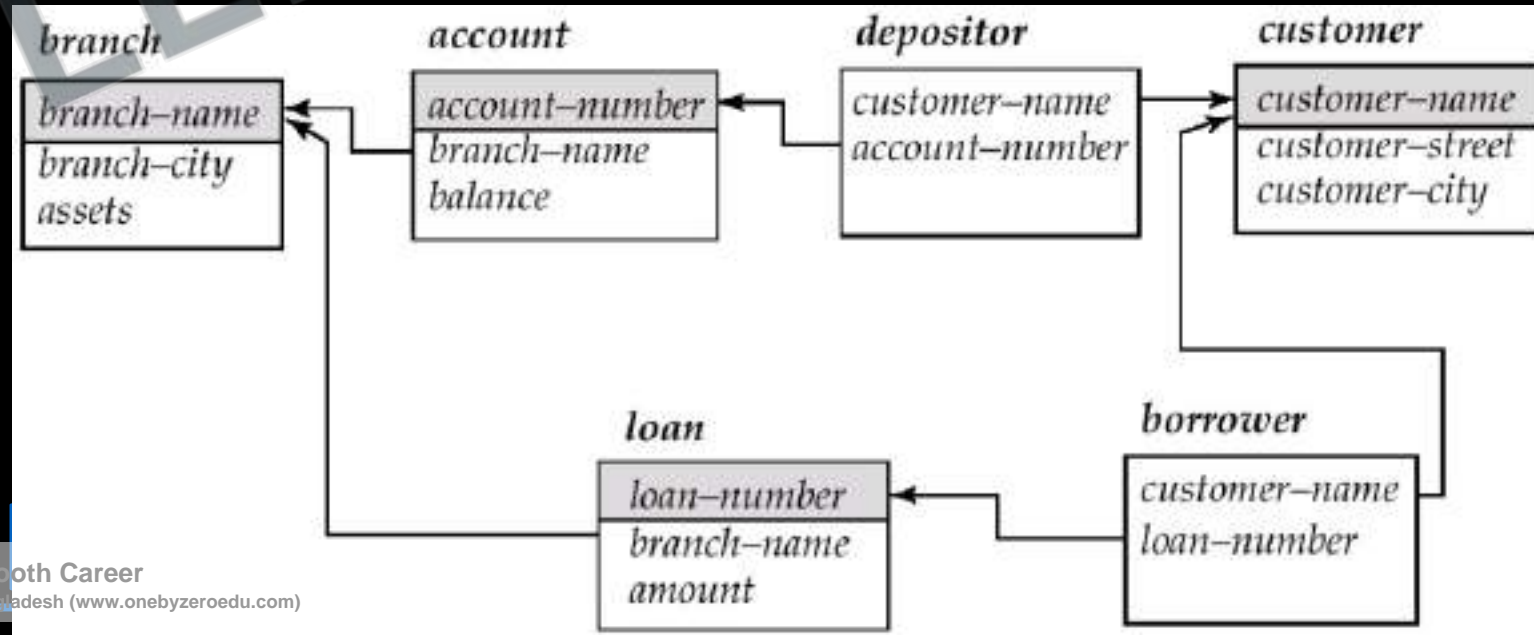
Queries on Multiple Relations

- So far, our example queries were on a single relation. Queries often need to access information from multiple relations.
- The **from** clause by itself defines a Cartesian product of the relations listed in the clause.
- Cartesian product of two relations, which concatenates each tuple of the first relation with every tuple of the second

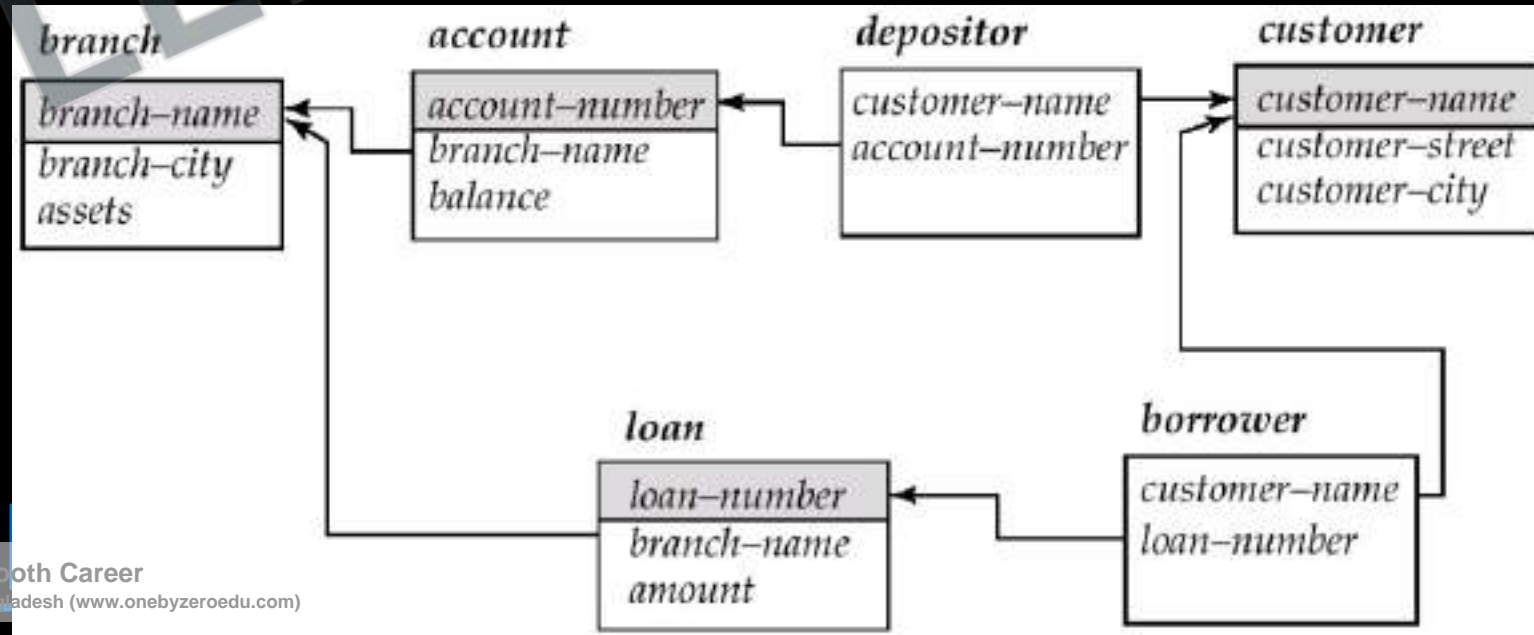
Q Write a SQL query to find the name of all the customers with account balance, who have an account in the bank?



Q Write a SQL query to find the name of all the customers, who have a loan in the bank of less than 1000 or loan from north_delhi branch?



Q Write a SQL query to find the name of the customer who have an account in the branch situated in Delhi?



Q Consider the set of relations shown below and the SQL query that follows **(Gate-2003) (2 Marks)**

Students: (Roll_number, Name, Date_of_birth)

Courses: (Course number, Course_name, Instructor)

Grades: (Roll_number, Course_number, Grade)

```
select distinct Name
from Students, Courses, Grades
where Students.Roll_number = Grades.Roll_number
and Courses.Instructor = Korth
and Courses.Course_number = Grades.Course_number
and Grades.grade = A
```

Which of the following sets is computed by the above query?

- (A)** Names of students who have got an A grade in all courses taught by Korth
- (B)** Names of students who have got an A grade in all courses
- (C)** Names of students who have got an A grade in at least one of the courses taught by Korth

(D) None of the above

<http://www.knowledgegate.in/gate>

Natural Join

1. To make the life of an SQL programmer easier for this common case, SQL supports an operation called the *natural join*.
2. The **natural join** operation like cartesian product operates on two relations and produces a relation as the result.
3. Natural join considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations.

1. Notice that we do not repeat those attributes that appear in the schemas of both relations; rather they appear only once.
2. Notice also the order in which the attributes are listed: first the attributes common to the schemas of both relations, second those attributes unique to the schema of the first relation, and finally, those attributes unique to the schema of the second relation.
3. commutative in nature

<http://www.knowledgegate.in/gate>

A **from** clause in an SQL query can have multiple relations combined using natural join, as shown here:

```
select  $A_1, A_2, \dots, A_n$   
from  $r_1$  natural join  $r_2$  natural join ... natural join  $r_m$   
where  $P$ ;
```

<http://www.knowledgegate.in/gate>

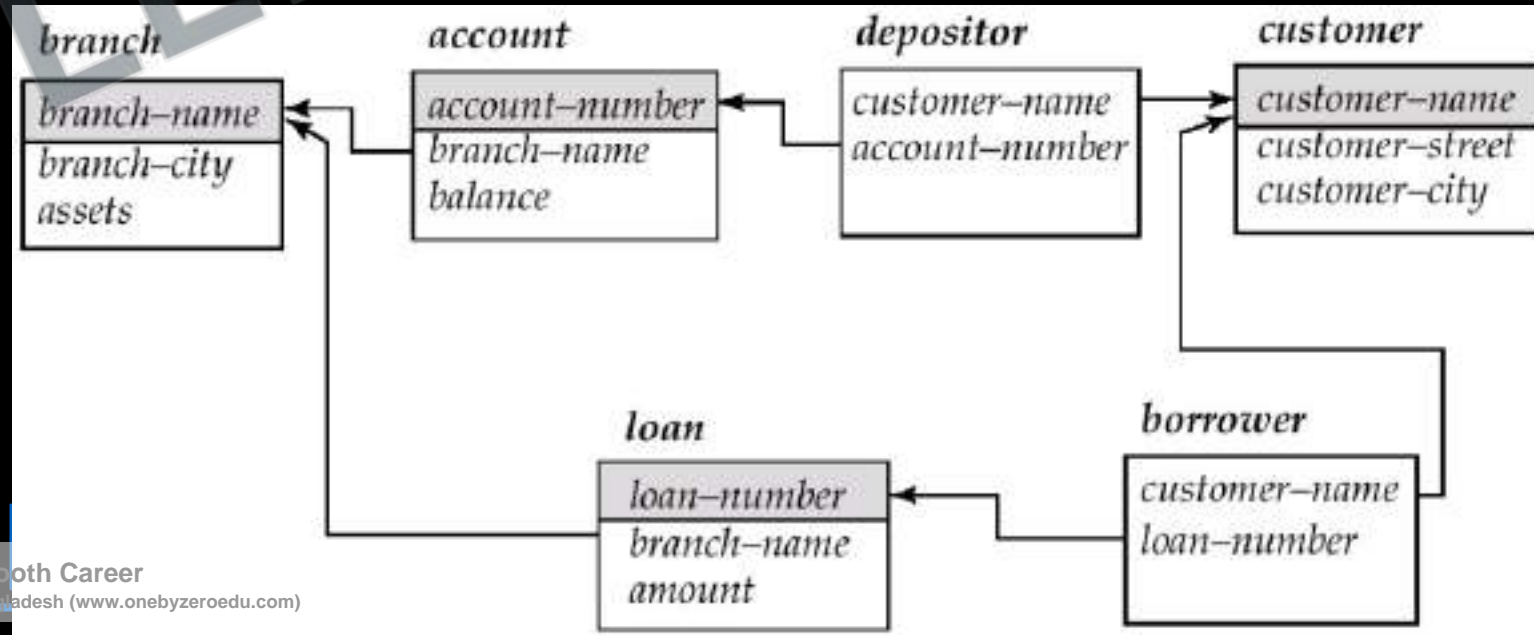
R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

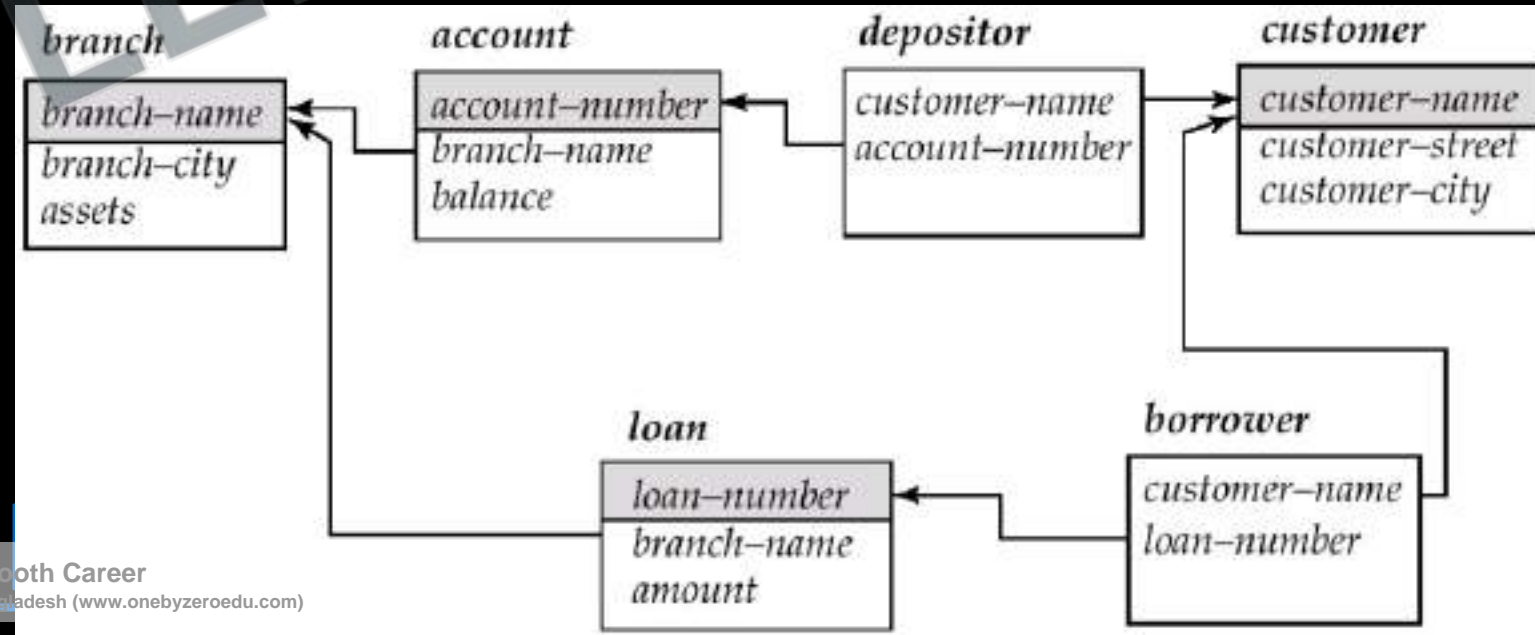
$R_1 * R_2$			
A	$R_1.B$	$R_2.B$	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

$R_1 \bowtie R_2$		
A	B	C

Q Write a SQL query to find the name of all the customers along with account balance, who have an account in the bank?



Q Write a SQL query to find the name of the customer who have an account in the branch situated in Delhi?



Q Database table by name Loan_Records is given below. (Gate - 2011) (2 Marks)

Borrower	Bank_Manager	Loan_Amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

What is the output of the following SQL query?

```
SELECT Count(*)  
FROM ( (SELECT Borrower, Bank_Manager FROM Loan_Records) AS S  
NATURAL JOIN  
(SELECT Bank_Manager, Loan_Amount FROM Loan_Records) AS T );
```

- (A) 3
- (B) 9
- (C) 5
- (D) 6

Q Consider the following relation schema pertaining to a students database:

Student (rollno, name, address)

Enroll (rollno, courseno, coursename)

where the primary keys are shown underlined. The number of tuples in the Student and Enroll tables are 120 and 8 respectively. What are the maximum and minimum number of tuples that can be present in (Student * Enroll), where '*' denotes natural join? **(Gate-2004) (2 Marks)**

(A) 8, 8

(B) 120, 8

(C) 960, 8

(D) 960, 120

Q Consider the following SQL query (GATE-2003) (1 Marks)
select distinct a_1, a_2, \dots, a_n
from r_1, r_2, \dots, r_m
where P

For an arbitrary predicate P, this query is equivalent to which of the following relational algebra expressions ?

- (A) $\Pi_{a_1, a_2, \dots, a_n} \sigma_P (r_1 \times r_2 \times \dots \times r_m)$
- (B) $\Pi_{a_1, a_2, \dots, a_n} \sigma_P (r_1 \bowtie r_2 \bowtie \dots \bowtie r_m)$
- (C) $\Pi_{a_1, a_2, \dots, a_n} \sigma_P (r_1 \cup r_2 \cup \dots \cup r_m)$
- (D) $\Pi_{a_1, a_2, \dots, a_n} \sigma_P (r_1 \cap r_2 \cap \dots \cap r_m)$

Join /inner join

1. Join or inner join is a operation which works same as cartesian product but when it is used with “using” keyword it provides additional functionality.
2. Join with using – Provides ability to explicitly chose columns which must be used by join for comparison and removal of redundant tuples, i.e. if there are more than one column which are common between two table but we do not want each of them to be considered, then we can use join with using.

1. So join is a form of the natural join construct that allows you to specify exactly which columns should be equated.
 - $R_1(A, B, C)$
 - $R_2(B, C, D)$
2. Consider the operation $R_1 \text{ join } R_2$ using (B). The operation is similar to R_1 natural join R_2 , except that a pair of tuples t_1 from R_1 and t_2 from R_2 match if $t_1.B = t_2.B$; even if R_1 and R_2 both have an attribute named C , it is *not* required that $t_1.C = t_2.C$.

Outer Join

1. The join operations we studied earlier that do not preserve nonmatched tuples are called inner join operations, to distinguish them from the outer-join operations.
2. The problem with natural join or join is only those values that appears in both relations will manage to reach final table, but if some value is explicitly in table one or in second table then that information will be lost, and that will be loss of information.
3. The outer join operation works in a manner similar to the join operations we have already studied, but preserve those tuples that would be lost in a join, by creating tuples in the result containing null values.

- There are in fact three forms of outer join:
 1. The **left outer join** (left join) preserves tuples only in the relation named before (to the left of) the left outer join operation.
 2. The **right outer join** (right join) preserves tuples only in the relation named after (to the right of) the right outer join operation.
 3. The **full outer join** preserves tuples in both relations.

