DBMS-IAT-2 Solution-4th semester-BCS403

1.Explain Unary Relational operations with examples?
Ans:

The SELECT Operation

- The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a¬ selection condition.
- It restricts the tuples in a relation to only those tuples that¬ satisfy the condition.
- It can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are discarded.
- For example, to select the EMPLOYEE tuples whose department is 4, or those whose salary is greater than $30,000

σDno=4(EMPLOYEE)
σSalary>30000(EMPLOYEE)

- In general, the SELECT operation is denoted by

  σ<selection condition>(R)

  where the symbol σ (sigma) is used to denote the SELECT operator and the selection condition is a Boolean expression (condition) specified on the attributes of relation R.
- The Boolean expression specified in is made up of a number of clauses of the form :

  <attribute name><comparison op><constant value>

  Or

  <attribute name><comparison op><attribute name>

The PROJECT Operation

- The PROJECT operation, selects certain columns from the table and discards the other columns.
- The result of the PROJECT operation can be visualized as a vertical partition of the relation into two relations: one has the needed columns (attributes) and contains the result of the operation, and the other contains the discarded columns.
- For example, to list each employee's first and last name and salary, we can use the PROJECT operation as follows:

πLname, Fname,

π<attribute list>(R)

- The general form of the PROJECT operation is
  where (pi) is the symbol used to represent the PROJECT operation, and is the desired sublist of attributes
  from the attributes of relation R.
- The result of the PROJECT operation has only the attributes specified in in the same order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>.
- The PROJECT operation removes any duplicate tuples, so the result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as duplicate elimination.

RENAME Operation

- The relations shown above depict operation results do not have any names.
- Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations.
- In the latter case, we must give names to the relations that hold the intermediate results.
- For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, apply a SELECT and a PROJECT operation.

πFname, Lname, Salary(σDno=5(EMPLOYEE))

✓

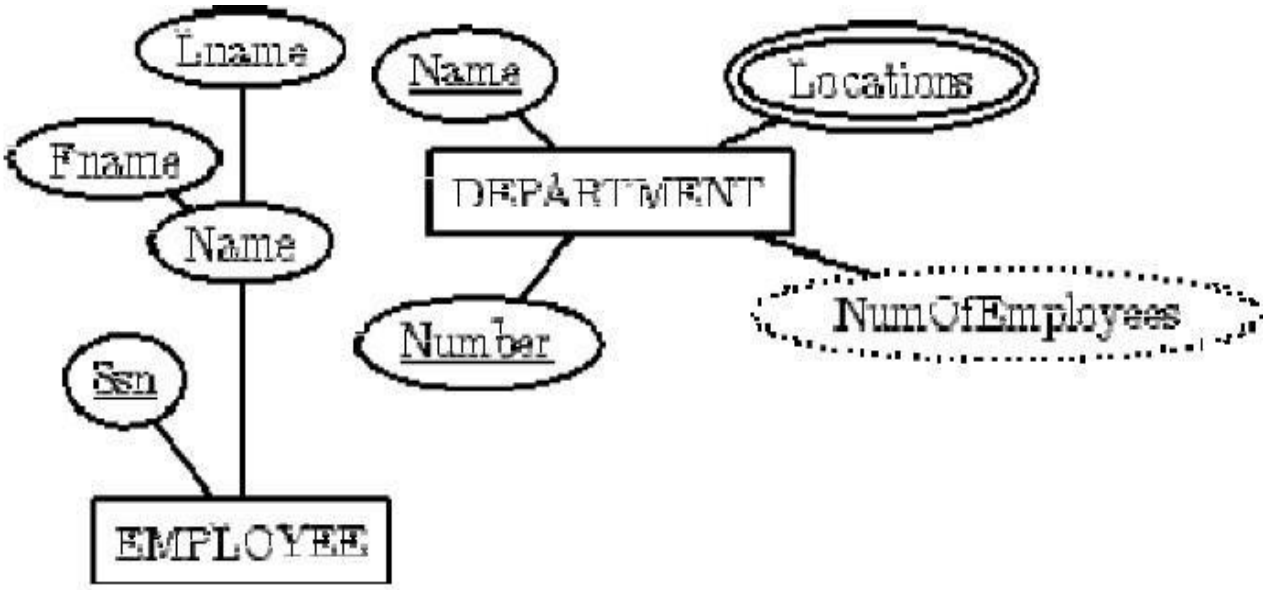
- Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate
  relation, and using the assignment operation, denoted by ← (left arrow), as follows:

DEP5_EMPS ← σDno=5(EMPLOYEE)
RESULT ← πFname, Lname, Salary(DEP5_EMPS)

2.Discuss ER to Relational Mapping algorithm with example for each step.

