2018 Exam - Information Management
CS3041

NSTRUCTOR: Vincent Wade & Seamus Lawless
vincent.wade@scss.tcd.ie
seamus.lawless@scss.tcd.ie

Two Hour Exam

Question 1 Mandatory

Answer 2 out of 3 Questions out of Q2-4

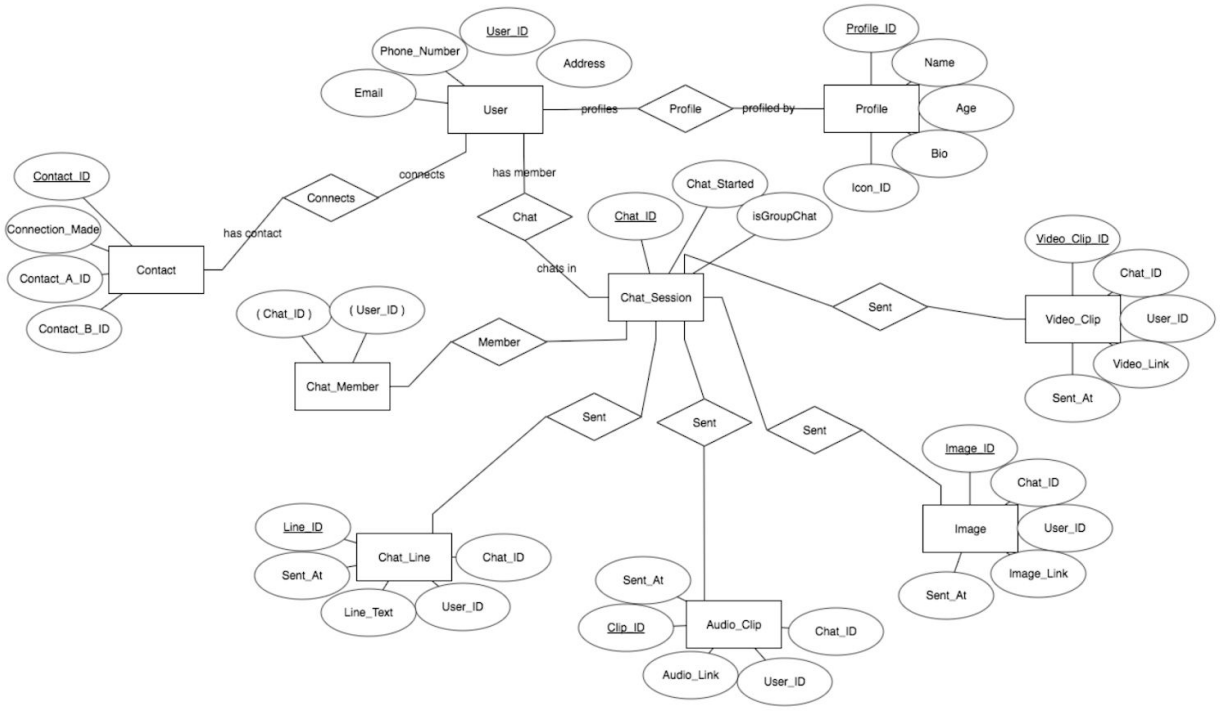
25 marks per question

40 mins per question

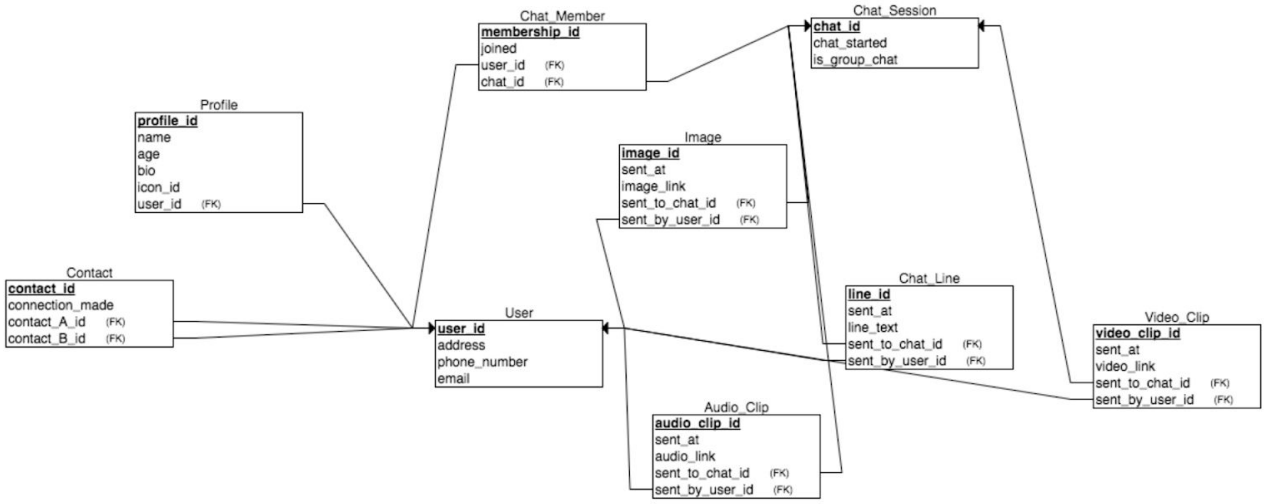
1% per mark

Question 1

ERD of Pied Piper & PiperChat

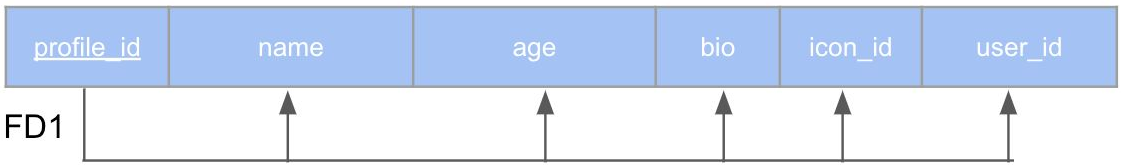


Relational Schema Mapping of Pied Piper & PiperChat