R_1	
A	B
1	P
2	Q
3	R

R_2	
B	C
Q	X
R	Y
S	Z

$R_1 * R_2$			
A	$R_1.B$	$R_2.B$	C
1	P	Q	X
1	P	R	Y
1	P	S	Z
2	Q	Q	X
2	Q	R	Y
2	Q	S	Z
3	R	Q	X
3	R	R	Y
3	R	S	Z

$R_1 \bowtie R_2$		
A	B	C
2	Q	X
3	R	Y

$R_1 \bowtie R_2$		
A	B	C

$R_1 \bowtie R_2$		
A	B	C

$R_1 \bowtie R_2$		
A	B	C

Q Consider the following two tables and four queries in SQL. (GATE - 2018) (1 Marks)

Book (isbn, bname)

Stock (isbn, copies)

Query 1: SELECT B.isbn, S.copies
FROM Book B INNER JOIN Stock S
ON B.isbn = S.isbn;

Query 2: SELECT B.isbn, S.copies
FROM Book B LEFT OUTER JOIN Stock S
ON B.isbn = S.isbn;

Query 3: SELECT B.isbn, S.copies
FROM Book B RIGHT OUTER JOIN Stock S
ON B.isbn = S.isbn;

Query 4: SELECT B.isbn, S.copies
FROM Book B FULL OUTER JOIN Stock S
ON B.isbn = S.isbn;

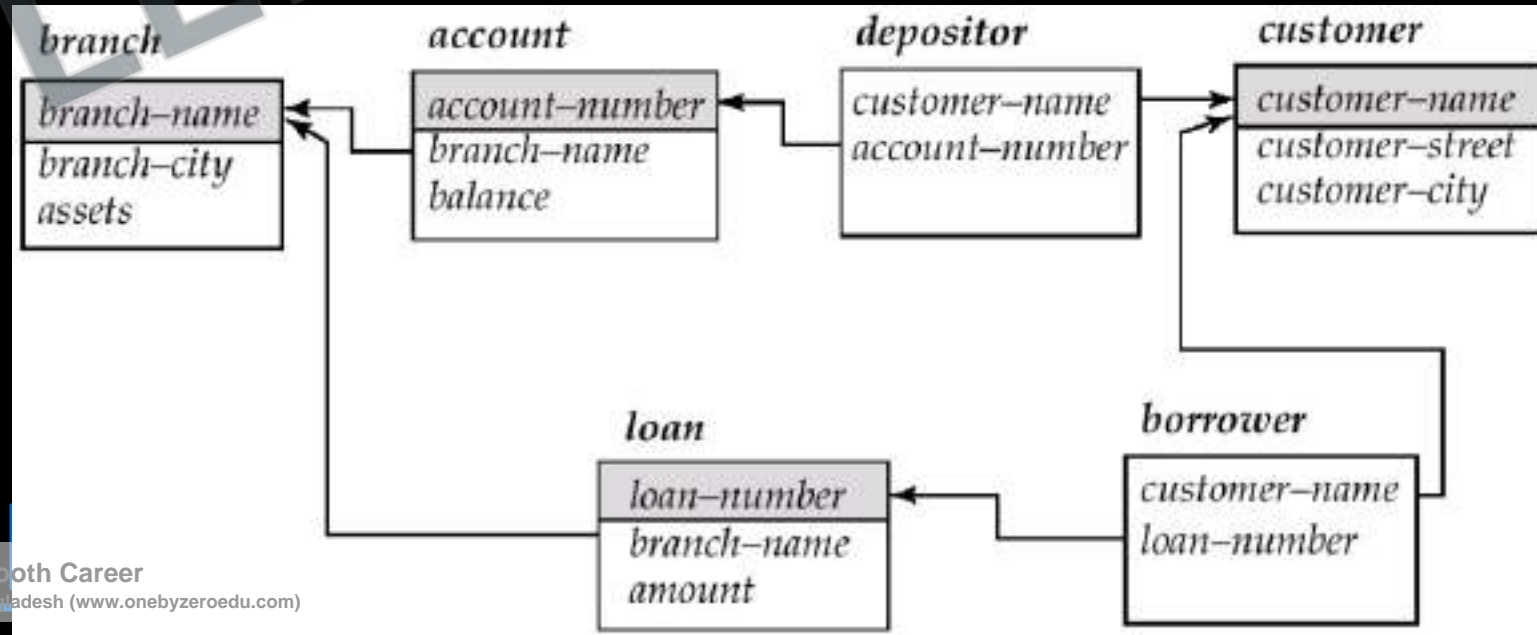
Which one of the queries above is certain to have an output that is a superset of the outputs of the other three queries?

- (a) Query 1
- (b) Query 2
- (c) Query 3
- (d) Query 4

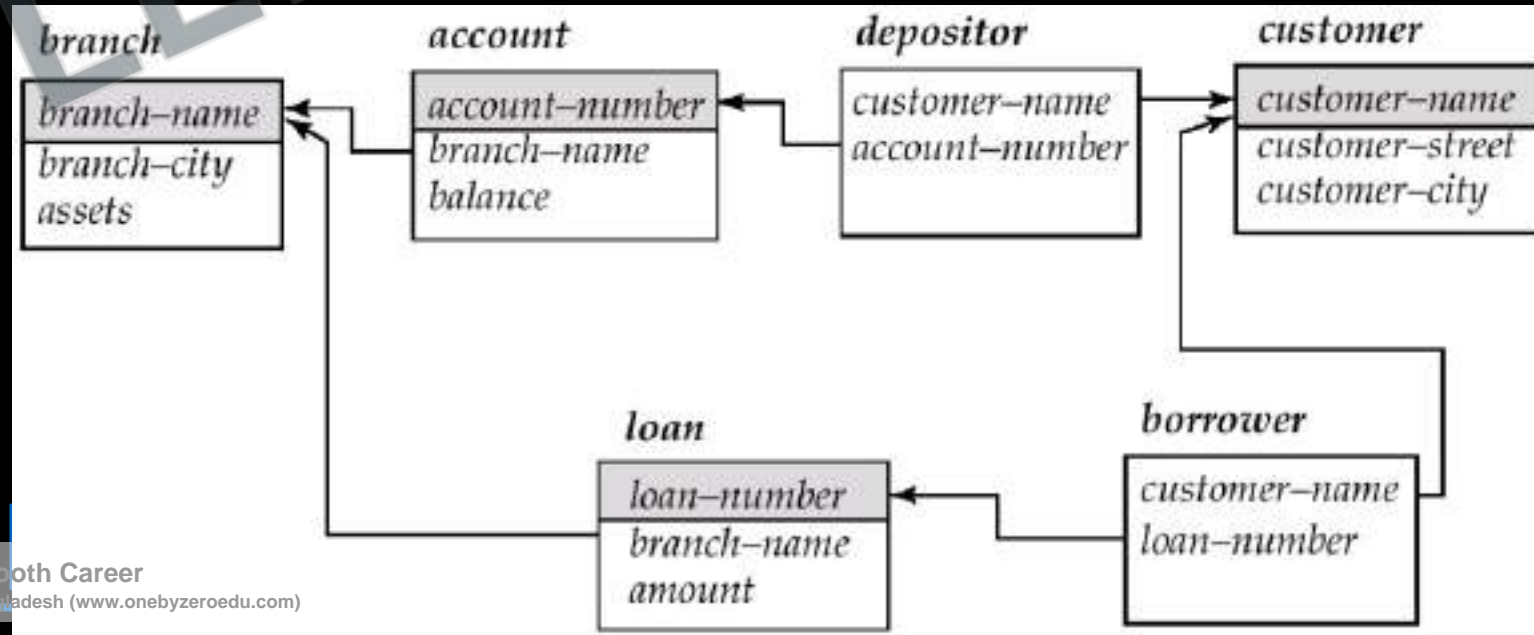
Alias Operation/rename

1. SQL aliases are used to give a table, or a column in a table, a temporary name. Just create a new copy but do not change anything in the data base.
2. Aliases are often used to make column names more readable.
3. It uses the as clause, taking the form: old-name as new-name. The as clause can appear in both the select and from clauses.
4. An alias only exists for the duration of the query.

Q Write a SQL query to find the account_no along and balance with 8% interest, as Account, total_balance?



Q Write a SQL query to find the loan_no with maximum loan amount?



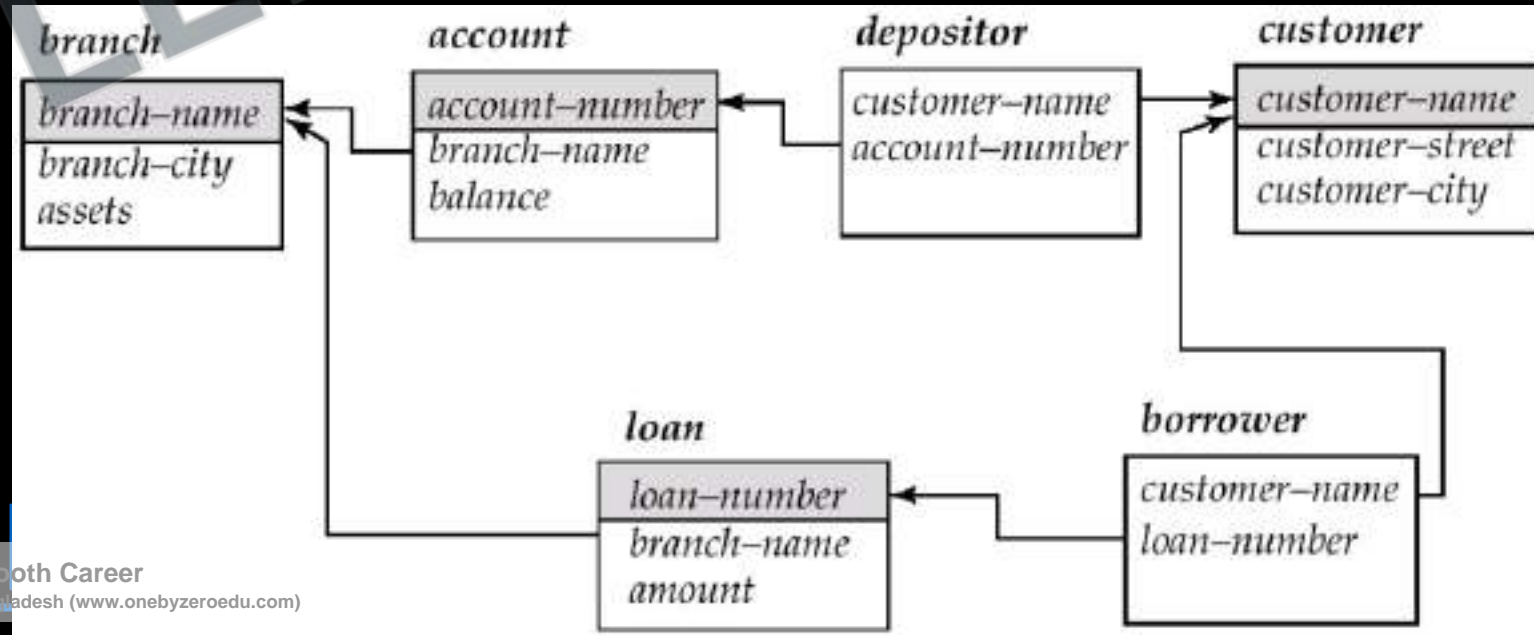
Aggregate Functions

- *Aggregate functions* are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:
 - Average: **avg**
 - Minimum: **min**
 - Maximum: **max**
 - Total: **sum**
 - Count: **count**

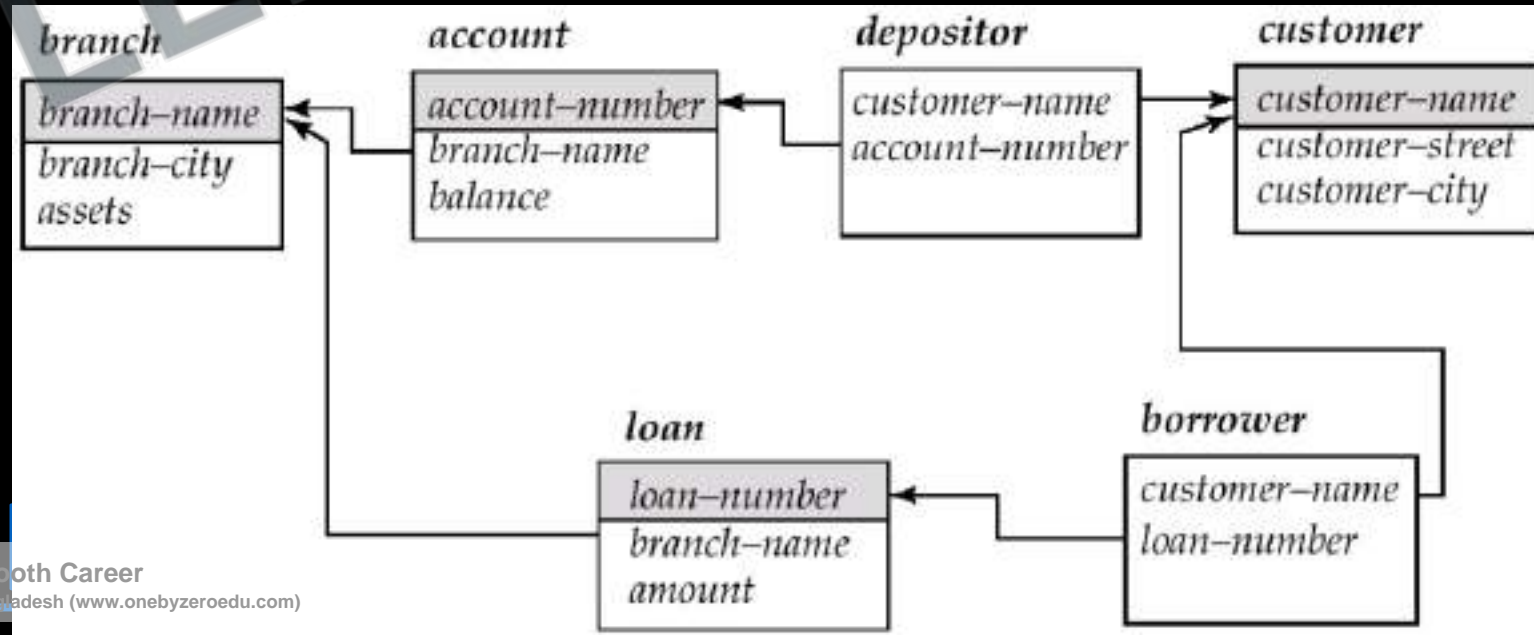
<http://www.knowledgegate.in/gate>

1. The input to **sum** and **avg** must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well.
2. We use the aggregate function **count** frequently to count the number of tuples in a relation. The notation for this function in SQL is **count (*)**.
3. Count is the only aggregate function which can work with null, all other aggregate functions simply ignore null.

Q Write a SQL query to find the number of accounts in the bank?



Q Write a SQL query to find the average balance of every account in the banks from south_delhi branch?



Q Consider a table along with two query?

**Select avg (balance)
from account**

Account_no	balance	Branch_name
Abc123	100	N_delhi
Pqr123	500	S_mumbai
Wyz123	null	S_delhi

**Select sum(balance)/count(balance)
from account**

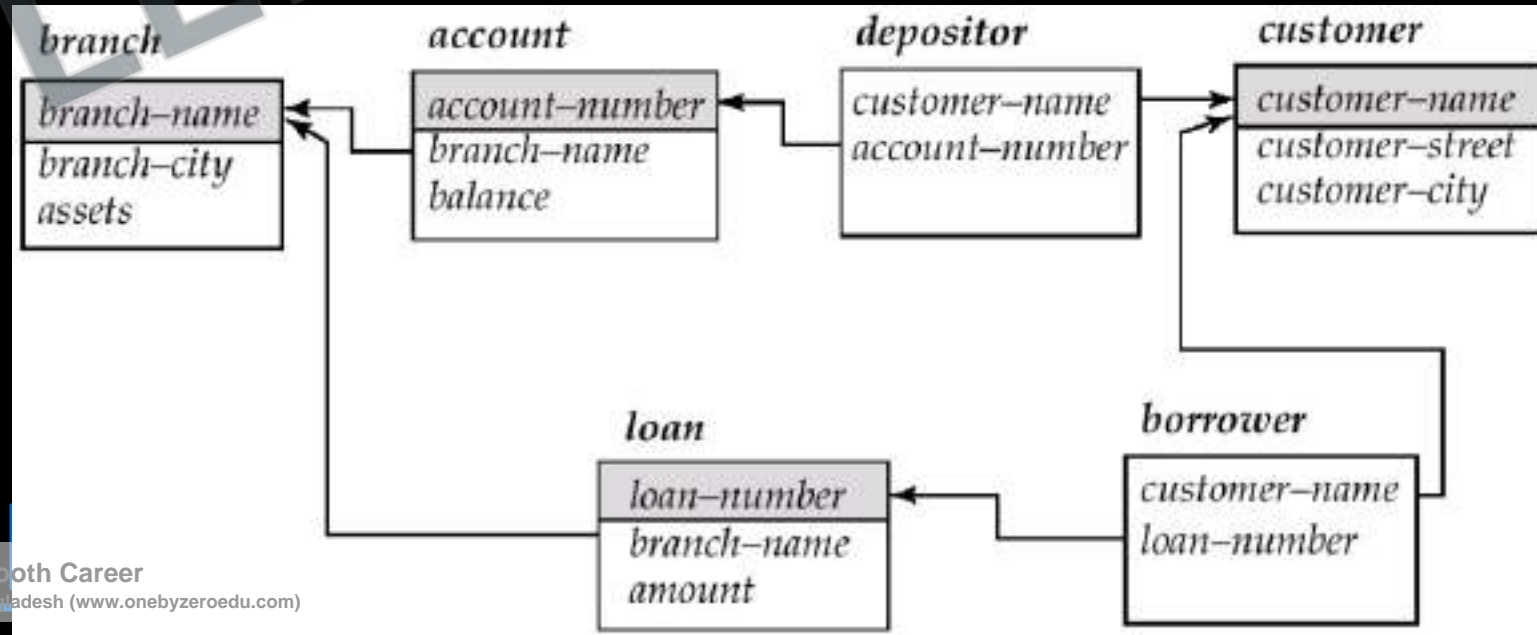
<http://www.knowledgegate.in/gate>

Ordering the Display of Tuples

- SQL offers the user some control over the order in which tuples in a relation are displayed. The **order by** clause causes the tuples in the result of a query to appear in sorted order.

<http://www.knowledgegate.in/gate>

Q Write a SQL query to find all the branch_name which are situated in Delhi in alphabetic order?



String Operations

1. SQL specifies strings by enclosing them in single quotes, for example, 'Computer'.
2. The SQL standard specifies that the equality operation on strings is case sensitive; as a result the expression 'Computer' = 'computer' evaluates to false.
3. However, some database systems, such as MySQL and SQL Server, do not distinguish uppercase from lowercase when matching strings; as a result, would evaluate to true on these databases.
4. This default behavior can, however, be changed, either at the database level or at the level of specific attributes.

1. SQL also permits a variety of functions on character strings, such as concatenating, extracting substrings, finding the length of strings, converting strings to uppercase and lowercase, removing spaces at the end of the string and so on.
2. There are variations on the exact set of string functions supported by different database systems.
3. A single quote character that is part of a string can be specified by using two single quote characters; for example, the string It's right can be specified by It''s right.

<http://www.knowledgegate.in/gate>

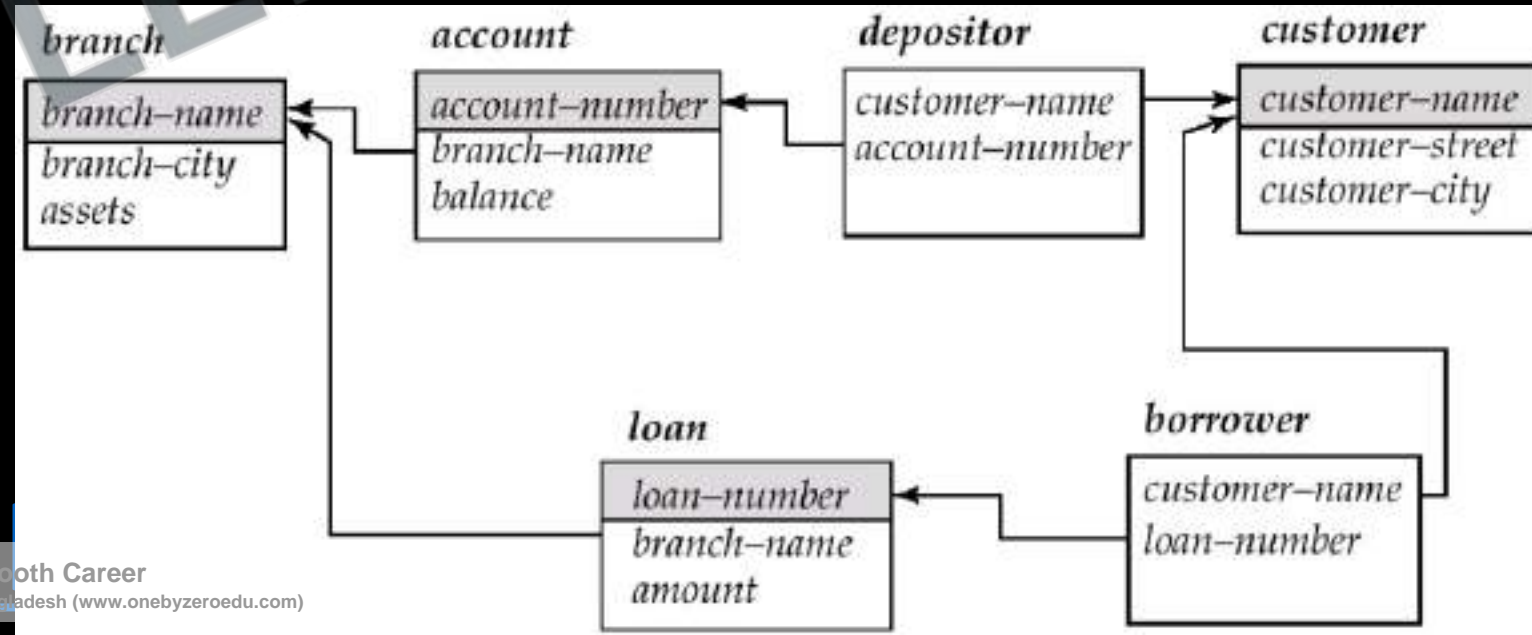
- Pattern matching can be performed on strings, using the operator **like**. We describe patterns by using two special characters:
 - Percent (%): The % character matches any substring.
 - Underscore (_): The _ character matches any character.