2.1 Relational Database Design using ER-to-Relational mapping.

Step 1: For each **regular (strong) entity type** E in the ER schema, create a relation R that includes all the simple attributes of E.



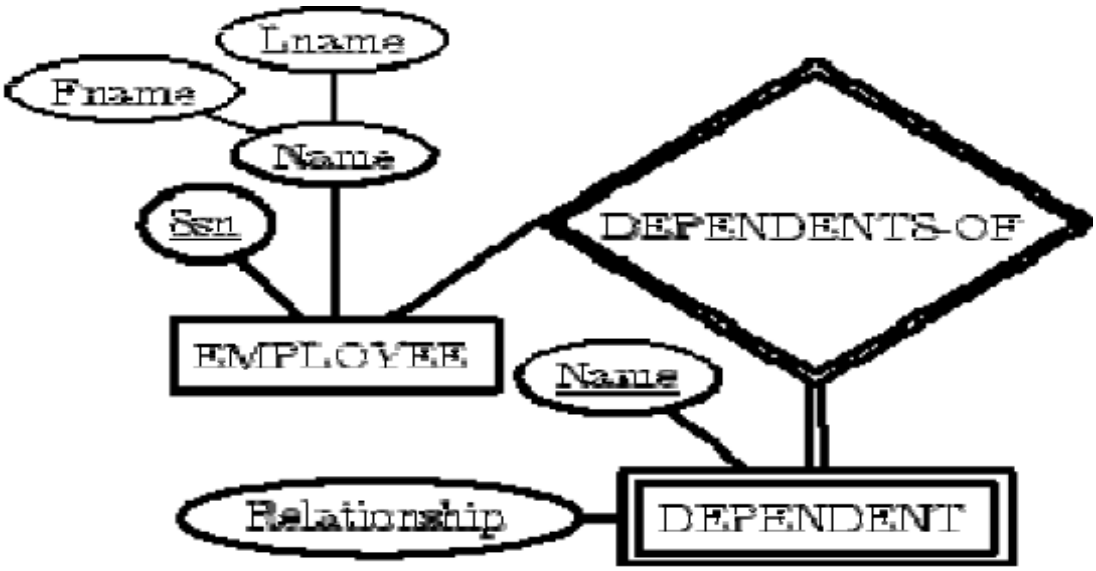
EMPLOYEE

SSN	Lname	Fname

DEPARTMENT

NUMBER	NAME

Step 2: For each **weak entity type** W in the ER schema with owner entity type E, create a relation R, and include all simple attributes (or simple components of composite attributes) of W as attributes. In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s).

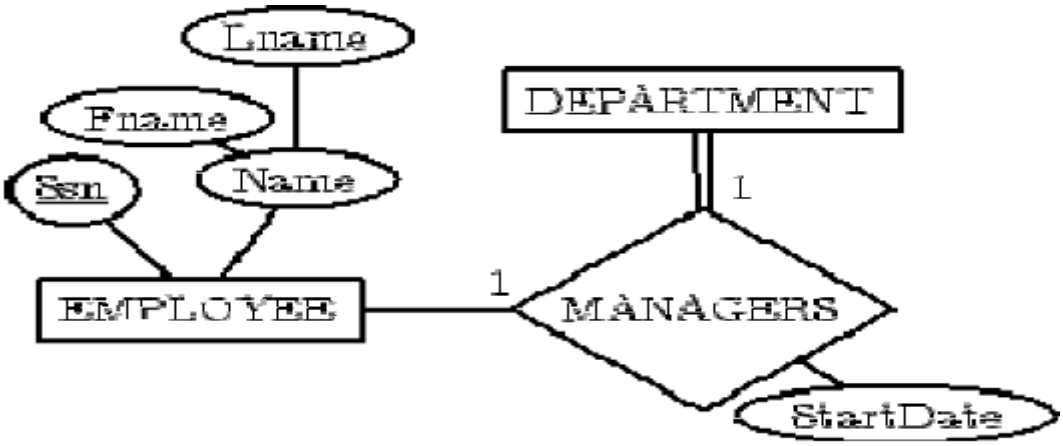


DEPENDENT

EMPL-SSN	NAME	Relationship

Step 3: For each **binary 1:1 relationship type** R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations, say S, and include the primary key of T as a foreign key in S. Include all the simple attributes of R as attributes of S.