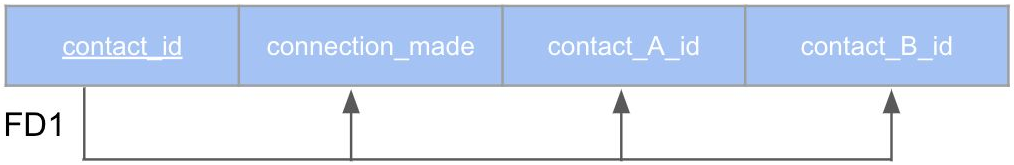


Functional Dependency Diagrams

PROFILE

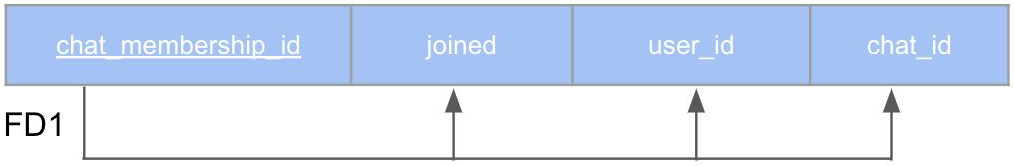


CONTACT

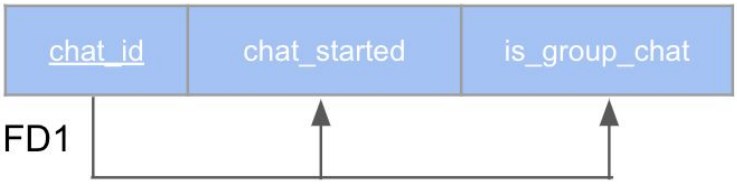


USER

CHAT_MEMBERSHIP



CHAT

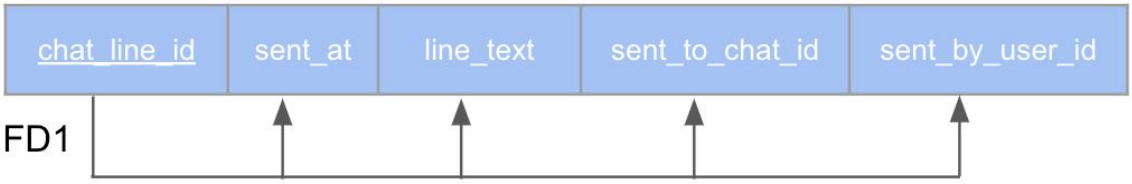


IMAGE

VIDEO_CLIP

AUDIO_CLIP

CHAT_LINE



Boyce Codd Normalisation

1st Normal Form

- All attributes atomic (can't be broken down further)
- Every row-column intersection must have only one value