<http://www.knowledgegate.in/gate>

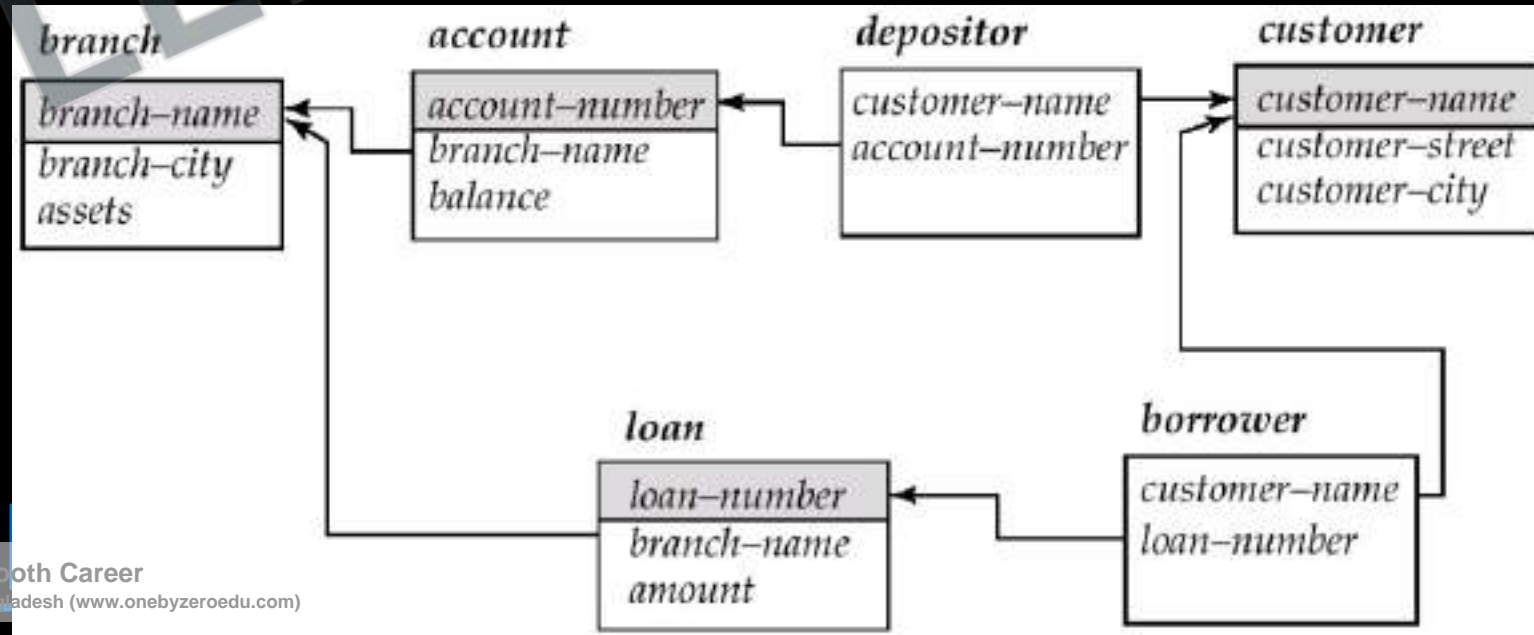
- '%Comp%' matches any string containing "Comp" as a substring, for example, 'Intro to Computer Science', and 'Computational Biology'.
- '___' matches any string of exactly three characters.
- '___%' matches any string of at least three characters.

<http://www.knowledgegate.in/gate>

Q Write a SQL query to find all the branch name who have exactly 5 character in their name ?



Q Write a SQL query to find all the customer name who have 'kumar' in their name ?



1. For patterns to include the special pattern characters (that is, % and _), SQL allows the specification of an escape character.
2. The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated like a normal character.
3. We define the escape character for a **like** comparison using the **escape** keyword. To illustrate, consider the following patterns, which use a backslash (\) as the escape character:
4. **like** 'ab\%cd%' **escape** '\' matches all strings beginning with "ab%cd".
5. **like** 'ab\\cd%' **escape** '\' matches all strings beginning with "ab\cd".

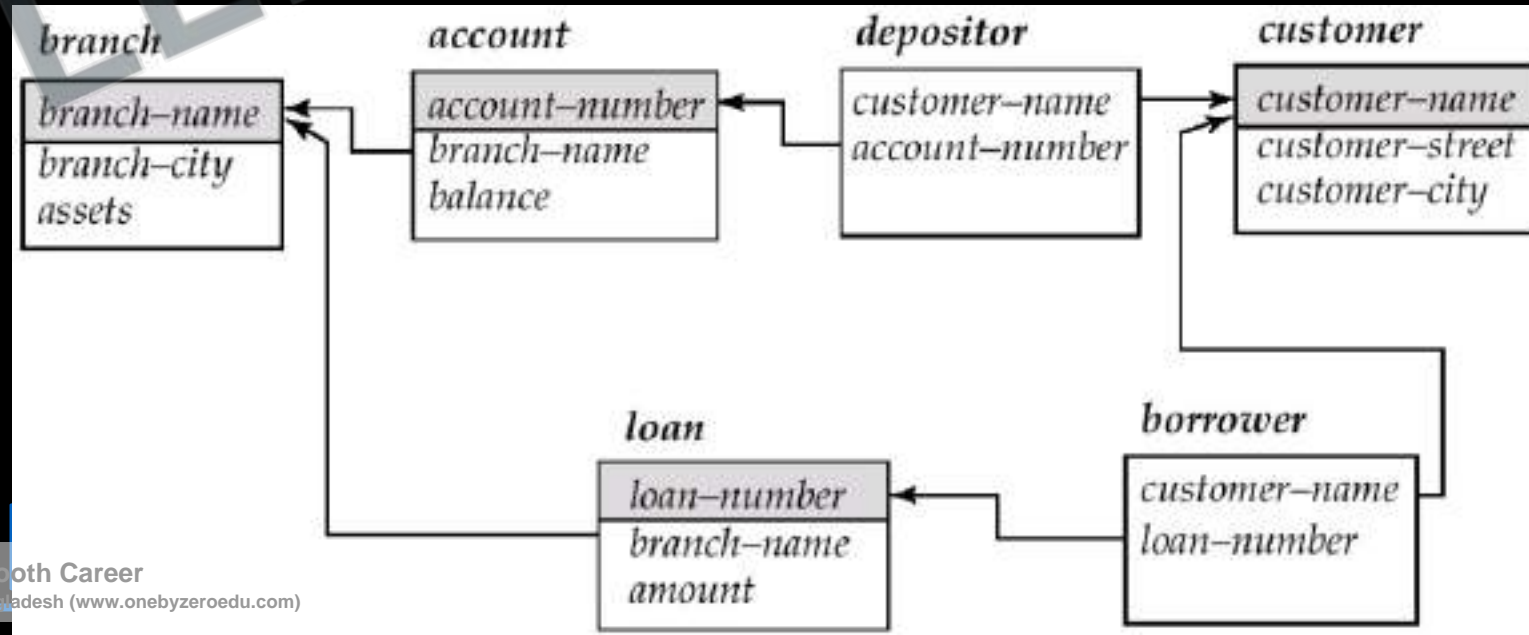
1. SQL allows us to search for mismatches instead of matches by using the **not like** comparison operator. Some databases provide variants of the **like** operation which do not distinguish lower and upper case.
2. SQL:1999 also offers a **similar to** operation, which provides more powerful pattern matching than the **like** operation; the syntax for specifying patterns is similar to that used in Unix regular expressions.

<http://www.knowledgegate.in/gate>

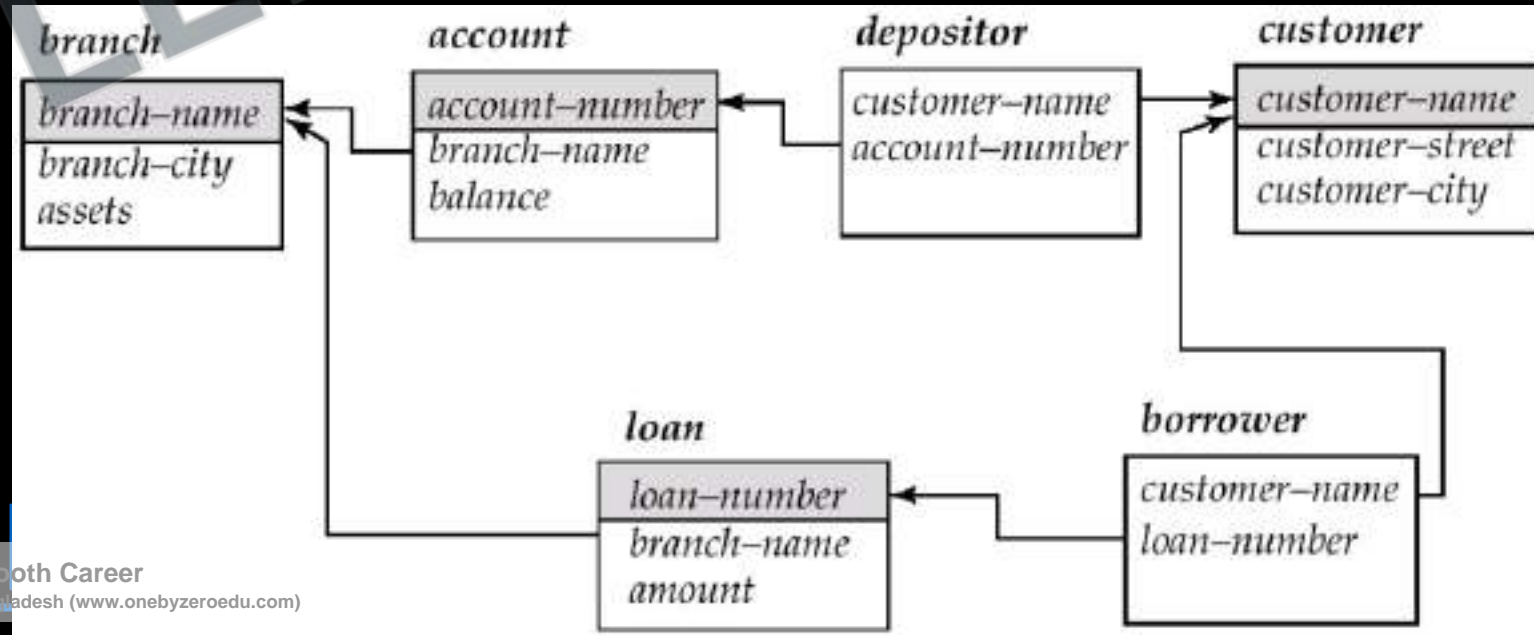
Group by clause

1. There are circumstances where we would like to work on a set of tuples in a relation rather than working on the whole table as one unit.
2. The attribute or attributes given in the **group by** clause are used to form groups. Tuples with the same value on all attributes in the **group by** clause are placed in one group.

Q Write a SQL query to find the average account balance of each branch?



Q Write a SQL query to find the branch name of Gwalior city with average balance more than 1500?



Q A relational database contains two tables Student and Performance as shown below: (GATE - 2019) (2 Marks)

The primary key of the Student table is Roll_no. For the Performance table, the columns Roll_no. and Subject_code together form the primary key. Consider the SQL query given below:

```
SELECT S.Student_name, sum (P.Marks)
FROM Student S, Performance P
WHERE P.Marks > 84
GROUP BY S.Student_name;
```

The number of rows returned by the above SQL query is _____.

Student		Performance		
Roll_no.	Student_name	Roll_no.	Subject_code	Marks
1	Amit	1	A	86
2	Priya	1	B	95
3	Vinit	1	C	90
4	Rohan	2	A	89
5	Smita	2	C	92
		3	C	80

Q Consider a database that has the relation schema EMP (EmpId, EmpName, and DeptName). An instance of the schema EMP and a SQL query on it are given below. (GATE-2018) (2 Marks)

EMP		
EmpId	EmpName	DeptName
1	XYA	AA
2	XYB	AA
3	XYC	AA
4	XYD	AA
5	XYE	AB
6	XYF	AB
7	XYG	AB
8	XYH	AC
9	XYI	AC
10	XYJ	AC
11	XYK	AD
12	XYL	AD
13	XYM	AE

```
SELECT AVG(EC. Num)
FROM EC
WHERE (DeptName, Num) IN
      (SELECT DeptName, COUNT(EmpId) AS
        EC(DeptName, Num)
       FROM EMP
        GROUP BY DeptName)
```

The output of executing the SQL query is _____

Q The employee information in a company is stored in the relation

Employee (name, sex, salary, deptName)

Consider the following SQL query

```
select deptName  
from Employee  
where sex = 'M'  
group by deptName  
having avg (salary) > (select avg (salary) from Employee)
```

It returns the names of the department in which **(Gate-2004) (2 Marks) (NET-JUNE-2013)**

- (A)** the average salary is more than the average salary in the company
- (B)** the average salary of male employees is more than the average salary of all male employees in the company
- (C)** the average salary of male employees is more than the average salary of employees in the same department
- (D)** the average salary of male employees is more than the average salary in the company

<http://www.knowledgegate.in/gate>

- The SQL WITH clause was introduced by Oracle in the Oracle 9i release 2 database. The SQL WITH clause allows you to give a sub-query block a name (a process also called sub-query refactoring), which can be referenced in several places within the main SQL query.
- The clause is used for defining a temporary relation such that the output of this temporary relation is available and is used by the query that is associated with the WITH clause.
- WITH clause is not supported by all database system.

Q Consider the following database table named water_schemes :
The number of tuples returned by the following SQL query is _____. **GATE(2015 SET 1)**

```
with total(name, capacity) as  
  select district_name, sum(capacity)  
  from water_schemes  
  group by district_name  
with total_avg(capacity) as  
  select avg(capacity)  
  from total  
select name  
  from total, total_avg  
  where total.capacity ≥ total_avg.capacity
```

water – schemes		
scheme_no	district_name	capacity
1	Ajmer	20
1	Bikaner	10
2	Bikaner	10
3	Bikaner	20
1	Churu	10
2	Churu	20
1	Dungargarh	10

<http://www.knowledgegate.in/gate>

1. When an SQL query uses grouping, it is important to ensure that the only attributes that appear in the **select** statement without being aggregated are those that are present in the **group by** clause.
2. In other words, any attribute that is not present in the **group by** clause must appear only inside an aggregate function if it appears in the **select** clause, otherwise the query is treated as erroneous.
3. The Having Clause - At times, it is useful to state a condition that applies to groups rather than to tuples.
4. To express such a query, we use the having clause of SQL. SQL applies predicates in the having clause after groups have been formed.
5. Any attribute that is present in the **having** clause without being aggregated must appear in the **group by** clause, otherwise the query is treated as erroneous.

- The meaning of a query containing aggregation, **group by**, or **having** clauses is defined by the following sequence of operations:
 1. As was the case for queries without aggregation, the **from** clause is first evaluated to get a relation.
 2. If a **where** clause is present, the predicate in the **where** clause is applied on the result relation of the **from** clause.
 3. Tuples satisfying the **where** predicate are then placed into groups by the **group by** clause if it is present. If the **group by** clause is absent, the entire set of tuples satisfying the **where** predicate is treated as being in one group.
 4. The **having** clause, if it is present, is applied to each group; the groups that do not satisfy the **having** clause predicate are removed.
 5. The **select** clause uses the remaining groups to generate tuples of the result of the query, applying the aggregate functions to get a single result tuple for each group.

Q Which of the following statements are TRUE about an SQL query? **(Gate-2012) (2 Marks)**

P : An SQL query can contain a HAVING clause even if it does not have a GROUP BY clause

Q : An SQL query can contain a HAVING clause only if it has a GROUP BY clause

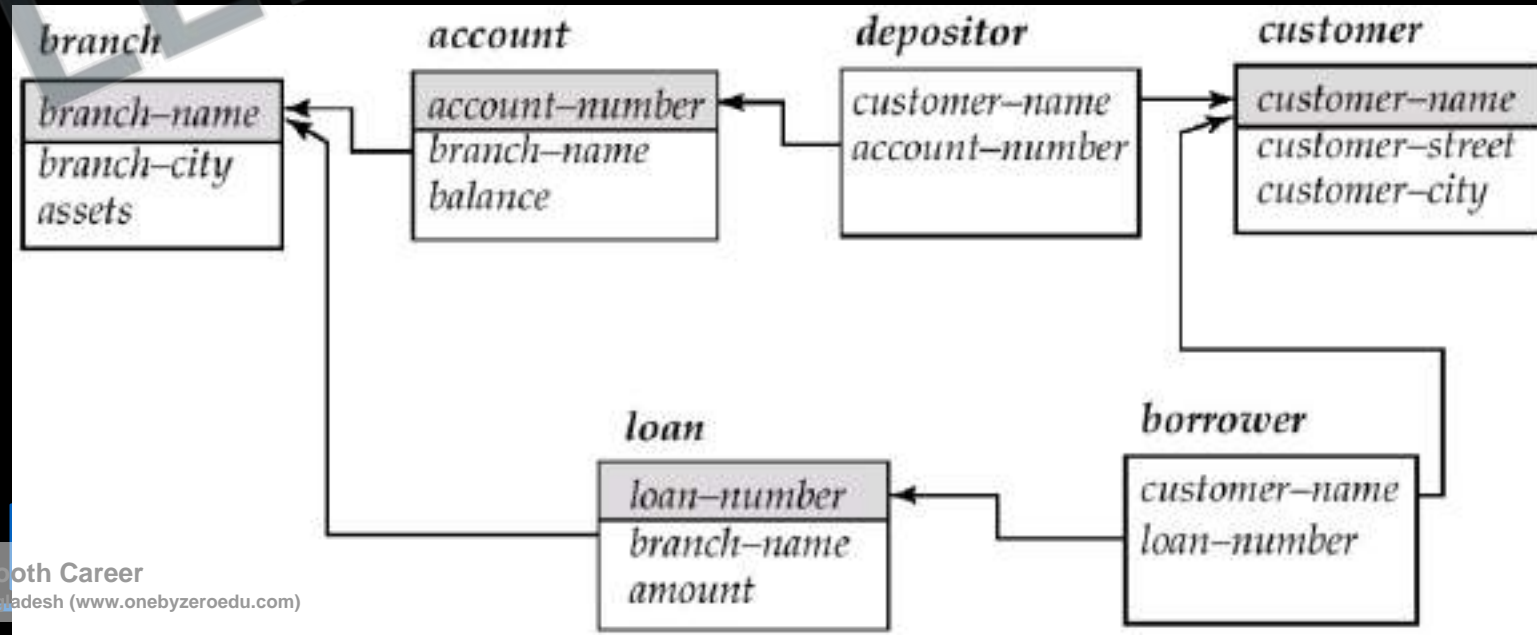
R : All attributes used in the GROUP BY clause must appear in the SELECT clause

S : Not all attributes used in the GROUP BY clause need to appear in the SELECT clause