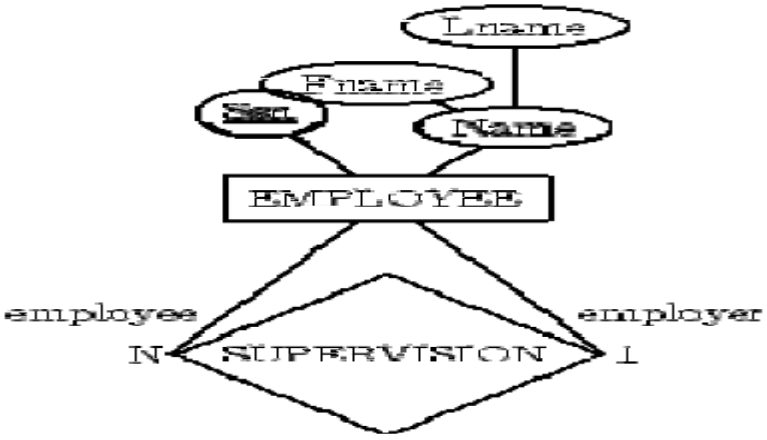
Step 4: For each regular **binary 1:N relationship type** R identify the relation (N) relation S.



DEPARTMENT

MANAGER-SSN	StartDate

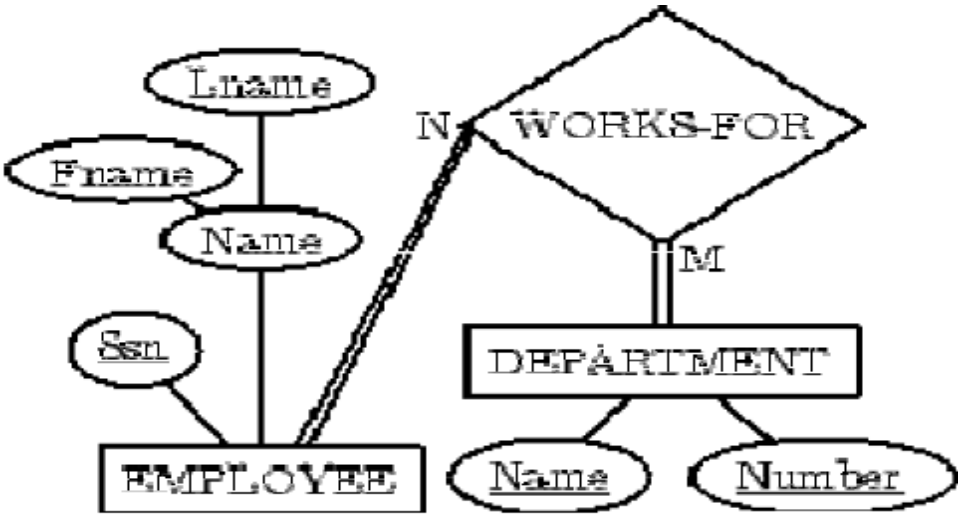
the primary key of T as a foreign key of S. Simple attributes of R map to attributes of S.



EMPLOYEE

SupervisorSSN

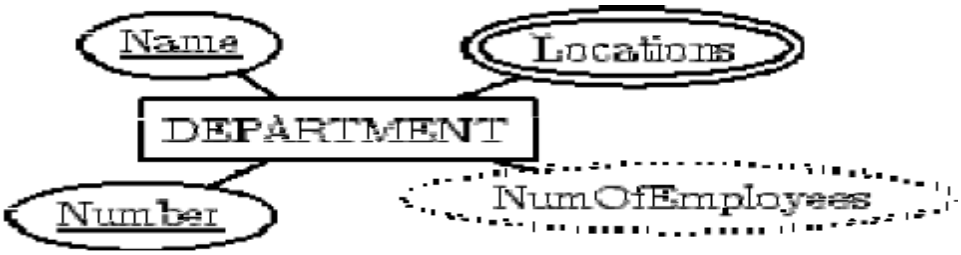
Step 5: For each **binary M:N relationship type** R, create a relation S. Include the primary keys of participant relations as foreign keys in S. Their combination will be the primary key for S. Simple attributes of R become attributes of S.