2nd Normal Form

- 1NF Compliant
- Every non-key column is functionally dependant on the primary key

3rd Normal Form

- 2NF Compliant
- No non-key attributes are transitively dependent upon the primary key

Boyce-Codd Normal Form

- For all FD's X=>Y, X is a superkey of R

Adding Type to Contact

```
ALTER TABLE Contact

ADD Type varchar(255),

CHECK (Type='Friend' OR Type='Family' OR Type='Work Colleague');
```

Adding Type to Chat

```
ALTER TABLE Chat

ADD Type varchar(255),

CHECK (Type='Friend' OR Type='Family' OR Type='Work Colleague');
```

Ensure Adding Contact of Correct Type to Chat

```
CREATE TRIGGER verify_correct_chat_type BEFORE INSERT ON Chat
Member

FOR EACH ROW

BEGIN

     IF (SELECT Type FROM Chat WHERE chat_id = NEW.chat_id)
!=(SELECT Type FROM Contact WHERE contact_A_id = NEW.added_by
AND contact_B_Id = NEW.user_id)

     THEN SIGNAL SQLSTATE '45000'

          SET MESSAGE_TEXT 'Contact and chat type must match'

     END IF;

END;
```

Creating User Table

```
CREATE TABLE  'User' (

     'user_id' int(11), NOT NULL AUTO_INCREMENT,

     'address' varchar(25), NOT NULL,

     'phone_number' varchar(13), NOT NULL,

     'email' varchar(25), NOT NULL,

     PRIMARY KEY ('user_id'),

     UNIQUE KEY 'user_id_UNIQUE' ('user_id')

)
```

Creating Profile Table

```
CREATE TABLE  Profile (
     'profile_id' int(11), NOT NULL AUTO_INCREMENT,
     'name' varchar(25), NOT NULL,
     'age' tinyint(4), NOT NULL,
     'bio' varchar(255), NOT NULL,
     'icon_id' int(11), NOT NULL,
     'user_id' int(11), NOT NULL,
     PRIMARY KEY ('profile_id'),
     UNIQUE KEY 'profile_id_UNIQUE' ('profile_id'),
     CONSTRAINT ('FK_icon_id') FOREIGN KEY ('icon_id')
REFERENCES 'Image'('image_id'),
     CONSTRAINT ('FK_user_id') FOREIGN KEY ('user_id')
REFERENCES 'User'('user_id'),
)
```

Change User Name

```
UPDATE Profile
SET 'name' = 'x'
WHERE 'user_id' = 'y'
```

Change User Name

```
UPDATE Profile

SET 'name' = 'x'

WHERE 'user_id' = 'y'
```

Get All Chats Jared Dunn was a Member of in 2017

```
SELECT * FROM Chat_Session

WHERE chat_id = (SELECT chat_id FROM Chat_Member WHERE user_id =
(SELECT user_id FROM Profile WHERE name = 'Jared Dunn')) AND
chat_started > '01-01-2017' AND chat_started < '31-12-2017'
```

Get All Video Chats with More Than 10 Participants

```
SELECT * FROM Chat_Session

WHERE chat_id = (SELECT chat_id FROM (SELECT chat_id, COUNT(*)
FROM Chat_Member GROUP by chat_id HAVING COUNT(*) > 8)
```

Question 2

a) *What are the four properties of a desirable transaction? Explain each of them.*

Atomicity: Each transaction should be an atomic unit of processing. It should be performed in its entirety or not at all.

Consistency Preservation: A transaction should preserve the consistency of a DB. Each transaction should take the DB from one consistent state to another.

Isolation: Each transaction should appear as if acting in isolation. The execution of one transaction should not be interfered with by another transaction.

Durability: The changes applied by a transaction must persist in the database. The changes must not be lost due to a failure.

b) *Define the properties of a serial schedule. What can be assumed about a serial schedule and what are the potential issues. Discuss serializability as a way of addressing these issues. How is serializability measured?*

A serial schedule is a series of transactions that are executed consecutively one after another. The commit or rollback of one transaction signals the start of the next transaction. All transactions in a serial schedule are said to be correct and should succeed however in a production-scale database they are extremely slow and are not efficient enough.

Non-serialized schedules however can be serialized using *equivalence*. This allows for them to be converted to a serialized schedule thus guaranteeing the success of transactions.