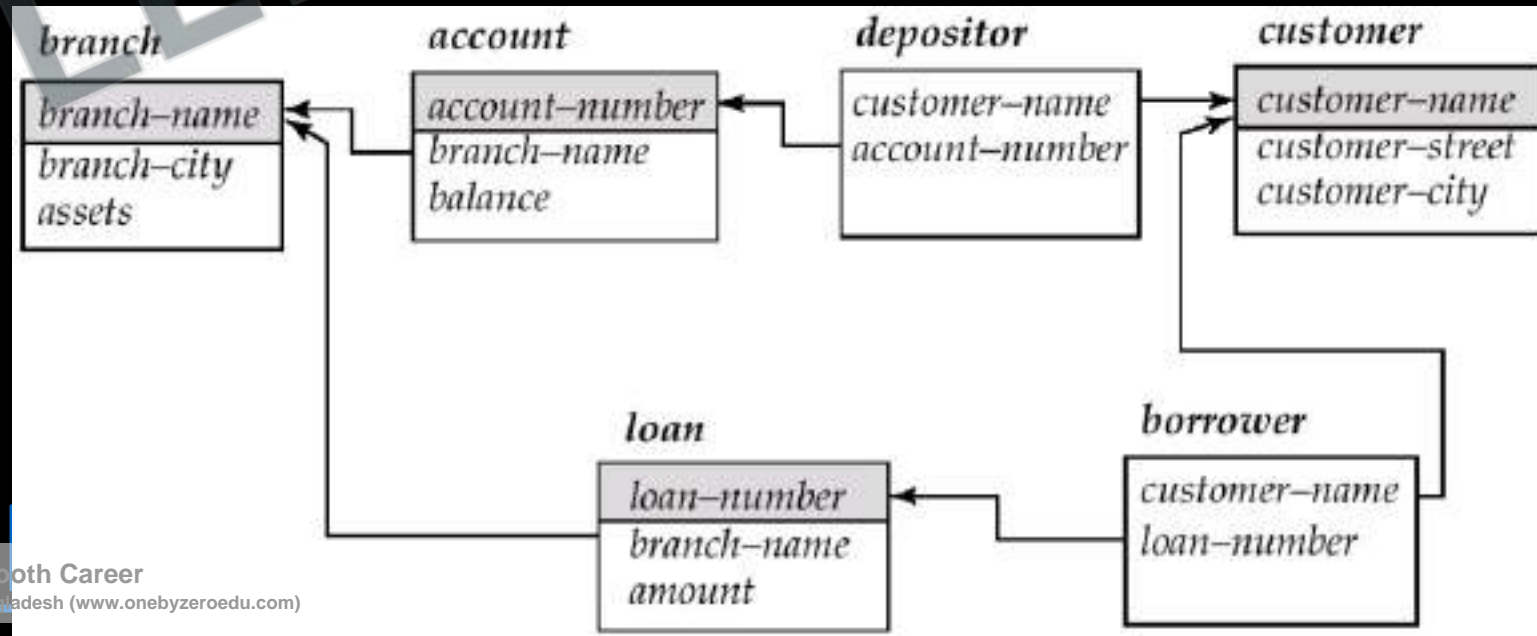
(A) P and R **(B)** P and S **(C)** Q and R **(D)** Q and S

<http://www.knowledgegate.in/gate>

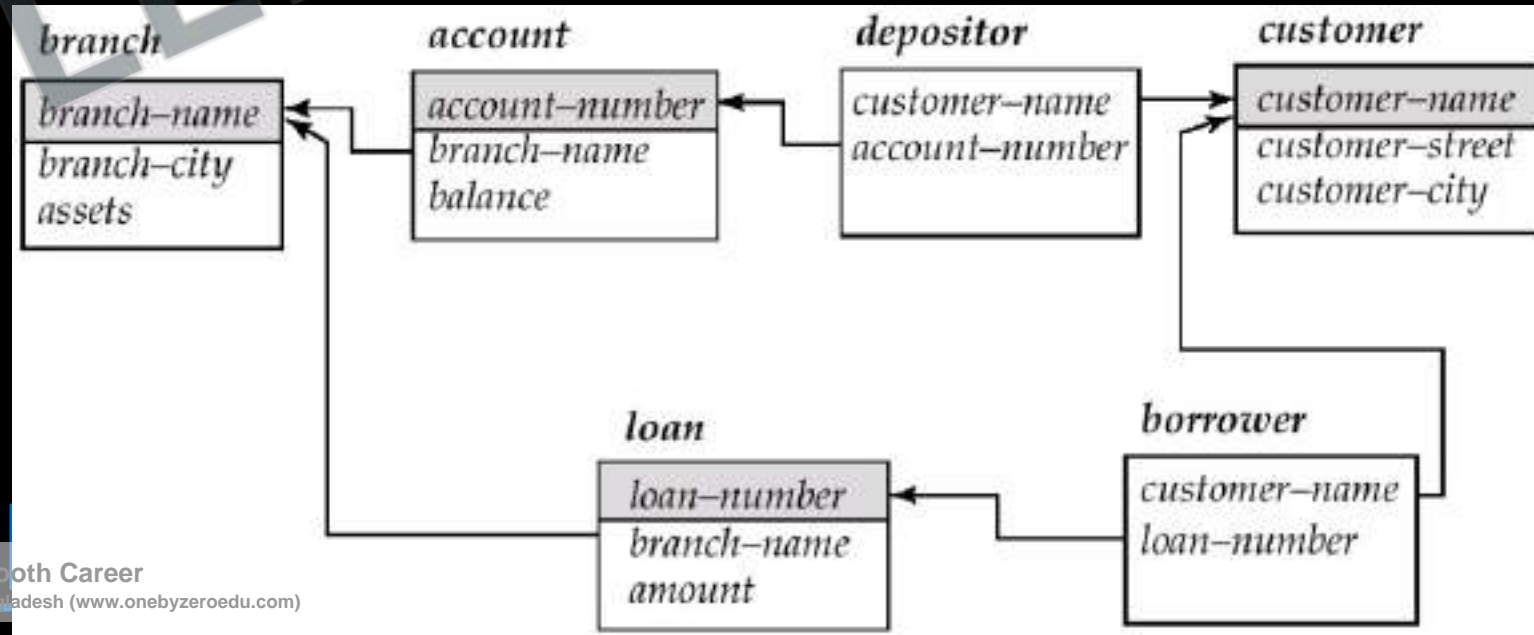
Q Write a SQL query to find all the customer name who have a loan and account?



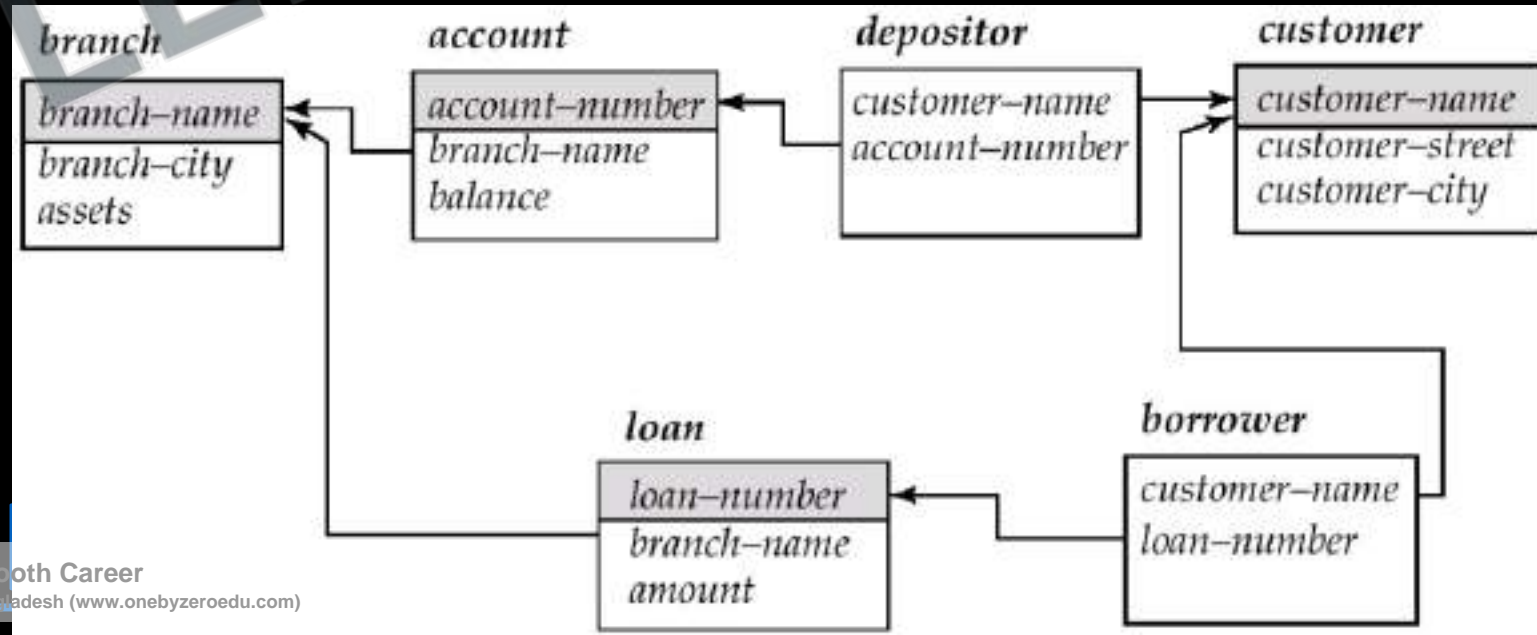
Q Write a SQL query to find all the customer name who have a loan but do not have an account?



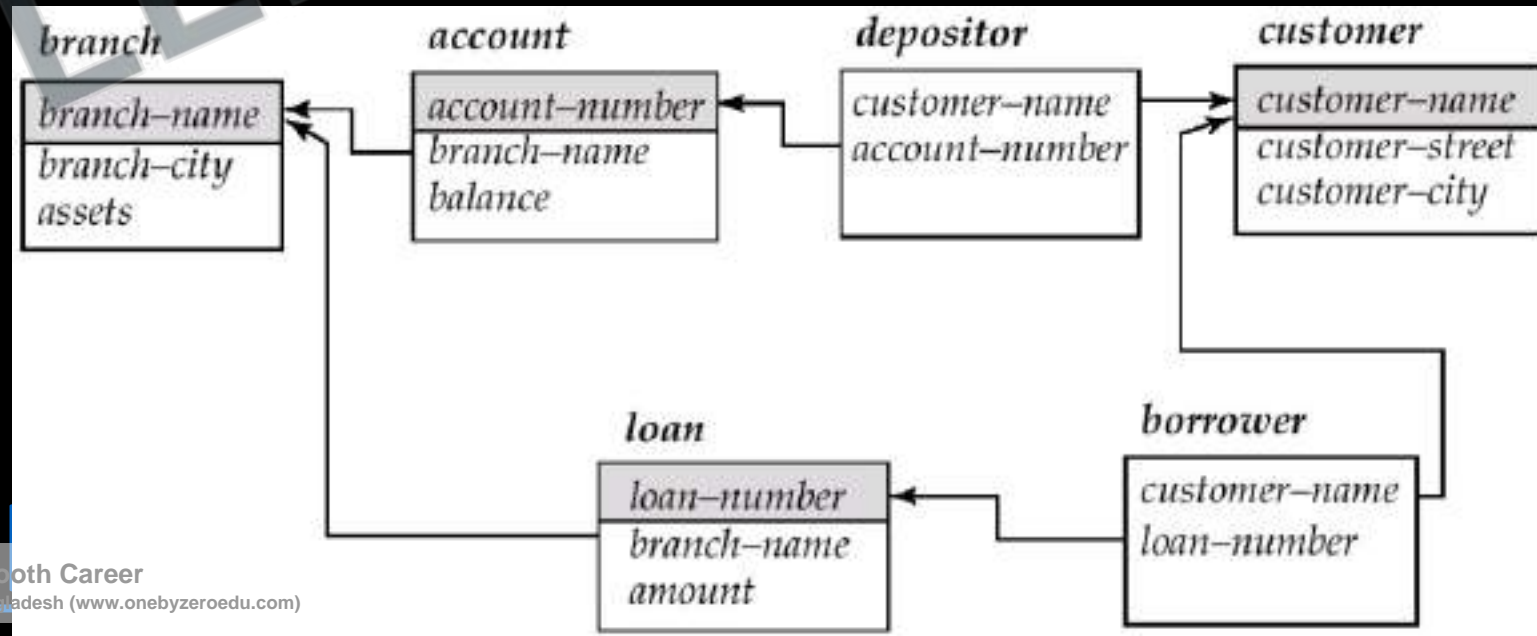
Q Write a SQL query to find the largest loan amount in the bank?



Q Write a SQL query to find loan amount which is not smallest?



Q Write a SQL query to find account no, which has balance greater than every account of north Delhi branch?



Q Consider the following relations A, B, C. How many tuples does the result of the following relational algebra expression contain? Assume that the schema of A U B is the same as that of A. (Gate-2012) (2 Marks)

A		
ID	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

B		
ID	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

C		
ID	Phone	Area
10	2200	02
99	2100	01

```
SELECT A.id  
FROM A  
WHERE A.age > ALL (SELECT B.age  
FROM B  
WHERE B.name = "arun")
```

(A) 4
(C) 0

(B) 3
(D) 1

<http://www.knowledgegate.in/gate>

Q The relation book (title, price) contains the titles and prices of different books. Assuming that no two books have the same price, what does the following SQL query list? **(Gate - 2005) (2 Marks)** [Asked in TCS NQT 2020]

```
select title
from book as B
where (select count(*)
      from book as T
      where T.price > B.price) < 5
```

- (A)** Titles of the four most expensive books
- (B)** Title of the fifth most inexpensive book
- (C)** Title of the fifth most expensive book
- (D)** Titles of the five most expensive books

<http://www.knowledgegate.in/gate>

Q Consider the following relation (**GATE-2015**) (**2 Marks**)

Cinema (theater, address, capacity)

Which of the following options will be needed at the end of the SQL query

SELECT P₁.address FROM Cinema P₁

Such that it always finds the addresses of theaters with maximum capacity?

(A) WHERE P₁. Capacity > = All (select P₂. Capacity from Cinema P₂)

(B) WHERE P₁. Capacity > = Any (select P₂. Capacity from Cinema P₂)

(C) WHERE P₁. Capacity > All (select max(P₂. Capacity) from Cinema P₂)

(D) WHERE P₁. Capacity > Any (select max (P₂. Capacity) from Cinema P₂)

<http://www.knowledgegate.in/gate>

Q Given the following schema (**GATE-2014**) (2 Marks)

employees(emp-id, first-name, last-name, hire-date, dept-id, salary)

departments(dept-id, dept-name, manager-id, location-id)

You want to display the last names and hire dates of all latest hires in their respective departments in the location ID 1700. You issue the following query:

```
SELECT last-name, hire-date
```

```
FROM employees
```

```
WHERE (dept-id, hire-date) IN (SELECT dept-id, MAX(hire-date)
```

```
FROM employees JOIN departments USING(dept-id)
```

```
WHERE location-id = 1700
```

```
GROUP BY dept-id);
```

What is the outcome?

(A) It executes but does not give the correct result.

(B) It executes and gives the correct result.

(C) It generates an error because of pairwise comparison.

(D) It generates an error because the GROUP BY clause cannot be used with table joins in a subquery

Test for Empty Relations

```
Select branch_name  
from branch as a  
Where assets <= 1,00,000 and  
exists(Select *
```

```
    from Branch as b  
    where branch_city = 'Gwalior'  
    and a.branch_name = b.branch_name)
```

- SQL includes a feature for testing whether a subquery has any tuples in its result. The exists construct returns the value true if the argument subquery is nonempty.
- The above query also illustrates a feature of SQL where a correlation name from an outer query, can be used in a subquery in the where clause. A subquery that uses a correlation name from an outer query is called a correlated subquery.

Q Consider the following relational schema (**Gate-2014**) (**2 Marks**)

employee(empId, empName, empDept)

customer(custId, custName, salesRepId, rating)

salesRepId is a foreign key referring to empId of the employee relation. Assume that each employee makes a sale to at least one customer. What does the following query return?

```
SELECT empName
```

```
FROM employee E
```

```
WHERE NOT EXISTS (SELECT custId
```

```
FROM customer C
```

```
WHERE C.salesRepId = E.EmpId AND C.rating <> 'GOOD');
```

- (a) Names of all the employees with at least one of their customers having a 'GOOD' rating.
- (b) Names of all the employees with at most one of their customers having a 'GOOD' rating.
- (c) Names of all the employees with none of their customers having a 'GOOD' rating.
- (d) Names of all the employees with all their customers having a 'GOOD' rating.

Q A relational schema for a train reservation database is given below.

Passenger (pid, pname, age)

Reservation (pid, class, tid)

What pids are returned by the following SQL query for the above instance of the tables? **(Gate-2010) (2 Marks)**

SELECT pid

FROM Reservation

WHERE class 'AC' AND EXISTS (SELECT *

FROM Passenger

WHERE age > 65 AND Passenger. pid = Reservation.pid)

(A) 1, 0

(B) 1, 2

(C) 1, 3

(D) 1, 5

Passenger		
pid	pname	age
0	Sachin	65
1	Rahul	66
2	Sourav	67
3	Anil	69

Reservation		
pid	class	tid
0	AC	8200
1	AC	8201
2	SC	8201
5	AC	8203
1	SC	8204
3	AC	8202

Q Consider the following Employee table (Gate-2015) (2 Marks)

How many rows are there in the result of following query?

SELECT E.ID

FROM Employee E

WHERE EXISTS (SELECT E2.salary
FROM Employee E2
WHERE E2.DeptName = 'CS' AND E.salary > E2.salary)

(A) 0

(B) 4

(C) 5

(D) 6

ID	salary	DeptName
1	10000	EC
2	40000	EC
3	30000	CS
4	40000	ME
5	50000	ME
6	60000	ME
7	70000	CS

null

<http://www.knowledgegate.in/gate>

Q In SQL, relations can contain null values, and comparisons with null values are treated as unknown. Suppose all comparisons with a null value are treated as false. Which of the following pairs is not equivalent? **(Gate-2000) (2 Marks)**

(A) $x = 5$ AND $\text{not}(\text{not}(x = 5))$

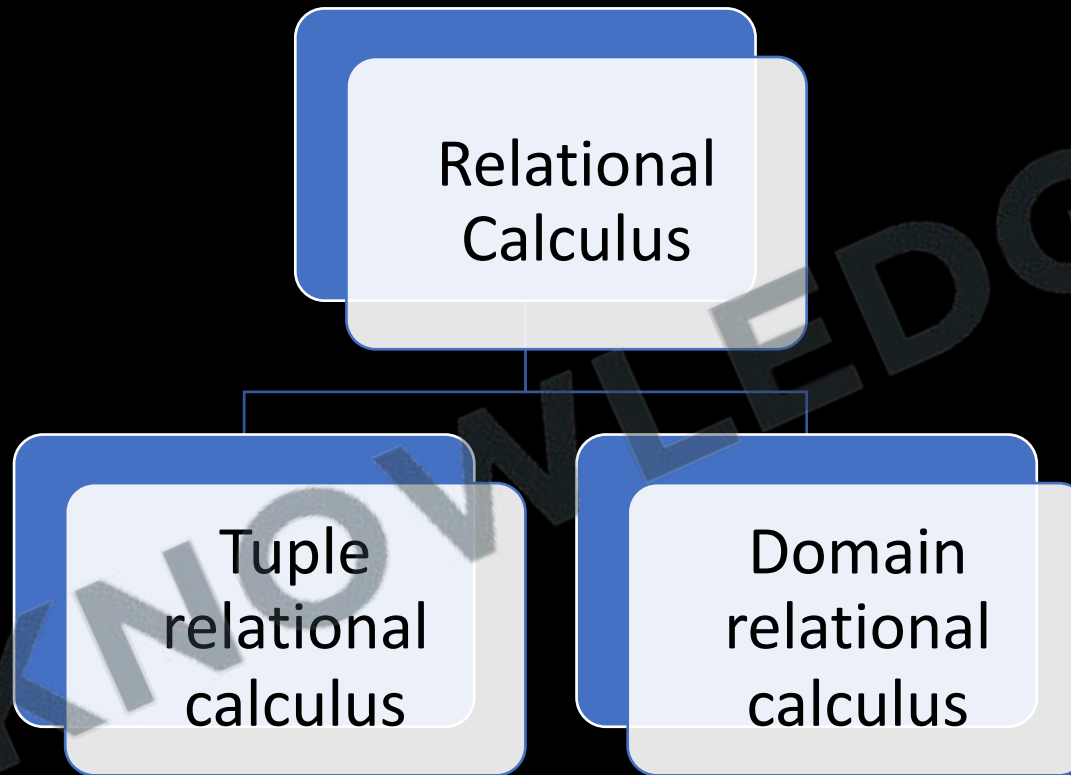
(B) $x = 5$ AND $x > 4$ and $x < 6$, where x is an integer

(C) $x < 5$ AND $\text{not}(x = 5)$

(D) None of the above

Relational Calculus

- Relational calculus is non-procedural query language, where we have to define what to get and not how to get it



<http://www.knowledgegate.in/gate>

Tuple Relational Calculus

- TRC is based on specifying a number of tuple variables. Each tuple variable usually range over a particular database relation, meaning that the variable may takes as its value from any individual tuple of a relation.
- A simple tuple relational calculus query is of the form.