WORKS-FOR

EmployeeSSN	DeptNumber

Step 6: For each **multi-valued attribute A**, create a new relation R. This relation will include an attribute corresponding to A, plus the primary key K of the parent relation (entity type or relationship type) as a foreign key in R. The primary key of R is the combination of A and K.



DEP-LOCATION

Location	DEP-NUMBER

Step 7: For each **n-ary relationship type R**, where n>2, create a new relation S to represent R. Include the primary keys of the relations participating in R as foreign keys in S. Simple attributes of R map to attributes of S. The primary key of S is a combination of all the foreign keys that reference the participants that have cardinality constraint > 1. For a recursive relationship, we will need a new relation.

3a.With a neat diagram explain transition diagram of a transaction.

Ans:

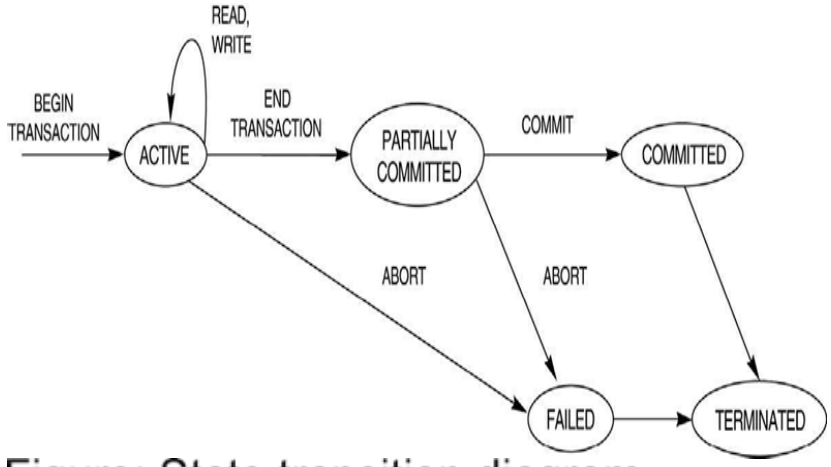


Figure: State transition diagram

- A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system keeps track of start of a transaction, termination, commit or aborts.
  - **BEGIN_TRANSACTION**: marks the beginning of transaction execution
  - **READ or WRITE**: specify read or write operations on the database items that are executed as part of a transaction
  - **END_TRANSACTION:** specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution
  - **COMMIT_TRANSACTION**: signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone
  - **ROLLBACK**: signals that the transaction has *ended unsuccessfully*, so that any changes or effects that the transaction may have applied to the database must be **undone**

- A transaction goes into **active state** immediately after it starts execution and can execute read and write operations.
- When the transaction ends it moves to **partially committed** state.
- At this end additional checks are done to see if the transaction can be committed or not. If these checks are successful the transaction is said to have reached commit point and enters **committed state.** All the changes are recorded permanently in the db.
- A transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its write operation.
- Terminated state corresponds to the transaction leaving the system. All the information about the transaction is removed from system tables.

3b.Why are concurrency control and recovery needed in DBMS? Explain 4 types of problems that may occur when two simple transactions run concurrently.
Ans:
Concurrency control and recovery are crucial in DBMS because they ensure data consistency and integrity, especially when multiple transactions run simultaneously. Concurrency control manages the execution of multiple transactions to prevent conflicts and maintain data integrity, while recovery mechanisms protect the database from failures, ensuring data can be restored to a consistent state. Four common problems that can occur when two simple transactions run concurrently are dirty reads, lost updates, unrepeatable reads, and phantom reads.

Concurrency Control:

Concurrency control aims to prevent interference between concurrent transactions by ensuring that the database remains in a consistent state even when multiple transactions are running simultaneously. Without proper concurrency control, the database could be corrupted, leading to inaccurate data or even application crashes.

Recovery:

Recovery mechanisms handle failures, such as system crashes or hardware failures, by restoring the database to a consistent state. Without recovery, the database could be left in an inconsistent state, leading to data corruption or loss.