Result Equivalence: When the operating of A and B produce the same result regardless of the the order in which they are executed they can be considered to be result equivalent.

Conflict Equivalence: If the order of any two conflicting transactions is the same in both a serial schedule and non-serial schedule then they can be considered conflict equivalent.

c) *Explain how concurrency control algorithms which are based upon locking techniques ensure that concurrently executing transactions do not interfere with each other's execution. Make reference to both binary and read-write locking. Two-phase locking is an additional locking protocol. Discuss two-phase locking and the benefit it offers.*

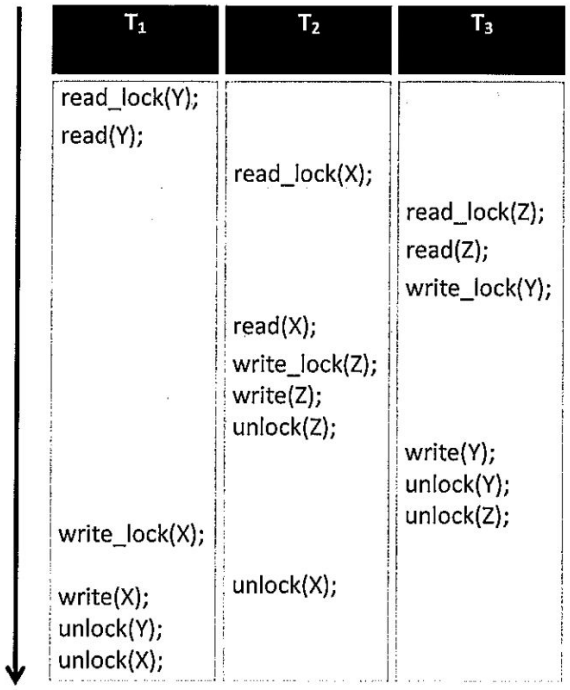
Concurrency control protocols are implemented by the DB and attempt to ensure concurrency exists within transactions and that all schedules are correct. They make use of the following to ensure concurrency:

Locking Protocols: Data items within transactions are locked when they are accessed/read. There are two main types of locks; Binary and Read/Write. Binary locks maintain two states; locked or unlocked. When a transaction attempts to access a resource it must first attempt to lock it and when it is finished it must also unlock this resource. Read/Write locks are more flexible as they allow for data items to be concurrently read by multiple transactions concurrently however only one transaction can obtain a write-lock on a resource at any given time.

Timestamps: Timestamps are used to identify transactions based upon start times and are also used to verify who had access to a given lock first and who may have priority to access a resource next.

Two-Phase Locking: This concerns the positioning of locking and unlocking in every transactions. Locking and unlocking can be performed in two phases/cycles known as the expanding phase and the shrinking phase. In the expanding phase new locks can be acquired but no locks can be released. In the shrinking phase, existing locks can be released buy no new locks can be acquired. If transactions follow this locking protocol the schedule is said to be serializable however it also comes at an extra cost. Transactions might not be able to release a given resource when they are finished with it and must wait until the next cycle. This reduces concurrency significantly and increases query execution latency.

d) Outline the operation of the "Wait-Die" algorithm. Indicate how "Wait-Die" would execute on the following schedule. Assume T_1 is older than T_2 etc. State any assumptions you make in determining the transaction operation ordering.



T_1	T_2	T_3
read_lock(Y);		
read(Y);		
	read_lock(X);	
		read_lock(Z);
		read(Z);
		write_lock(Y);
	read(X);	
	write_lock(Z);	
	write(Z);	
	unlock(Z);	
		write(Y);
		unlock(Y);
		unlock(Z);
write_lock(X);		
	unlock(X);	
write(X);		
unlock(Y);		
unlock(X);		

"Wait-Die": This is an algorithm designed to prevent deadlock occurring within a transaction schedule. It considers transactions in regards to whom occurred first. This is decided using unique timestamps. If T_a tries to lock an item X which is currently held by T_b it is allowed to wait iff T_a is older than T_b (T_a occurred first). If this is not the case T_a is aborted and restarted after a given time delay.

In the below solution we are assuming that the time delay is a set constant of *t* that is larger than the total time required to complete any of the transactions above.