$\{t \mid \text{Condition}(t)\}$

- Where t is a tuple variable and $\text{condition}(t)$ is a conditional expression involving. The result of such a query is the set of all tuple t that satisfy condition (t).

- We use $t[A]$ or $t.A$ to denote the value of tuple t on attribute A .
- we use $t \in r$ or $r(t)$ to denote that tuple t is in relation r .

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the details of all computer science students?

SQL: select * from student where branch = CSE

RA: $\{\sigma_{\text{branch} = \text{CSE}}(\text{Student})\}$

TRC:

DRC:

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the details of all computer science students?

SQL: select * from student where branch = CSE

RA: $\{\sigma_{\text{branch} = \text{CSE}}(\text{Student})\}$

TRC: $\{t \mid \text{Student}(t) \wedge t.\text{branch} = \text{CSE}\}$

DRC:

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the Roll No of all computer science students?

SQL: select Roll No from student where branch = CSE

RA: $\{\Pi_{\text{roll no}} (\sigma_{\text{branch} = \text{CSE}} (\text{Student}))\}$

TRC:

DRC:

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the Roll No of all computer science students?

SQL: select Roll No from student where branch = CSE

RA: $\{\Pi_{\text{roll no}} (\sigma_{\text{branch} = \text{CSE}} (\text{Student}))\}$

TRC: $\{t. \text{Roll No} \mid \text{Student}(t) \wedge t. \text{branch} = \text{CSE}\}$

DRC:

<http://www.knowledgegate.in/gate>

Formal Definition

- A tuple-relational-calculus expression is of the form: $\{t \mid P(t)\}$ where P is a formula.
- Several tuple variables may appear in a formula.
- A tuple variable is said to be a **free variable** unless it is quantified by a \exists or \forall .
- A tuple variable is said to be a bounded variable if it is quantified by \exists or \forall .

- $P(t)$ may have various conditions logically combined with OR (\vee), AND (\wedge), NOT (\neg).
- It also uses quantifiers:
 - $\exists t \in r (Q(t))$ = "there exists" a tuple in t in relation r such that predicate $Q(t)$ is true.
 - $\forall t \in r (Q(t))$ = $Q(t)$ is true "for all" tuples in relation r .

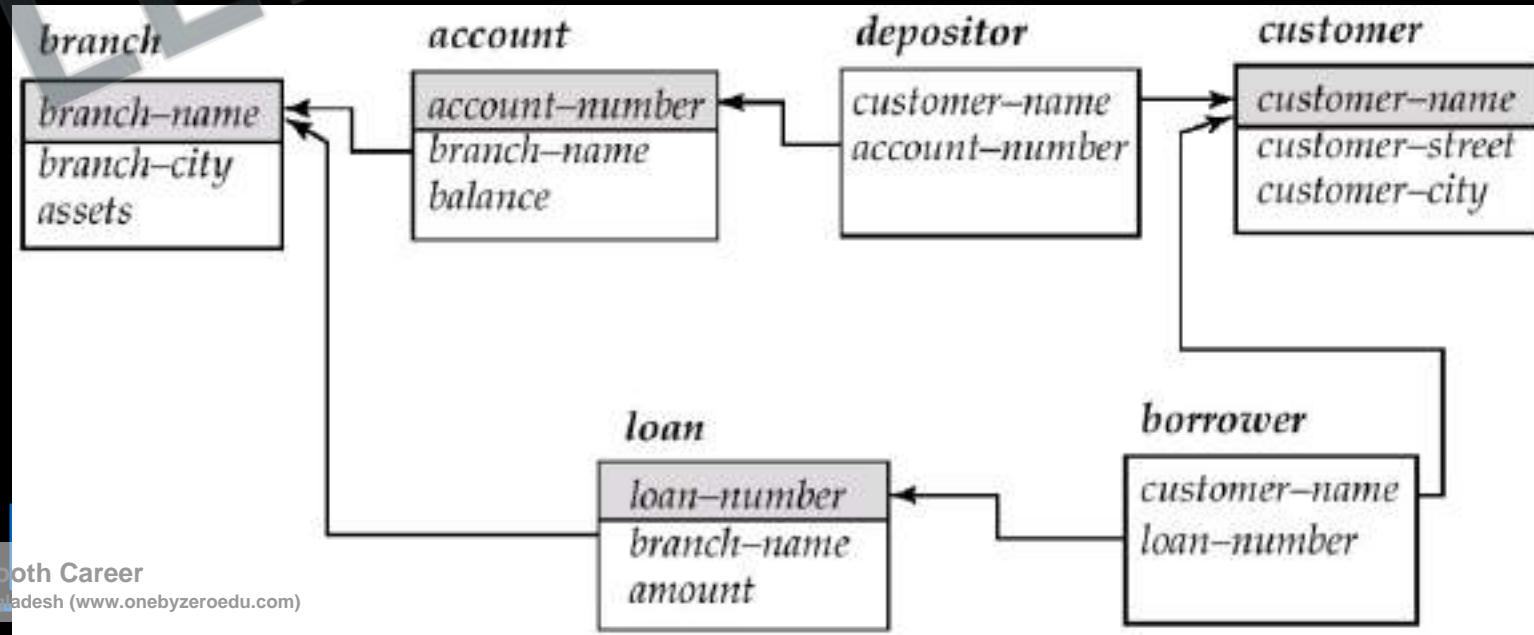
<http://www.knowledgegate.in/gate>

- A tuple-relational-calculus formula is built up out of atoms. An atom has one of the following forms:
 - $s \in r$, where s is a tuple variable and r is a relation.
 - $s[x] \text{ op } u[y]$, where s and u are tuple variables, x is an attribute on which s is defined, y is an attribute on which u is defined, and op is a comparison operator ($<, \leq, =, =, >, \geq$); we require that attributes x and y have domains whose members can be compared by .
 - $s[x] \text{ op } c$, where s is a tuple variable, x is an attribute on which s is defined, op is a comparison operator, and c is a constant in the domain of attribute x .

- We build up formulae from atoms by using the following rules:
 - An atom is a formula.
 - If P_1 is a formula, then so are $\neg P_1$ and (P_1) .
 - If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$.
 - If $P_1(s)$ is a formula containing a free tuple variable s , and r is a relation,
 - then $\exists s \in r (P_1(s))$ and $\forall s \in r (P_1(s))$

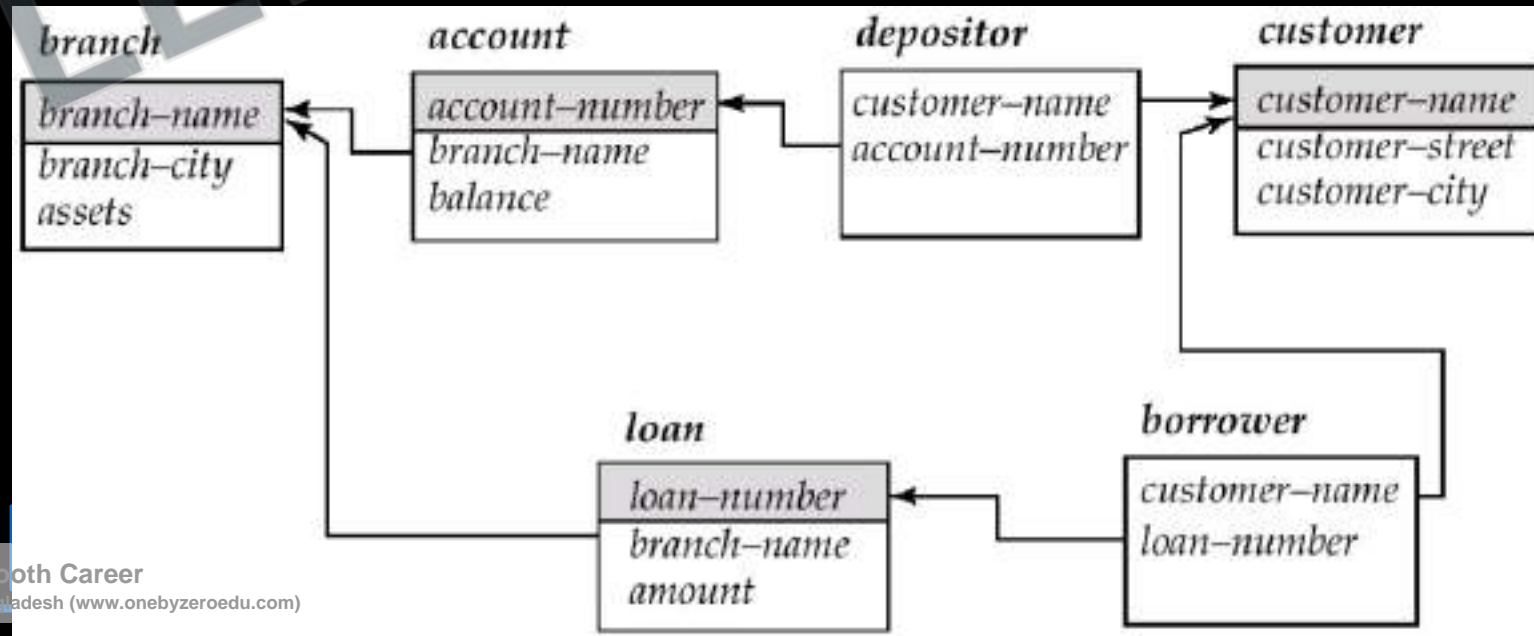
1. $P_1 \wedge P_2$ is equivalent to $\neg (\neg (P_1) \vee \neg (P_2))$.
2. $\forall t \in r (P_1(t))$ is equivalent to $\neg \exists t \in r (\neg P_1(t))$.
3. $P_1 \Rightarrow P_2$ is equivalent to $\neg(P_1) \vee P_2$.

$\{t \mid t \in \text{loan} \wedge t[\text{amount}] > 1200\}$

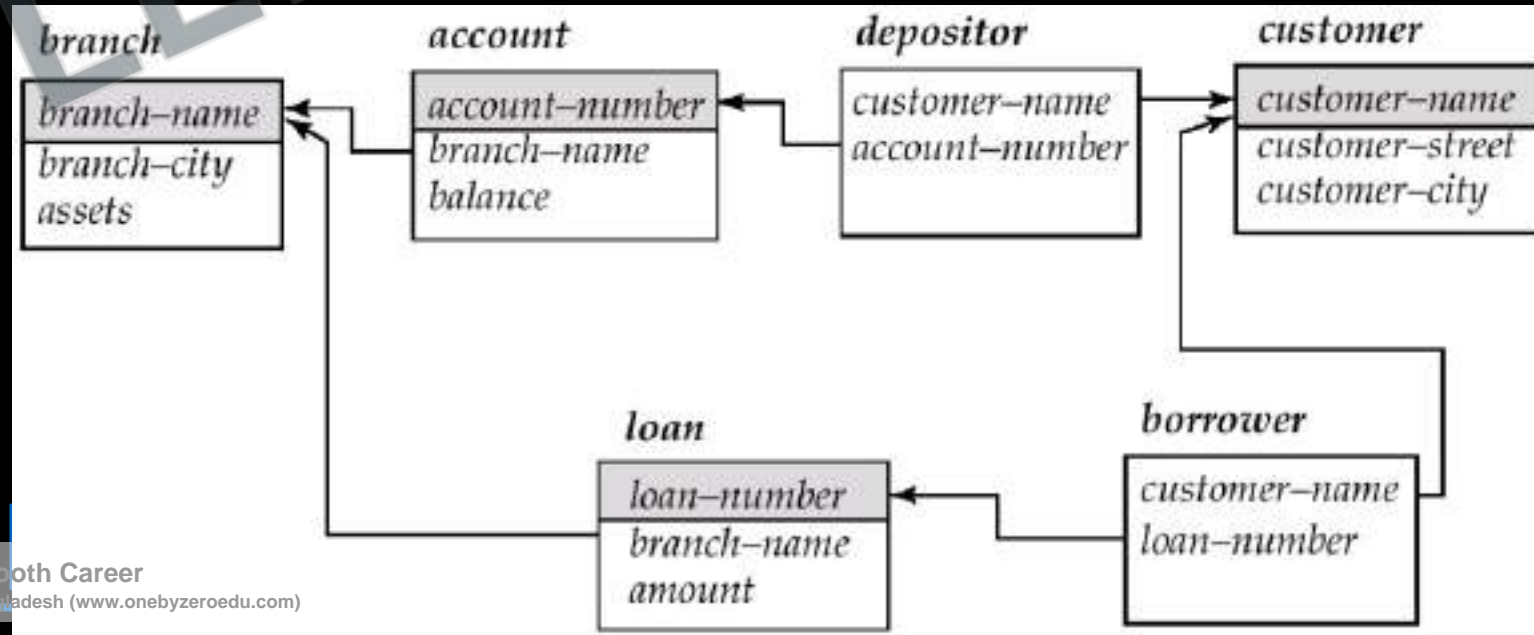


Q Find all the details of loan for amount over 1200?

$\{t \mid t \in \text{loan} \wedge t[\text{amount}] > 1200\}$

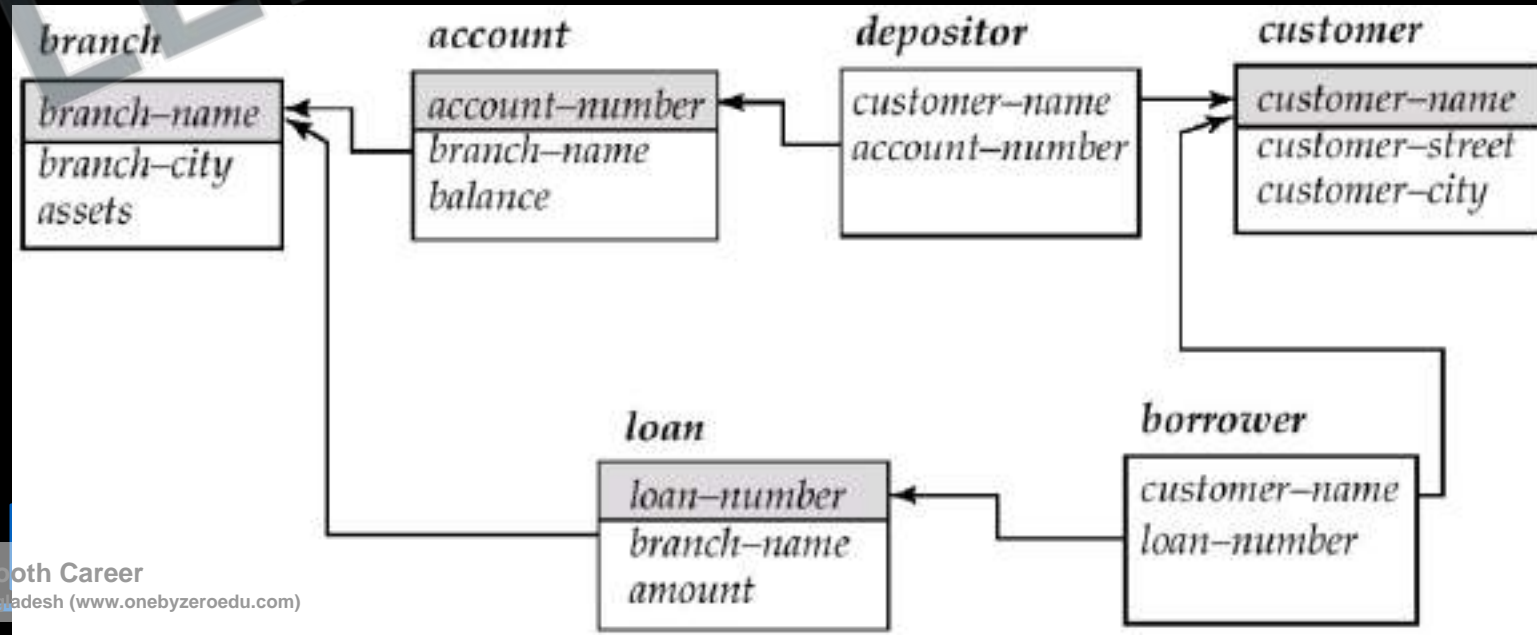


$\{t \mid \exists s \in \text{loan } (t[\text{loan number}] = s[\text{loan number}] \wedge s[\text{amount}] > 1200) \}$

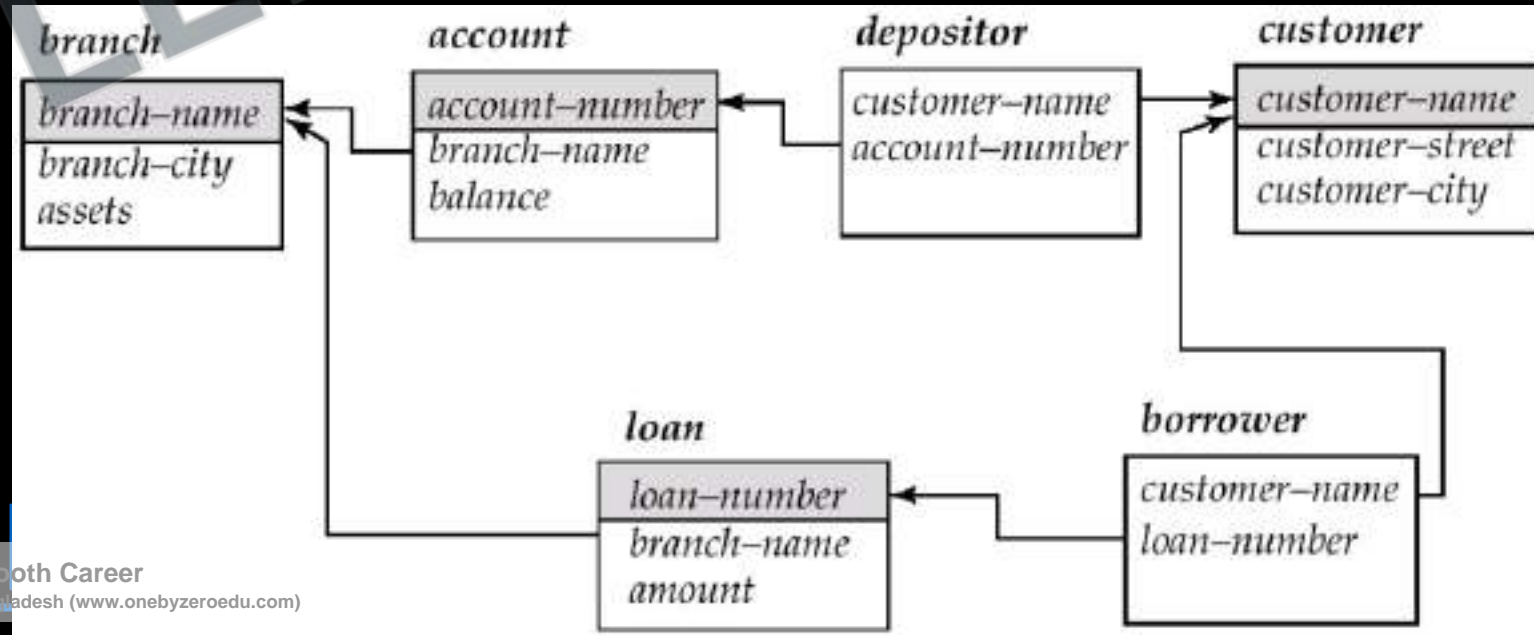


Q Find the loan number for each loan of amount over 1200?