Four Problems with Concurrent Transactions:

1. Dirty Reads:

One transaction reads data that is being updated by another, uncommitted transaction. This can lead to an inaccurate view of the data, as the updated data may not yet be committed.

2. Lost Updates:

Two transactions attempt to update the same data simultaneously, and one transaction's update is lost or overwritten by the other. This can lead to inconsistencies and incorrect data.

3. Unrepeatable Reads:

A transaction reads a value, and then another transaction updates that value and commits the change. If the first transaction attempts to read the same value again, it may receive a different value. This can lead to inconsistencies and incorrect data.

4.Phantom Reads:

A transaction reads a set of records, and then another transaction inserts new records that satisfy the same query criteria. If the first transaction attempts to read the same set of records again, it may find that the new records have been inserted. This can lead to inconsistencies and incorrect data.

4a.What is Normalization.Why it is needed?

Ans:

Normalization is the process of minimizing data redundancy and inconsistency from a relation (table) or from a set of relations. The data redundancy in relation may cause insertion, deletion, and updating anomalies.

To address the anomalies and to make the database as consistent, the normal forms are used. Using normal forms we can eliminate or reduce redundancy in database tables.

b.Explain 1NF,2NF,3NF, BCNF with an example

Ans:

The domain of an attribute must include only atomic values and the value of any attribute in a tuple must be a single value from the domain of that attribute.A table is considered to be in 1NF if all the fields contain only single values (instead of list of values).

Example (Not in 1 - NF)

ISBN	Title	AuName	AuPhone	PubName	PubPhone	Price
0-321-32132-1	Balloon	Sleepy, Snoopy, Grumpy	321-321-1111, 232-234-1234, 665-235-6532	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Jones, Smith	123-333-3333, 654-223-3455	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Joyce	666-666-6666	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Roman	444-444-4444	Big House	123-456-7890	$25.00

1. Place all items that appear in the repeating group in a new table
2. Designate a primary key for each new table produced.
3. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.

ISBN	Title	PubName	PubPhone	Price
0-321-32132-1	Balloon	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Big House	123-456-7890	$25.00

ISBN	AuName	AuPhone
0-321-32132-1	Sleepy	321-321-1111
0-321-32132-1	Snoopy	232-234-1234
0-321-32132-1	Grumpy	665-235-6532
0-55-123456-9	Jones	123-333-3333
0-55-123456-9	Smith	654-223-3455
0-123-45678-0	Joyce	666-666-6666
1-22-233700-0	Roman	444-444-4444

Second Normal Form

- A second normal is a method of arranging attributes semantically (logically) based on the constraints 1) a relation must be in first normal form and 2) relation should not contain any partial dependency.

- No non-prime attribute (attribute which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

- The partial dependency - is the proper subset of candidate key determines non-primary attribute in a relation.

- Every non-key attribute is fully functionally dependent on the primary key. Thus, no non-key attributes are functionally dependent on the part (but not all) of the primary key. That means, no partial dependency exists.

- Note: If a key is single attribute, then it is always in 2nd Normal form.

  Example 2 (Not 2NF)

  Scheme □ {City, Street, HouseNumber, HouseColor, CityPopulation}

  1. key □ {City, Street, HouseNumber}

2. {City, Street, HouseNumber} □ {HouseColor}
3. {City} □ {CityPopulation}
4. CityPopulation does not belong to any key.
5. CityPopulation is functionally dependent on the City which is a proper subset of the key

Example 2 (Convert to 2NF)

Old Scheme □ {City, Street, HouseNumber, HouseColor, CityPopulation}

New Scheme □ {City, Street, HouseNumber, HouseColor}

New Scheme □ {City, CityPopulation}

Example 2 (Convert to 2NF)

Old Scheme □ {Studio, Movie, Budget, StudioCity}

New Scheme □ {Movie, Studio, Budget}

New Scheme □ {Studio, City}

Example 3 (Not 2NF)
Scheme □ {studio, movie, budget, studio_city}

1. Key □ {studio, movie}
2. {studio, movie} □ {budget}
3. {studio} □ {studio_city}
4. studio_city is not a part of a key
5. studio_city functionally depends on studio which is a proper subset of the key

Example 3 – Decomposed into 2NF

Example 3 (Convert to 2NF)

Old Scheme □ {City, Street, HouseNumber, HouseColor, CityPopulation}

New Scheme □ {City, Street, HouseNumber, HouseColor}

New Scheme □ {City, CityPopulation}

Third Normal Form

This form dictates that all non-key attributes of a table must be functionally dependent on a candidate key i.e. there can be no interdependencies among non-key attributes.