Conflicts

- $r_1(Y)$ and $w_3(Y)$ - T_3 is aborted and pushed back to a given time since younger
- $w_2(Z)$ and $r_3(Z)$ - T_2 would be allowed to wait, but T_3 has already been aborted
- $w_1(X)$ and $r_2(X)$ - T_1 is allowed to wait since older than T_2

e) Compare and contrast pure timestamp-based concurrency control with lock-based techniques. What problem(s) does timestamp ordering prevent.

Timestamp concurrency control methods do not use locking whatsoever and as a result deadlock cannot occur. For each data item accessed by a conflicting operation in a schedule the order of access must not violate timestamp ordering. This produces a serializable schedule as all operations are conducted in the order they would be in a serial schedule whilst still allowing non-conflicting operations to execute concurrently.

f) What timestamp values must be stored for each data item in a database? What process occurs when a transaction issues a Read, and when a transaction issues a Write?

The DBMS keeps track of two timestamps per resource:

1. *read_TS(X)* - Timestamp of the youngest transaction that has most recently successfully read X
2. *write_TS(X)* - Timestamp of the youngest transaction that has most recently successfully wrote to X

If a transaction T attempts to write to X and *read_TS(X)* or *write_TS(X)* are greater than TS(T) then abort and rollback. If less than, execute write operation and set *write_TS(X)* to TS(T).

If a transaction T attempts to read an X and *read_TS(X)* is greater than TS(T) then abort and rollback. If less than, execute write operation and set *read_TS(X)* to TS(T).

Question 3

a) What are privileges, and why are they used in database systems? Describe the two levels of privilege used and identify example commands from each level. Describe, with the aid of the appropriate SQL commands, how privileges can be assigned and removed.

Privileges are certain granted or restricted characteristics of a user functional ability. They are used in database systems to both allow and restrict DB users from performing certain operations/queries. Privileges can be defined on two levels:

1. Account Level: DBA can specify the privileges that each account holds independently of the relations of the database.

```
CREATE USER 'admin'@'%' IDENTIFIED BY 'password'

GRANT ALL PRIVILIGES ON *.* TO 'admin'@'%'
```

2. Relational Level: DBA can specify the privileges that each account holds independently of the relations of the database.

```
CREATE USER 'result_clerk'@'%' IDENTIFIED BY 'password'

GRANT SELECT, INSERT ON Result TO 'result_clerk'@'%'
```

Granting priviliges to a user is performed by:

```
GRANT ALL PRIVILIGES ON *.* TO 'admin'@'%'
```

Revoking priviliges from a user is performed by:

```
REVOKE ALL PRIVILIGES ON *.* FROM 'admin'@'%'
```

b) Privilege propagation is an important aspect of Discretionary Access Control. Describe propagation, making specific reference to the dangers involved.

Propogation allows for a given user X to grant priviliges to another user Y and allow Y to pass such privileges on to another user Z.

```
GRANT ALL PRIVILIGES ON Result TO 'result_clerk'@'%' WITH GRANT OPTION
```

This means that result_clerk now also has the ability to grant these priviliges to any user he/she wishes. This can be quite dangerous as even after you have revoked result_clerk privileges, any privileges that he has granted cannot be revoked and still exist.

c) Compare and Contrast Discretionary Access Control and Mandatory Access Control.

Discretionary Access Control is handled from a user level and is a security enforcement for events that you can expect to occur. Such an example is by granting priviliges and revoking priviliges. It is a form of access control fully at your discretion.

Mandatory Access Control defines security levels for both data and users. Certain users will have access to certain relations/features based on their security level/clearance. This is not commonly available in commercial DBMS' however is desirable for government and military intelligence.

d) What is a database constraint? Distinguish between explicit constraints and semantic constraints. Define three basic types of integrity constraint that all relational databases must support.

Database Constraints are restrictions or rules imposed against both the relational schema that ensure the integrity of the database and the validity of the data.

Explicit Constraints are constraints that are expressed directly within the relational schema.

Semantic Constraints are constraints that cannot be expressed in the relational schema. They are usually enforced by application programs,

The three types of integrity constraints that all databases must support are:

1. **Key:** Primary key, candidate keys. No duplicate values of primary key.
2. **Entity Integrity:** No NULL values in primary key.
3. **Referential Integrity:** Foreign key in one table refers to primary key of another.

e) What operations on a database can violate referential integrity? What clauses and constraints can be used to avoid violating referential integrity?