$\{t \mid \exists s \in \text{loan} (t[\text{loan number}] = s[\text{loan number}]) \wedge s[\text{amount}] > 1200\}$



$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$
 $\wedge \exists u \in \text{loan } (u[\text{customer name}] = s[\text{customer name}])$
 $\wedge u[\text{branch name}] = \text{'Noida'} \}$

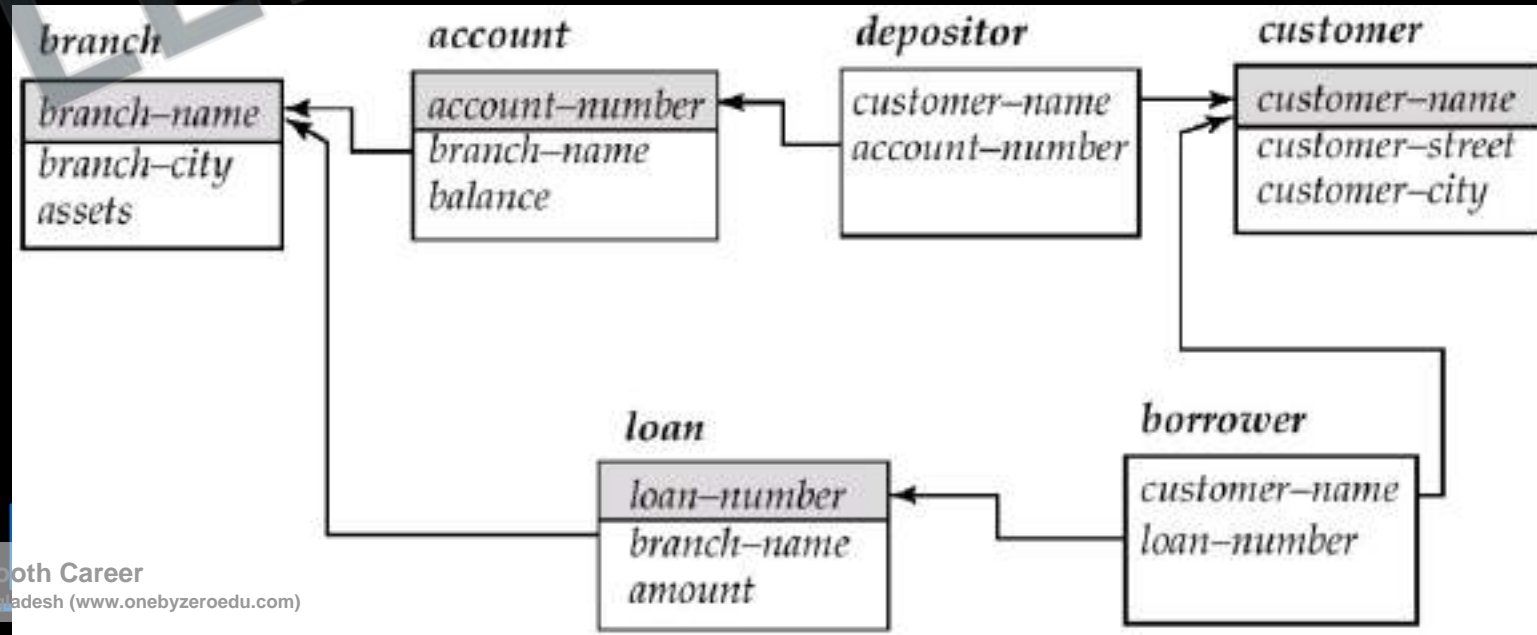


Q Find the name of all the customers who have a loan from Noida branch?

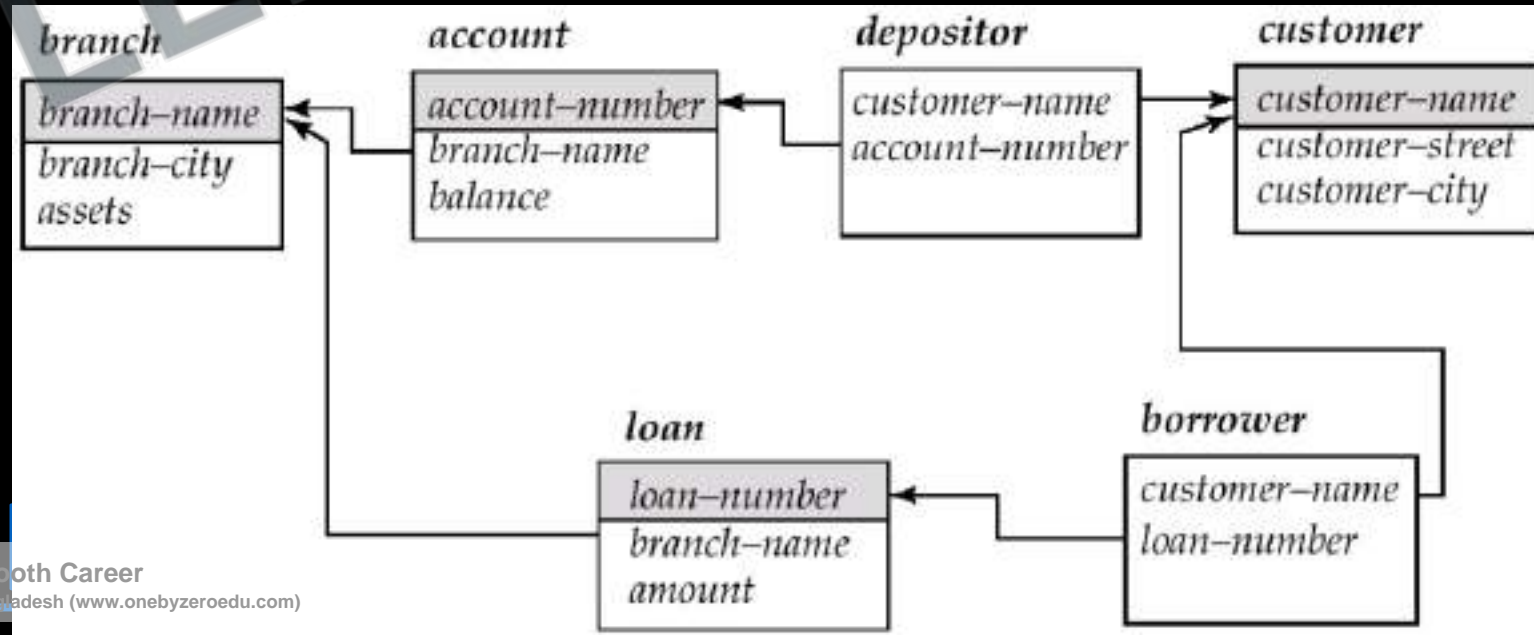
$\{t \mid \exists s \in \text{borrower} (t[\text{customer name}] = s[\text{customer name}])$

$\wedge \exists u \in \text{loan} (u[\text{customer name}] = s[\text{customer name}])$

$\wedge u[\text{branch name}] = \text{'Noida'} \}$



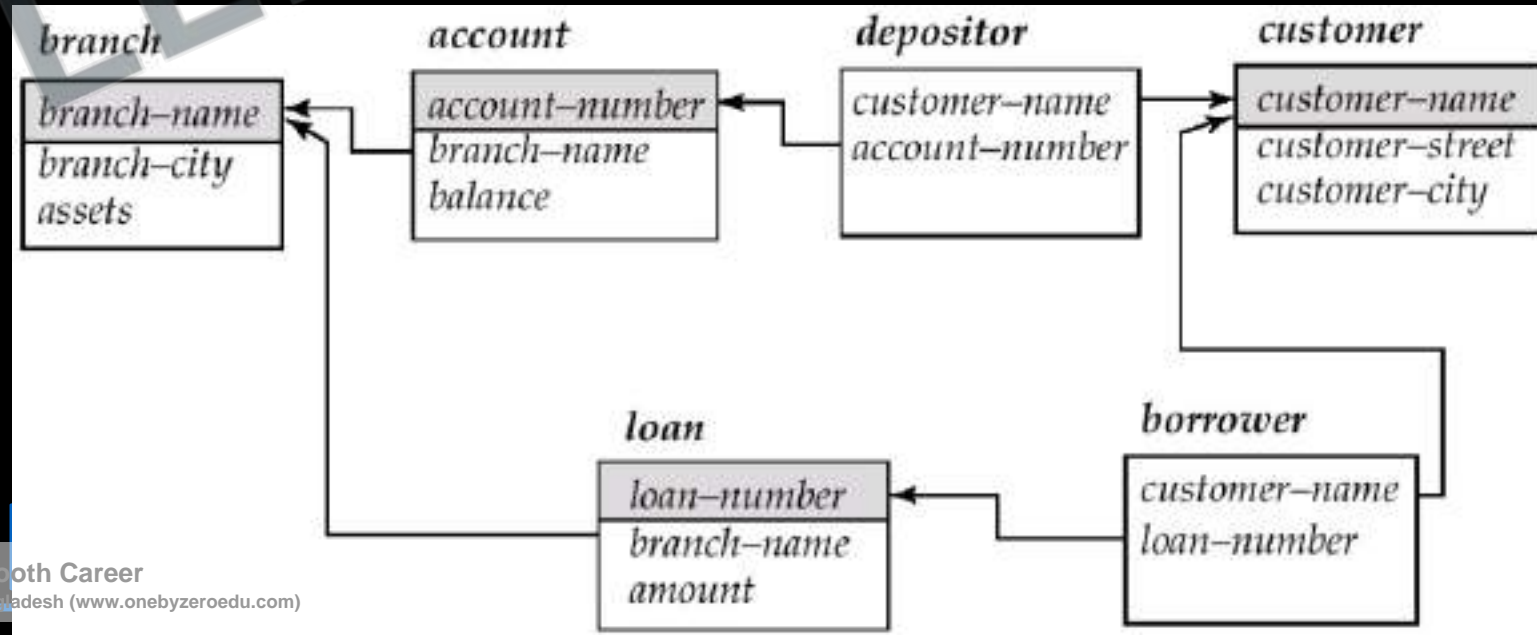
$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$
 $\vee \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}]) \}$



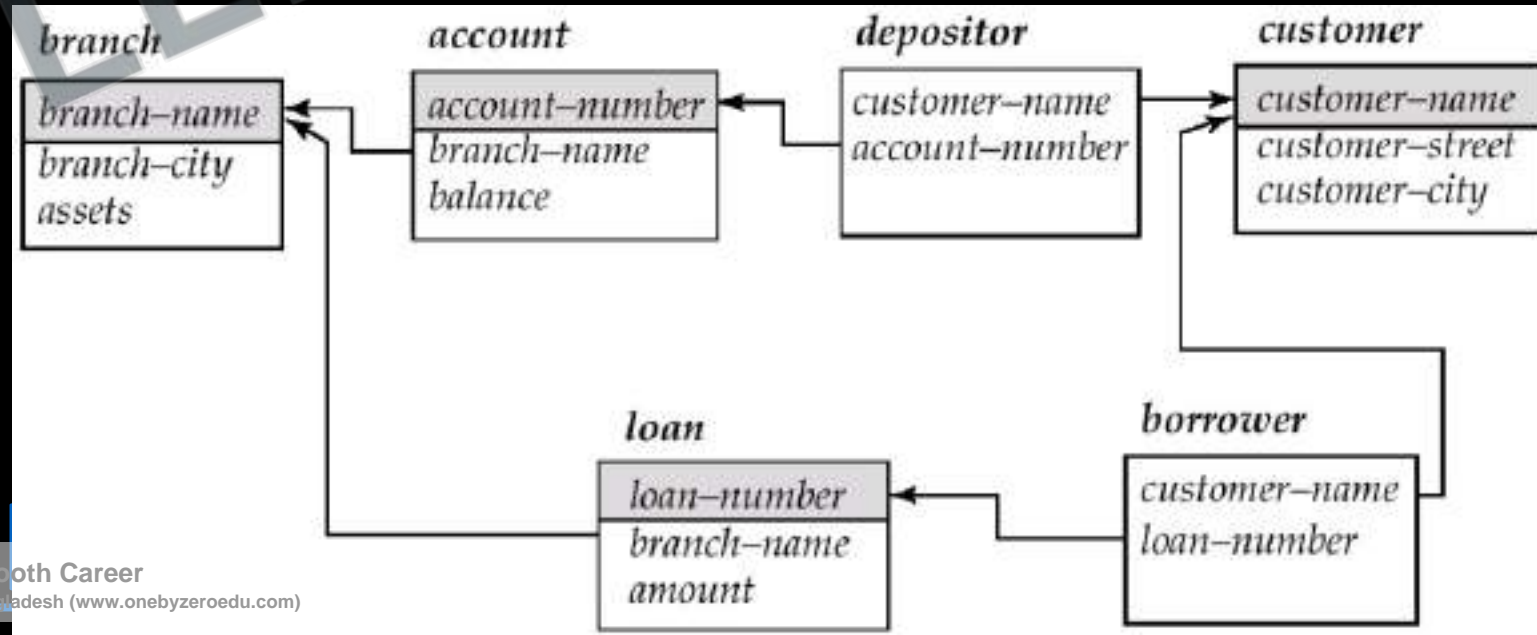
Q Find the name of all the customers who have a loan or account or both at the bank?

$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$

$\vee \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}]) \}$



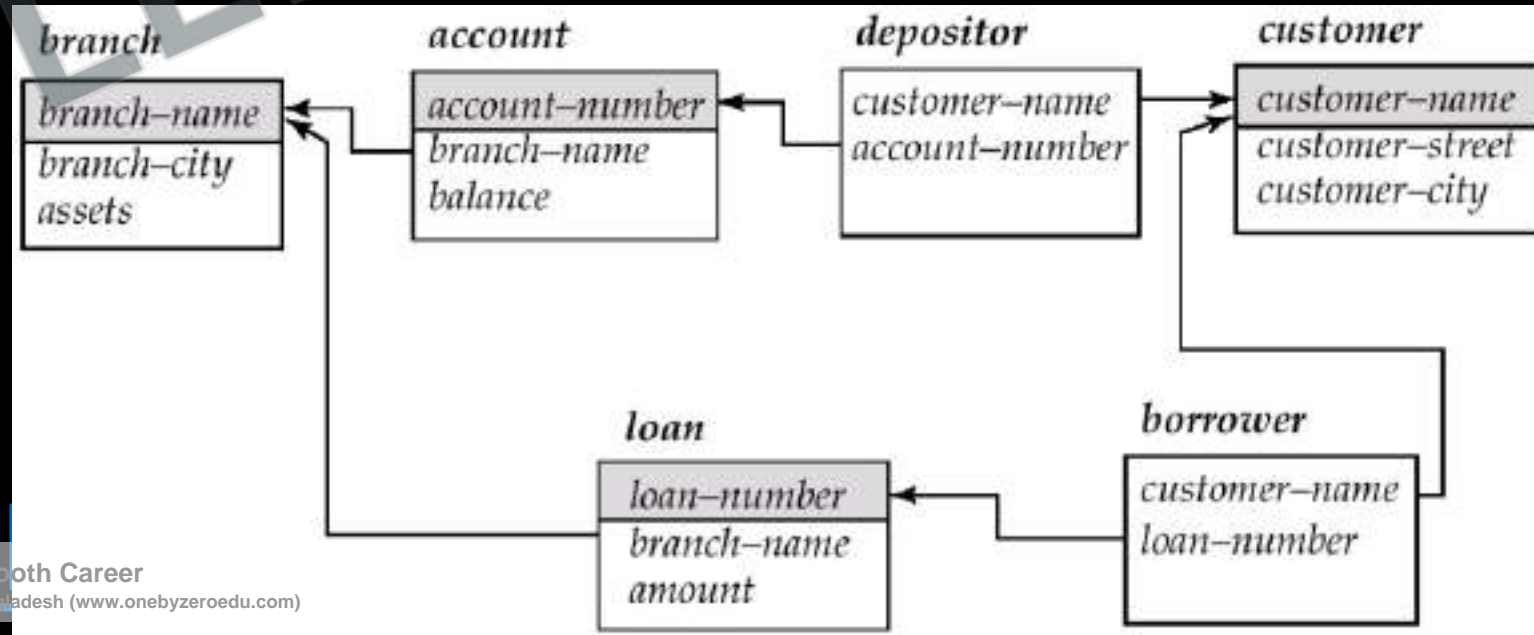
$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$
 $\wedge \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}]) \}$



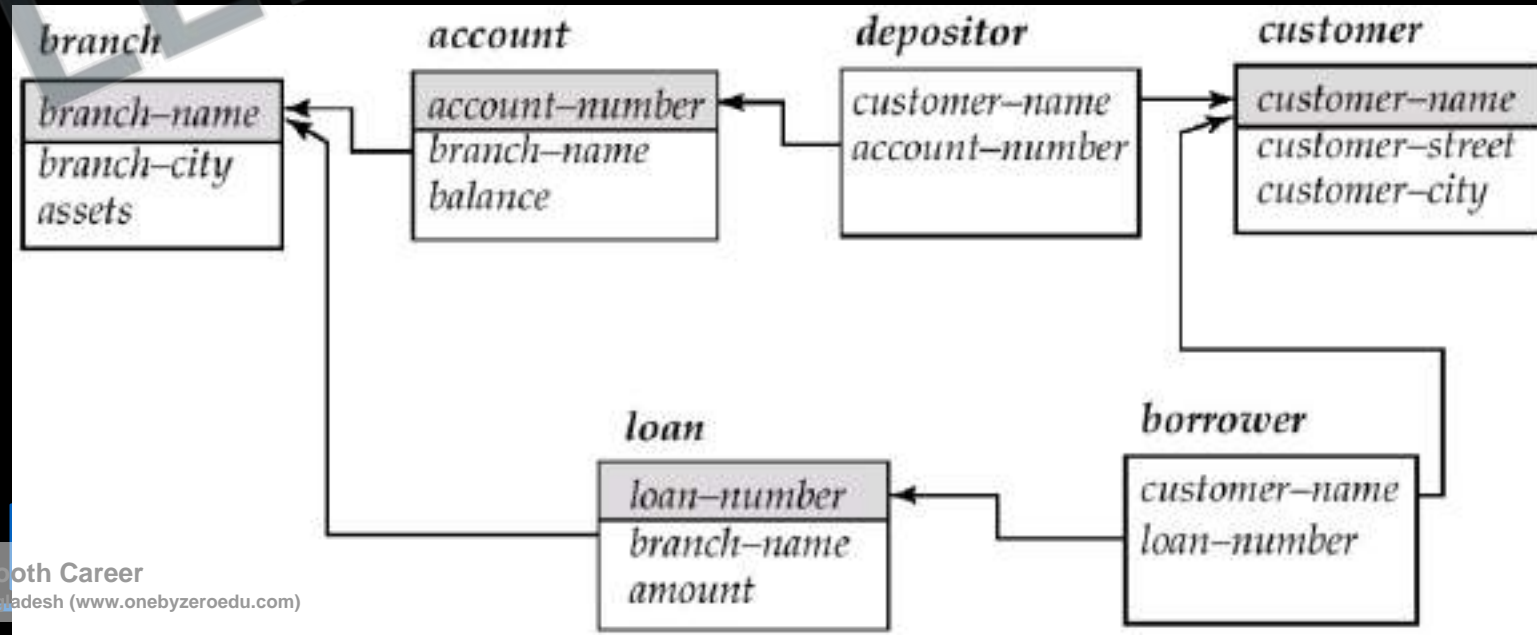
Q Find the name of all the customers who have a loan and account both at the bank?