For a table to be in 3NF, there are two requirements

- The table should be second normal form
- No attribute is transitively dependent on the primary key

Example (Not in 3NF)
Scheme □ {Studio, StudioCity, CityTemp}
1. Primary Key □ {Studio}

2. {Studio} □ {StudioCity}
3. {StudioCity} □ {CityTemp}
4. {Studio} □ {CityTemp}
5. Both StudioCity and CityTemp depend on the entire key hence 2NF
6. CityTemp transitively depends on Studio hence violates 3NF

Example (Convert to 3NF)

Old Scheme □ {Studio, StudioCity, CityTemp}

New Scheme □ {Studio, StudioCity}

New Scheme □ {StudioCity, CityTemp}

Boyce-Codd Normal Form (BCNF)

A relation schema 'R' is in BCFN with respect to a set of 'F' of functional dependencies if, for all functional dependencies they are in the form α □ β where, α, β ⊂= R, at least of the following holds:

- α □ β is a trivial FD (β ⊂= α)
- α is the super key for schema R

A relation is in BCNF if every determinant is a candidate key.

- **BCNF does not allow dependencies between attributes that belong to candidate keys.**
- BCNF is a refinement of the third normal form in which it drops the restriction of a non-key attribute from the 3rd normal form.

Third normal form and BCNF are not same if the following conditions are true:

- The table has two or more candidate keys
- At least two of the candidate keys are composed of more than one attribute
- The keys are not disjoint i.e. The composite candidate keys share some attributes

BCNF - Decomposition

1. Place the two candidate primary keys in separate entities
2. Place each of the remaining data items in one of the resulting entities according to its dependency on the primary key.

Example 1 - Address (Not in BCNF)

Scheme □ {City, Street, ZipCode }

1. Key1 □ {City, Street }
2. Key2 □ {ZipCode, Street}
3. No non-key attribute hence 3NF
4. {City, Street} □ {ZipCode}
5. {ZipCode} □ {City}
6. Dependency between attributes belonging to a key

Example 1 (Convert to BCNF)

Old Scheme □ {City, Street, ZipCode }

New Scheme1 □ {ZipCode, Street}

New Scheme2 □ {City, Street}

Loss of relation {ZipCode} □ {City}

Alternate New Scheme1 □ {ZipCode, Street }

Alternate New Scheme2 □ {ZipCode, City}

5a. What is document based NOSQL systems? Explain basic operations CRUD in MongoDB.

Ans:

Document based Databases

Imagine you have a folder for every person in your class. Each folder has different things—some have drawings, some have stories, some have both. You don't need every folder to look the same.

That's how document databases work, they keep everything about one thing in one place, and each one can look different!

- Document-based databases store data as documents, usually in JSON or BSON format.
  Each document can have its own structure, unlike SQL tables that require fixed columns.
  Ideal when:
  - You have varying types of data for each record
  - You want to retrieve entire "objects" easily (like a blog post, product, or user profile)
- Fast and flexible – easy to update, and great for agile development.

Use Cases:

Content Management Systems
E-commerce product catalogs
User profile storage

CRUD Operations

```
// Syntax to create a collection:
db.createCollection(name, options);
// Example: db.createCollection("users");
```

1. Create (C)

To create a new document or insert data into a collection:

```
db.collection.insertOne({ key: value });
// Example:
db.users.insertOne({ name: "Alice", age: 30, email: "alice@example.com" });
```

You can also insert multiple documents at once using insertMany():

```
db.collection.insertMany([
{ key1: value1 },
{ key2: value2 },
// More documents...
```

```
]);
// Example:
db.users.insertMany([
{ name: "Bob", age: 25, email: "bob@example.com" },
{ name: "Charlie", age: 35, email: "charlie@example.com" }
]);
```

2. Read (R)

To retrieve or read documents from a collection:

```
// Find all documents in a collection
db.collection.find();
// Find documents that match a specific condition
db.collection.find({ condition });
// Example:
db.users.find(); // Find all documents in the 'users' collection
db.users.find({ age: { $gt: 25 } }); // Find users where age is greater than 25
```