The SQL commands `INSERT, DELETE, UPDATE` can all violate referential integrity. The DBMS could automatically propagate these operations such as `DELETE` to all affected tuples that may refer to the deleted tuple. Similar to this, if we wanted to `UPDATE` the primary key of a current tuple that is referenced elsewhere we would once again violate the referential integrity of the primary key. This change can also be propagated or cascaded to all affected tuples. An alternative to propagation/cascading is to update the affected foreign keys to either NULL or a default value.

```
CREATE TABLE child (
	id INT,
	parent_id INT,
	FOREIGN KEY (parent_id)
		REFERENCES parent(id)
		ON DELETE CASCADE
) ENGINE=INNODB;
```

e) Why are views important and what can they be used to restrict? Using the appropriate SQL command create a new view which is used to display information from at least two of your tables from question 1.

Views are important within a database as they allow us to generate condensed/specific sub and cross-relational views. This functionality allows the view designer to decide what attributes are desired to be kept restricted. This way users can be granted access on a view whilst maybe not having access to users credit card details etc.

```
CREATE VIEW managers_overview AS

SELECT m.manager_name, t.team_name

FROM Team t, Manager m

WHERE t.team_id=m.team_id
```

Question 4

a) Given the two tables below, provide a JSON representation for a document database. Briefly explain the main difference between the two representations in terms of data normalisation.

The main difference between the two representations is that John Murphy and all of his related data is considered to be one object.

In a relational database John Murphy is considered to be an author, and his post is considered a separate entity completely and they are housed in different relations.

In JSON however, John and his attributed post are all considered to be the one artefact and are referenced all within the one document.

b) Provide a practical example where the use of graph databases is more beneficial than relational databases. Explain why graph databases would be more suitable in this situation.

In Graph databases the relationship between nodes is considered just as important as the data contained within the node itself. It allows entries to be broken into objects and represented as nodes with individual relationships between nodes. Graph databases are extremely useful when defining schemas around networks.

Graph databases would be particularly useful in the above situation as there is a direct relationship between each author and the posts he/she has posted. By using graph databases you could easily query upon this relationship and find all of the posts written by a given author directly by querying the relationship between a given author and posts. In a relational databases you would have to execute a query to find all posts with a given ID.

c) What does BASE stand for and what is its relationship to ACID?

ACID

- **A**tomic: Everything in a transaction succeeds or the entire transaction is rolled back
- **C**onsistent: A transaction cannot leave the database in an inconsistent state
- **I**solated: Transactions cannot interfere with each other
- **D**urable: Completed transactions persist, even when servers restart etc.

BASE

- **B**asic **A**vailability: An application works basically all the time
- **S**oft-state: It does not have to be consistent all the time
- **E**ventual consistency: It will be in some known-state state eventually

Each node is always available to serve requests. As a trade-off, data modifications are propagated in the background to other nodes. The system may be inconsistent, but the data is still largely accurate.

ACID refers to the desired properties of a valid transaction within a relational database whilst BASE refers to the desired properties of a NoSQL database/platform. NoSQL databases are designed to address performance and scalability requirements of the web and allow for near-instant query time. This comes with some slight downfalls and requires sharding and for certain data to not be available all of the time.

The BASE system drops the consistency part of the CAP theorem and aims to provide availability and partition tolerance.

d) What does that acronym CAP stand for? Describe the CAP theorem.

CAP stands for:

Consistency: This states that all node must see the same data at the same time. In other words all data within the database system must be coherent across all nodes.

Availability: This states that every request to a database system must receive a response whether it be a success or a failure. No requests must result in an endless loop or stall.

Partition Tolerance: In the event of a malfunctioning node, this will not cause the entire data network to go offline and out of order. This allows for data nodes to be partitioned/fragmented into multiple clusters allowing a continous operation even in the event of a node failure.

The CAP theorem states that it is near impossible to achieve all of the above three aspects at the same time within a given solution. More often than not you must substitute one of the above criteria.

e) List at least five characteristics that distinguish NoSQL databases from relational databases.

- No strict schemas imposed.
- Easier to update, maintain and scale.
- Can handle unstructured data forms (texts, images, video).
- Sharding across multiple clusters and sources.
- Open source, cheaper
- No defined structure, can re-model on the fly
- Horizontally scalable (more nodes) vs vertically scalable (more CPU)
- Table based vs document, graph, key-value based
-