$\{t \mid \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}])$

$\wedge \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}]) \}$

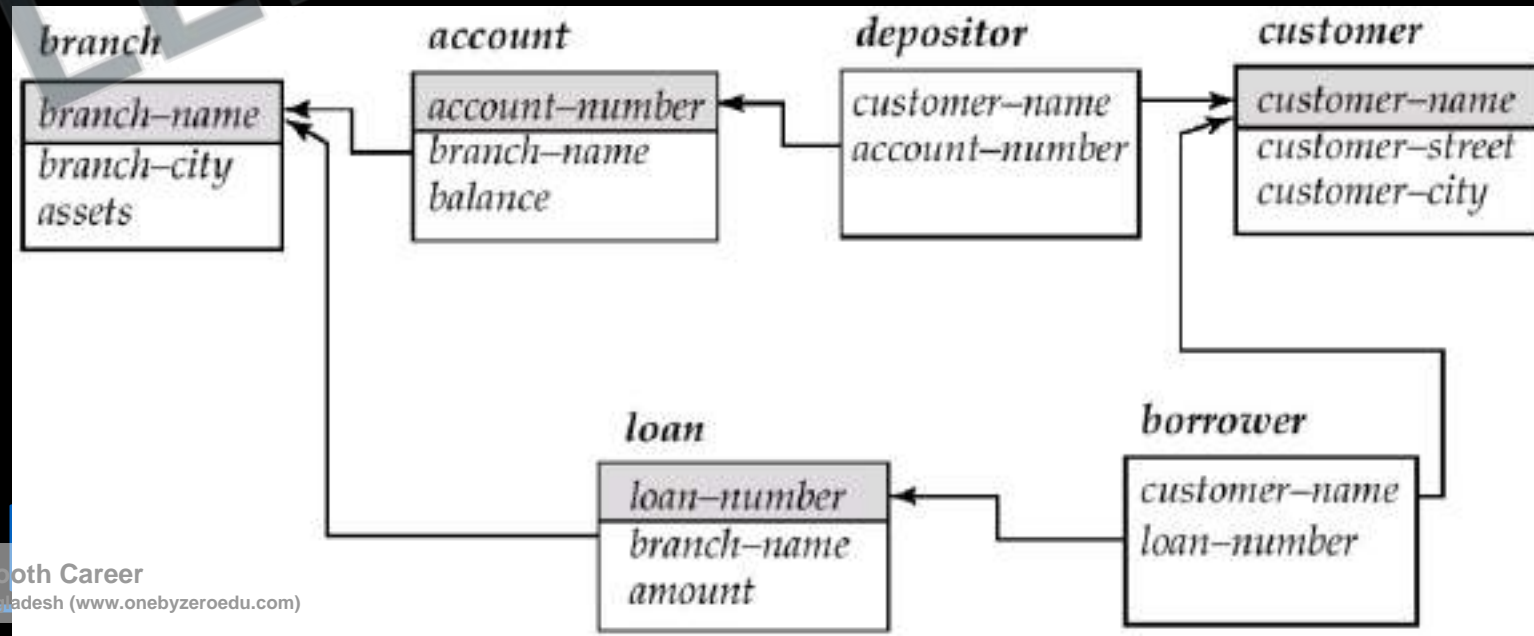


$\{t \mid \exists u \in \text{depositor borrower } (t[\text{customer name}] = u[\text{customer name}])$
 $\wedge \neg \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}]) \}$



Q Find the name of all the customers who have a loan from the bank and do not have a account?

$\{t \mid \exists u \in \text{depositor } (t[\text{customer name}] = u[\text{customer name}])$
 $\wedge \neg \exists s \in \text{borrower } (t[\text{customer name}] = s[\text{customer name}]) \}$



Expressive Power of Languages

Important Points to Remember

- The tuple relational calculus restricted to safe expressions is equivalent in expressive power to the basic relational algebra (with the operators \cup , $-$, \times , and \Join , but without the extended relational operations such as generalized projection and aggregation (G)).
- For every relational-algebra expression using only the basic operations, there is an equivalent expression in the tuple relational calculus, and for every tuple-relational-calculus expression, there is an equivalent relational algebra expression.
- Tuple relational calculus does not have any equivalent of the aggregate operation, but it can be extended to support aggregation. Extending the tuple relational calculus to handle arithmetic expressions is straightforward.

Safety of Expressions

- A tuple-relational-calculus expression may generate an infinite relation.

Example: $\{t \mid \neg (t \in \text{instructor})\}$

There are infinitely many tuples that are not in instructor. Most of these tuples contain values that do not even appear in the database. We do not want to allow such expressions.

- To help us define a restriction of the tuple relational calculus the concept of the **domain** of a tuple relational formula, P is introduced.
- The domain of P , denoted $\text{dom}(P)$, is the set of all values referenced by P .
- They include values mentioned in P itself, as well as values that appear in a tuple of a relation mentioned in P .

Example: $\text{dom}(t \in \text{instructor} \wedge t[\text{salary}] > 80000)$ is the set containing 80000 as well as the set of all values appearing in any attribute of any tuple in the instructor relation.

- An expression $\{t \mid P(t)\}$ is safe if all values that appear in the result are values from $\text{dom}(P)$.
- The expression $\{t \mid \neg (t \in \text{instructor})\}$ is not safe.
- Safe expressions are guaranteed to have finite results.

Q Which of the rational calculus expression is not safe? (Gate-2001) (2 Marks)

- a) $\{t \mid \exists u \in R_1(t[A] = u[A]) \wedge \neg \exists s \in R_2(t[A] = s[A])\}$
- b) $\{t \mid \forall u \in R_1(u[A] = "x" \Rightarrow \exists s \in R_2(t[A] = s[A] \wedge s[A] = u[A]))\}$
- c) $\{t \mid \neg(t \in R_1)\}$
- d) $\{t \mid \exists u \in R_1(t[A] = u[A]) \wedge \exists s \in R_2(t[A] = s[A])\}$

Domain Relational Calculus

- Domain calculus differs from the tuple calculus in the type of variables used in formulas: rather than having variables range over tuples.
- The variable range over single values from domains of attributes. To form a relation of degree n for a query result, we must have n of these domain variables. One for each attribute.
- An expression of the domain calculus is of the form
- $(x_1, x_2, \dots, x_n \mid \text{COND}(x_1, x_2, \dots, x_n, x_{n+1}, x_{n+2}, \dots, x_{n+m}))$
- Where x_1, x_2, \dots, x_n are domain variables that range over domains and COND is a condition of the domain relational calculus.

Student(Roll No, Name, Branch)

Q Find the details of all computer science students?

SQL: select * from student where branch = CSE

RA: $\{\sigma_{\text{branch} = \text{CSE}}(\text{Student})\}$

TRC: $\{t \mid \text{Student}(t) \wedge t.\text{branch} = \text{CSE}\}$

DRC:

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the details of all computer science students?

SQL: select * from student where branch = CSE

RA: $\{\sigma_{\text{branch} = \text{CSE}}(\text{Student})\}$

TRC: $\{t \mid \text{Student}(t) \cap t.\text{branch} = \text{CSE}\}$

DRC: $\{(\text{Roll No, Name, Branch}) \mid \text{Student}(\text{Roll no, Name, Branch}) \cap \text{branch} = \text{CSE}\}$

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the Roll No of all computer science students?

SQL: select Roll No from student where branch = CSE

RA: $\{\Pi_{\text{Roll No}} (\sigma_{\text{branch} = \text{CSE}} (\text{Student}))\}$

TRC: $\{t. \text{Roll No} \mid \text{Student}(t) \wedge t. \text{branch} = \text{CSE}\}$

DRC:

<http://www.knowledgegate.in/gate>

Student(Roll No, Name, Branch)

Q Find the Roll No of all computer science students?

SQL: select Roll No from student where branch = CSE

RA: $\{\Pi_{\text{roll no}} (\sigma_{\text{branch} = \text{CSE}} (\text{Student}))\}$

TRC: $\{t. \text{ Roll No} \mid \text{Student}(t) \cap t. \text{ branch} = \text{CSE}\}$

DRC: $\{(\text{Roll No}) \mid \text{Student}(\text{Roll no, Name, Branch}) (\text{Name, Branch}) \cap \text{branch} = \text{CSE}\}$

Instructor(instructorID, name, dept name, and salary)

$\{ \langle i, n, d, s \rangle \mid \langle i, n, d, s \rangle \in \text{instructor} \wedge s > 80000 \}$

<http://www.knowledgegate.in/gate>

Example: Find all details of instructors whose salary is greater than \$80,000

$\{ \langle i, n, d, s \rangle \mid \langle i, n, d, s \rangle \in \text{instructor} \wedge s > 80000 \}$

<http://www.knowledgegate.in/gate>

The Domain Relational Calculus

- Uses domain variables that take on values from an attributes domain, rather than values for an entire tuple.
- Closely related to the tuple relational calculus.
- Domain relational calculus serves as the theoretical basis of the widely used QBE language

<http://www.knowledgegate.in/gate>

Formal Definition

- An expression in the domain relational calculus is of the form $\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$ where x_1, x_2, \dots, x_n represent domain variables. P represents a formula composed of atoms.

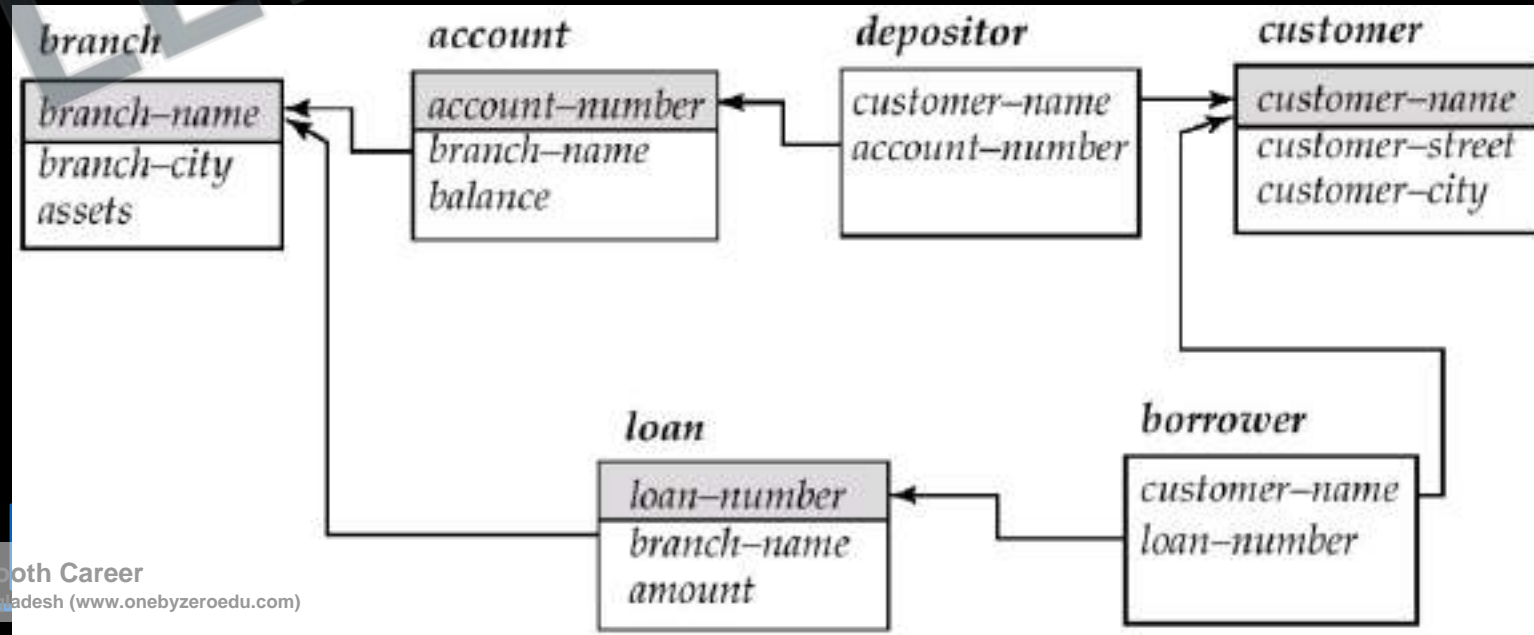
- An atom in the domain relational calculus has one of the following forms:
 - $\langle x_1, x_2, \dots, x_n \rangle \in r$, where r is a relation on n attributes and x_1, x_2, \dots, x_n are domain variables or domain constants.
 - $x \text{ op } y$, where x and y are domain variables and op is a comparison operator ($<$, $>$, \geq). We require that attributes x and y have domains that can be compared by .
 - $x \text{ op } c$, where x is a domain variable, op is a comparison operator, and c is a constant in the domain of the attribute for which x is a domain variable.

We build up formulae from atoms by using the following rules:

- An atom is a formula.
 - If P_1 is a formula, then so are $\neg P_1$ and (P_1) .
 - If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$.
 - If $P_1(x)$ is a formula in x , where x is a free domain variable, then $\exists x (P_1(x))$ and $\forall x (P_1(x))$ are also formulae.

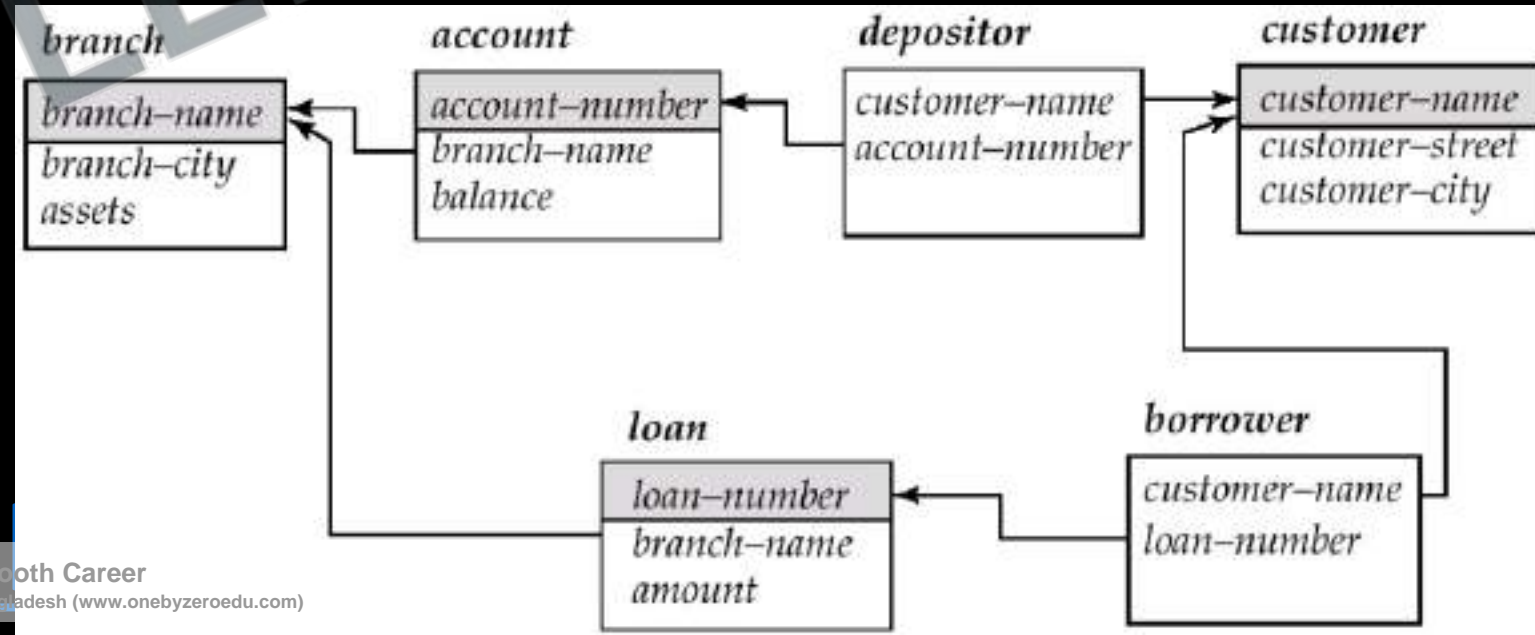
As a notational shorthand, we write $\exists a, b, c (P(a, b, c))$ for $\exists a (\exists b (\exists c (P(a, b, c))))$.

$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$

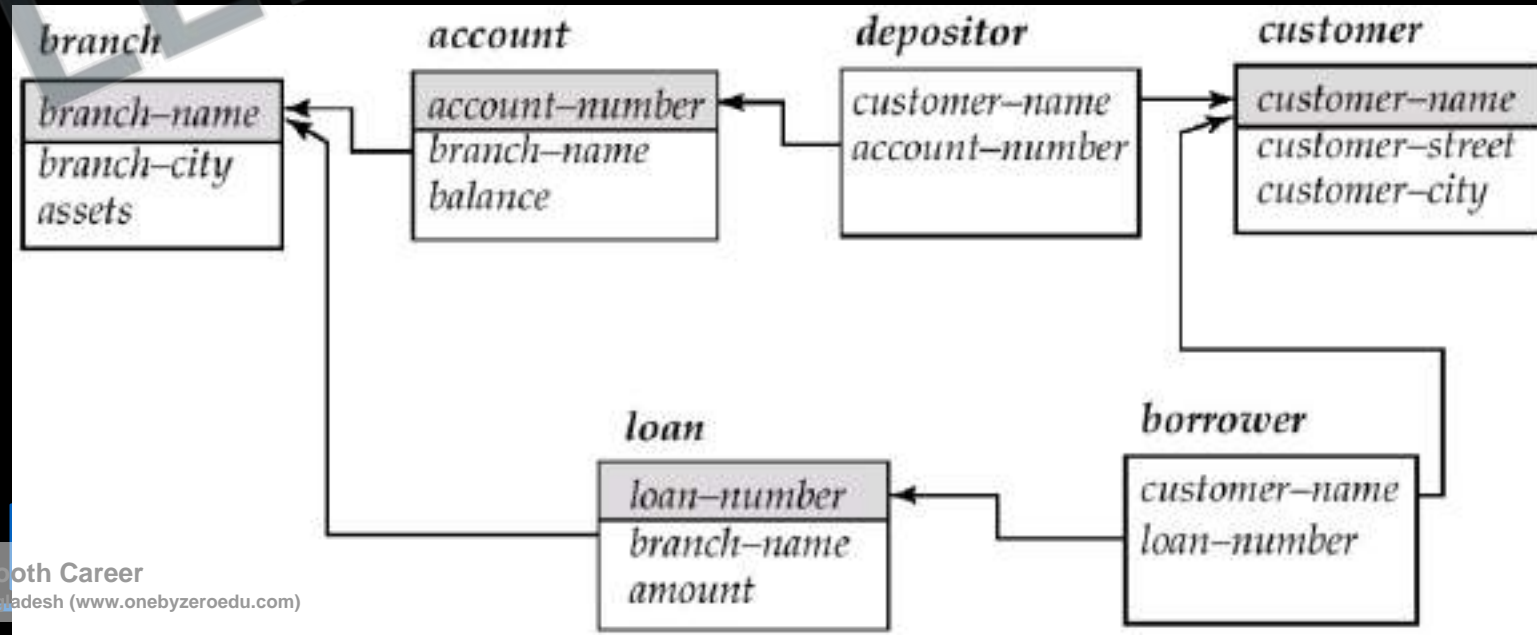


Q Find the branch name, loan number and amount for loan of amount over 1200?