3. Update (U)

To update existing documents in a collection:

```
// Update a single document that matches a condition
db.collection.updateOne({ filter }, { $set: { update } });
// Update multiple documents that match a condition
db.collection.updateMany({ filter }, { $set: { update } });
// Example:
db.users.updateOne({ name: "Alice" }, { $set: { age: 31 } }); // Update Alice's age
to 31
db.users.updateMany({ age: { $lt: 30 } }, { $inc: { age: 1 } }); // Increment age by
1 for all users under 30
```

4. Delete (D)

To delete documents from a collection:

```
// Delete a single document that matches a condition
db.collection.deleteOne({ filter });
// Delete all documents that match a condition
db.collection.deleteMany({ filter });
// Example:
db.users.deleteOne({ name: "Bob" }); // Delete the document where name is Bob
db.users.deleteMany({ age: { $gte: 40 } }); // Delete all users who are 40 years or
Older
```

5b.Explain CAP theorem

Ans:

Imagine you're sharing information with your friends over walkie-talkies. Sometimes, one friend's walkie-talkie doesn't work (like a broken network).

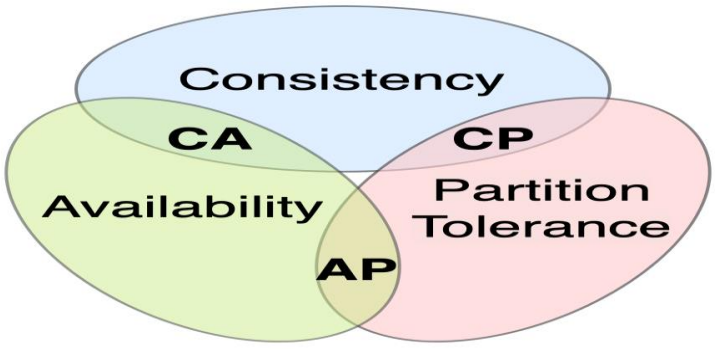
Now, you have to choose between:

- Making sure everyone gets the same number of information(Consistency),
- Making sure everyone gets a reply when they ask for information(Availability),
- Or keeping things going even when some walkie-talkies don't work (Partition Tolerance).

CAP Theorem applies to distributed systems (like NoSQL databases spread over servers).

It says a system can only guarantee two of the following three:

1. Consistency (C): All nodes see the same data at the same time.
   (Like a bank – your balance is always the same everywhere)
2. Availability (A): Every request gets a response, even if it's not the most recent data.
   (Like WhatsApp – you always get a reply, even if slow)
3. Partition Tolerance (P): The system keeps working even if there's a communication failure between nodes.
   (Like servers in two cities losing connection but still running)
   Example:
   MongoDB chooses Availability + Partition Tolerance, which means it may temporarily return slightly outdated data to stay available when there's a network issue.



6.Explain the Concurrency control based on Timestamp ordering
Ans:

- The idea for this scheme is to order the transactions based on their timestamps.
- A schedule in which the transactions participate is then serializable, and the only equivalent serial schedule permitted has the transactions in order of their timestamp values. This is called **timestamp ordering (TO)**.
- The algorithm must ensure that, for each item accessed by *conflicting Operations* in the schedule, the order in which the item is accessed does not violate the timestamp order.
- To do this, the algorithm associates with each database item X two timestamp (**TS**) values:
    1. **read_TS(X).** The **read timestamp** of item X is the largest timestamp among all the timestamps of transactions that have successfully read item X—that is, read_TS(X) = TS(T), where T is the *youngest* transaction that has read X successfully.
    2. **write_TS(X).** The **write timestamp** of item X is the largest of all the timestamps of transactions that have successfully written item X—that is, write_TS(X) = TS(T), where T is the *youngest* transaction that has written X successfully.

Thomas's Write Rule

- A modification of the basic TO algorithm, known as **Thomas's write rule**, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows:
    1. If read_TS(X) > TS(T), then abort and roll back T and reject the operation.
    2. If write_TS(X) > TS(T), then do not execute the write operation but continue processing. This is because some transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of X. Thus, we must ignore the write_item(X) operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).

  If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).