$\{ \langle l, b, a \rangle \mid \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$

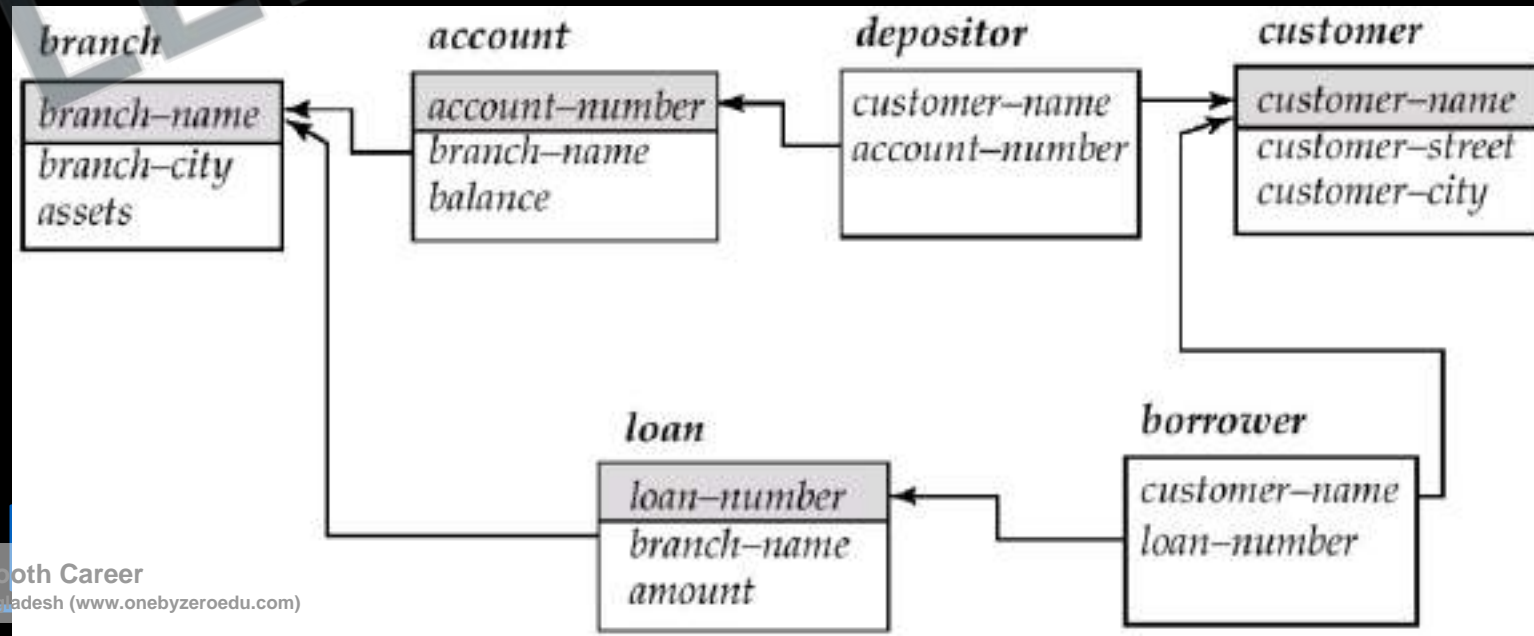


$\{ \langle l \rangle \mid \exists_{b,a} \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$

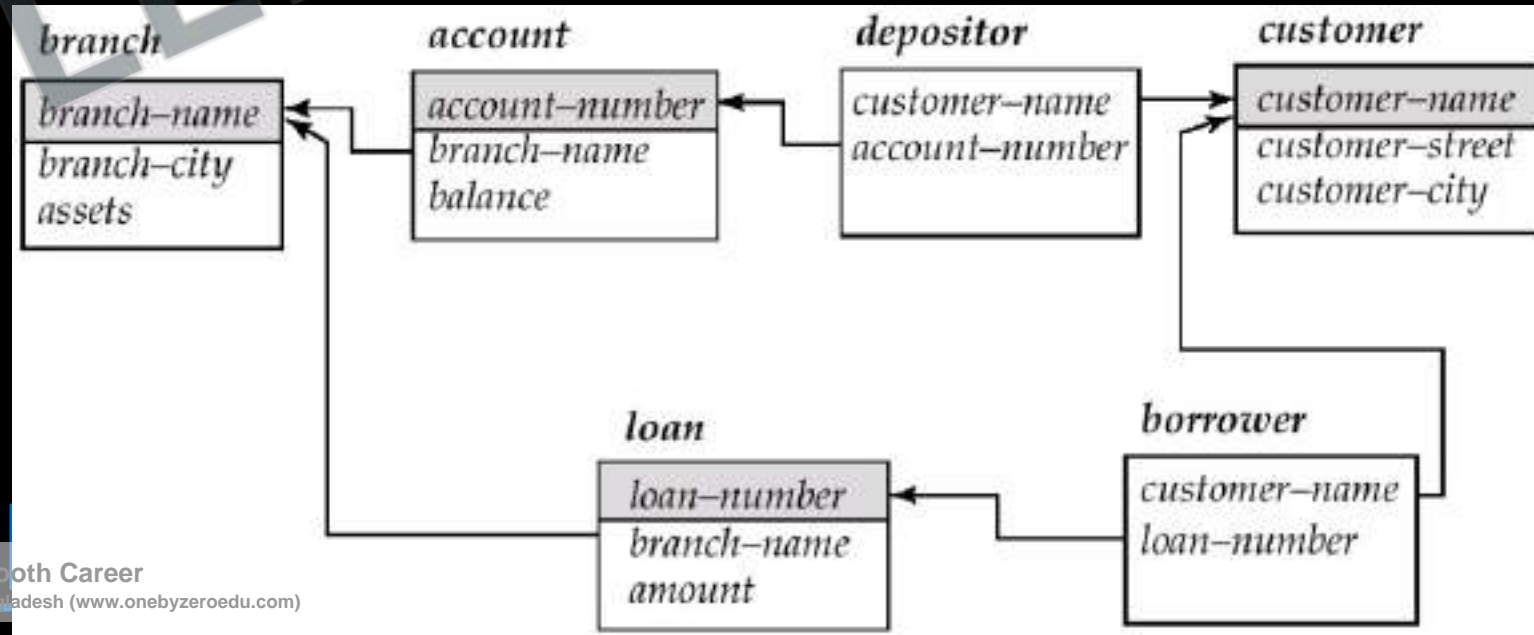


Q Find the loan number for each loan of amount over 1200?

$\{ \langle l \rangle \mid \exists_{b,a} \langle l, b, a \rangle \in \text{loan} \wedge a > 1200 \}$

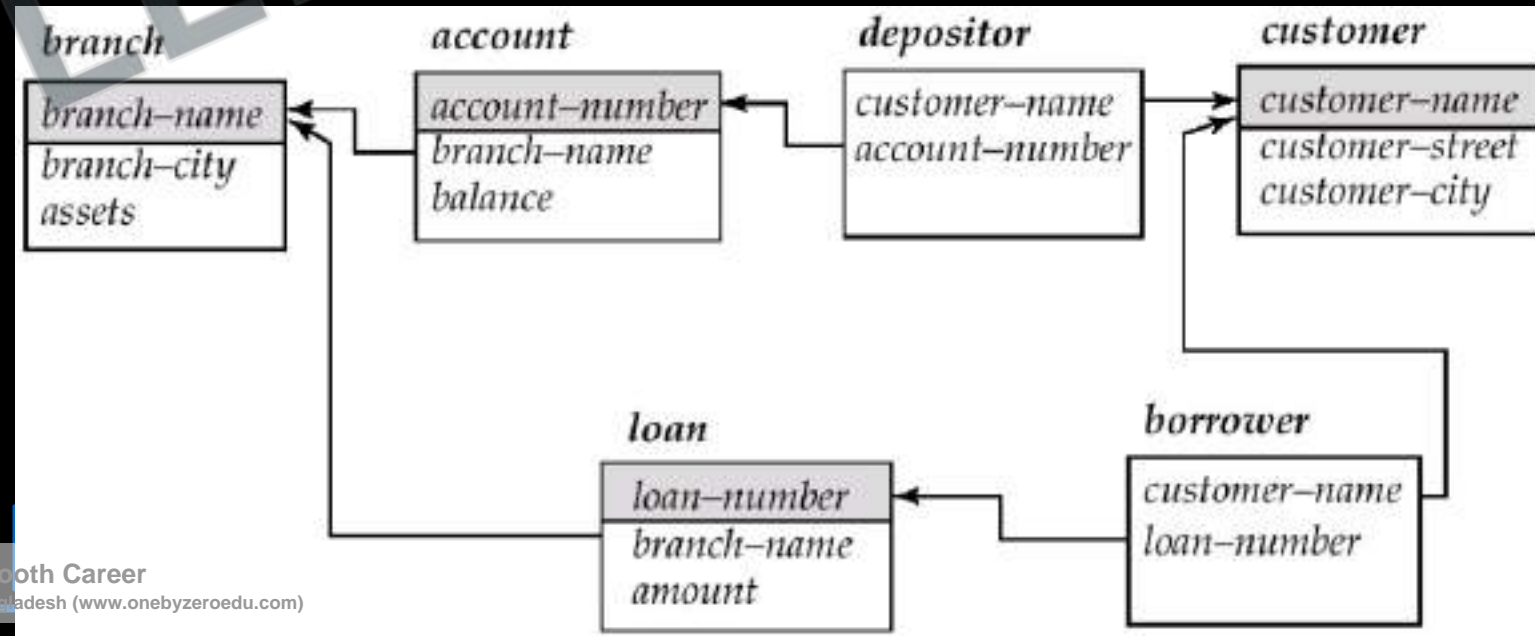


$\{ \langle c, a \rangle \mid \exists_l (\langle c, l \rangle \in \text{borrower} \wedge \exists_b (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{'Noida'})) \}$



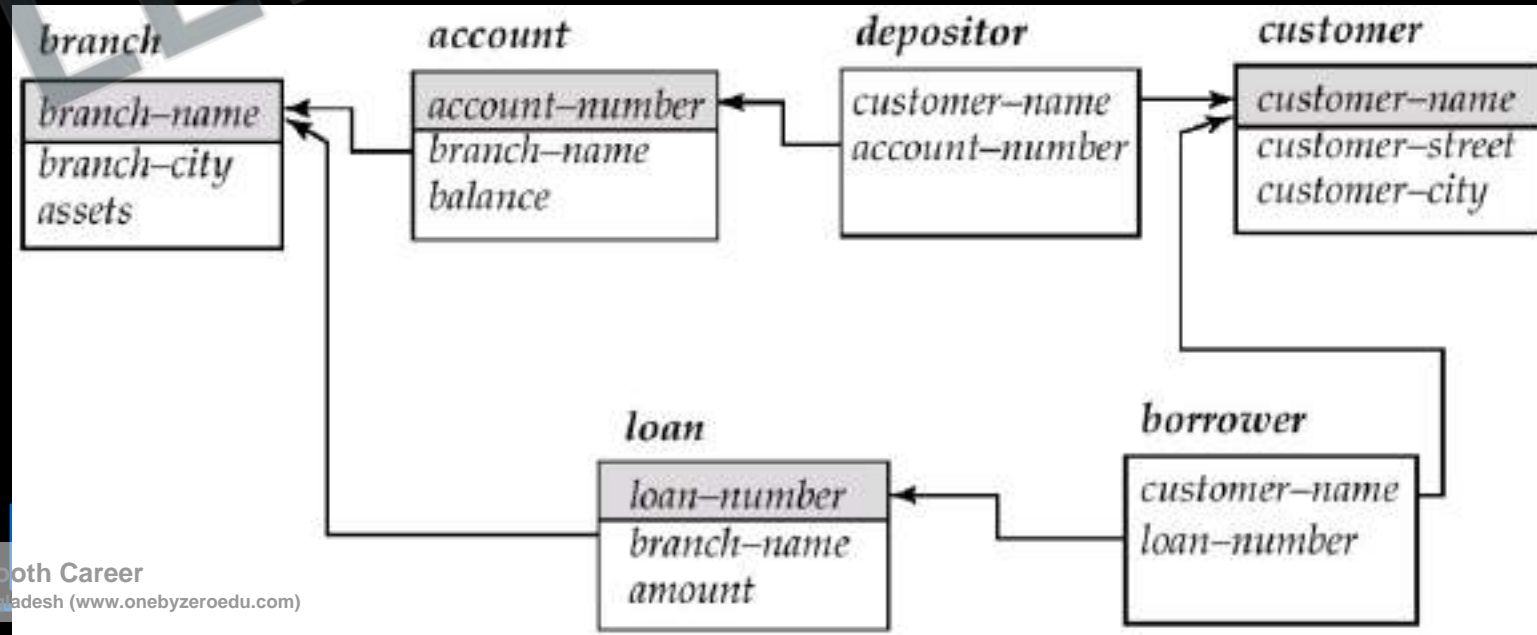
Q Find the name of all the customers who have a loan from Noida branch with loan amount?

$\{ \langle c, a \rangle \mid \exists_l (\langle c, l \rangle \in \text{borrower} \wedge \exists_b (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{'Noida'})) \}$



$\{ \langle c \rangle \mid \exists_l (\langle c, l \rangle \in \text{borrower} \wedge \exists_{b,a} (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{'Noida'})) \}$

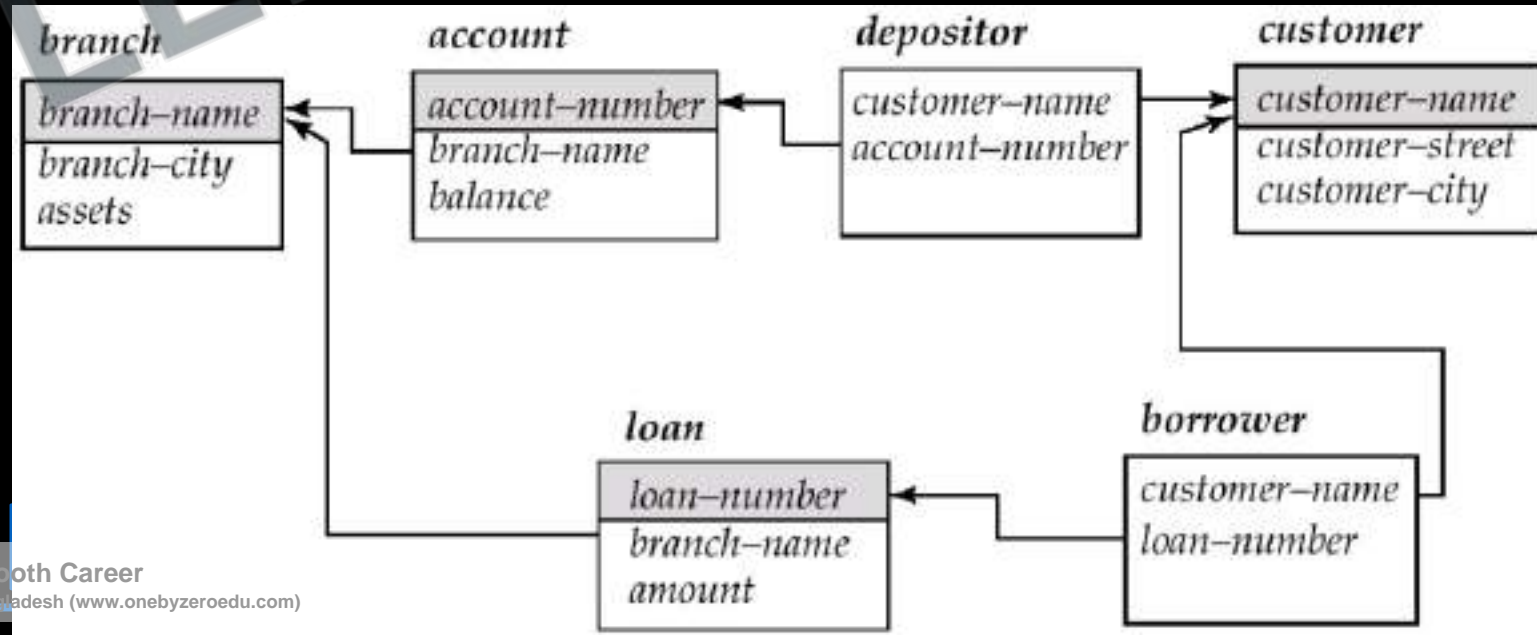
$\forall \exists_a (\langle c, a \rangle \in \text{depositor} \wedge \exists_{bn,bal} (\langle a, bn, bal \rangle \in \text{account} \wedge bn = \text{'Noida'})) \}$



Q Find the name of all the customers who have a loan or account or both at the bank?

$\{ \langle c \rangle \mid \exists_l (\langle c, l \rangle \in \text{borrower} \wedge \exists_{b,a} (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{'Noida'}))$

$\vee \exists_a (\langle c, a \rangle \in \text{depositor} \wedge \exists_{b,a} (\langle a, bn, bal \rangle \in \text{account} \wedge bn = \text{'Noida'})) \}$



Safety of Expressions

- Similar to TRC we can have an unsafe expression that may generate infinite relations.
- An expression such as

$$\{ \langle i, n, d, s \rangle \mid \neg (\langle i, n, d, s \rangle \in \text{instructor}) \}$$

is unsafe, because it allows values in the result that are not in the domain of the expression.

- We say that an expression $\{ \langle x_1, x_2, \dots, x_n \rangle \mid P(x_1, x_2, \dots, x_n) \}$ is safe if all of the following hold:
 1. All values that appear in tuples of the expression are values from $\text{dom}(P)$.
 2. For every “there exists” sub formula of the form $\exists x (P_1(x))$, the sub formula is true if and only if there is a value x in $\text{dom}(P_1)$ such that $P_1(x)$ is true.
 3. For every “for all” sub formula of the form $\forall x (P_1(x))$, the sub formula is true if and only if $P_1(x)$ is true for all values x from $\text{dom}(P_1)$.

Expressive Power of Languages

- When the domain relational calculus is restricted to safe expressions, it is equivalent in expressive power to the tuple relational calculus restricted to safe expressions.
- All three of the following are equivalent:
 - The basic relational algebra (without the extended relational-algebra operations)
 - The tuple relational calculus restricted to safe expressions
 - The domain relational calculus restricted to safe expressions
- Domain relational calculus also does not have any equivalent of the aggregate operation, but it can be extended to support aggregation, and extending it to handle arithmetic expressions is straightforward.

Important difference between Tuple Relational Calculus and Domain Relational Calculus.

- In the tuple calculus, when we write $\exists s$ for some tuple variable s , we bind it immediately to a relation by writing $\exists s \in r$.
- However, when we write $\exists n$ in the domain calculus, n refers not to a tuple, but rather to a domain value. Thus, the domain of variable n is unconstrained until the sub formula $\langle i, n, d, s \rangle \in \text{instructor}$ constrains n to instructor names that appear in the instructor relation.

Q Consider the following relational schema. (Gate-2013) (2 Marks)

Students (rollno: integer, sname: string)

Courses (courseno: integer, cname: string)

Registration (rollno: integer, courseno: integer, percent: real)

Which of the following queries are equivalent to this query in English?

“Find the distinct names of all students who score more than 90% in the course numbered 107”

i) SELECT DISTINCT S. sname FROM Students as S, Registration as R WHERE

R. rollno = S. rollno AND R. courseno = 107 AND R. percent > 90

ii) $\Pi_{\text{sname}} (\sigma_{\text{courseno} = 107 \wedge \text{percent} > 90} (\text{Registration} \bowtie \text{Students}))$

iii) $\{T | \exists S \in \text{Students}, \exists R \in \text{Registration} (S.\text{rollno} = R.\text{rollno} \wedge R.\text{courseno} = 107 \wedge R.\text{percent} > 90 \wedge T.\text{sname} = S.\text{sname})\}$

iv) $\{\langle SN \rangle | \exists SR \exists RP (\langle SR, SN \rangle \in \text{Students} \wedge \langle SR, 107, RP \rangle \in \text{Registration} \wedge RP > 90)\}$

a) I, II, III and IV

b) I, II and III only

c) I, II and IV only

d) II, III and IV only

<http://www.knowledgegate.in/gate>

GATE (2008)

Which of the following tuple relational calculus expression(s) is/are equivalent to $\forall t \in r(P(t))$?

- I. $\neg \exists t \in r(P(t))$
- II. $\exists t \notin r(P(t))$
- III. $\neg \exists t \in r(\neg P(t))$
- IV. $\exists t \notin r(\neg P(t))$

- (A) I only
- (B) II only
- (C) III only
- (D) III and IV only

<http://www.knowledgegate.in/gate>

Q Consider the relation employee (name, sex, supervisor Name) with name as the key. supervisor Name gives the name of the supervisor of the employee under consideration. What does the following Tuple Relational Calculus query produce? **(Gate-2007) (2 Marks)**

$$\{e.name \mid \text{employee}(e) \wedge (\forall x) [\neg \text{employee}(x) \vee x.\text{supervisorName} \neq e.name \vee x.\text{sex} = \text{"male"}] \}$$

- (A)** Names of employees with a male supervisor.
- (B)** Names of employees with no immediate male subordinates.
- (C)** Names of employees with no immediate female subordinates.
- (D)** Names of employees with a female supervisor.

<http://www.knowledgegate.in/gate>

Q With regard to the expressive power of the formal relational query languages, which of the following statements is true? **(CS-2002)**

- (A)** Relational algebra is more powerful than relational calculus
- (B)** Relational algebra has the same power as relational calculus
- (C)** Relational algebra has the same power as safe relational calculus
- (D)** None of the above

<http://www.knowledgegate.in/gate>