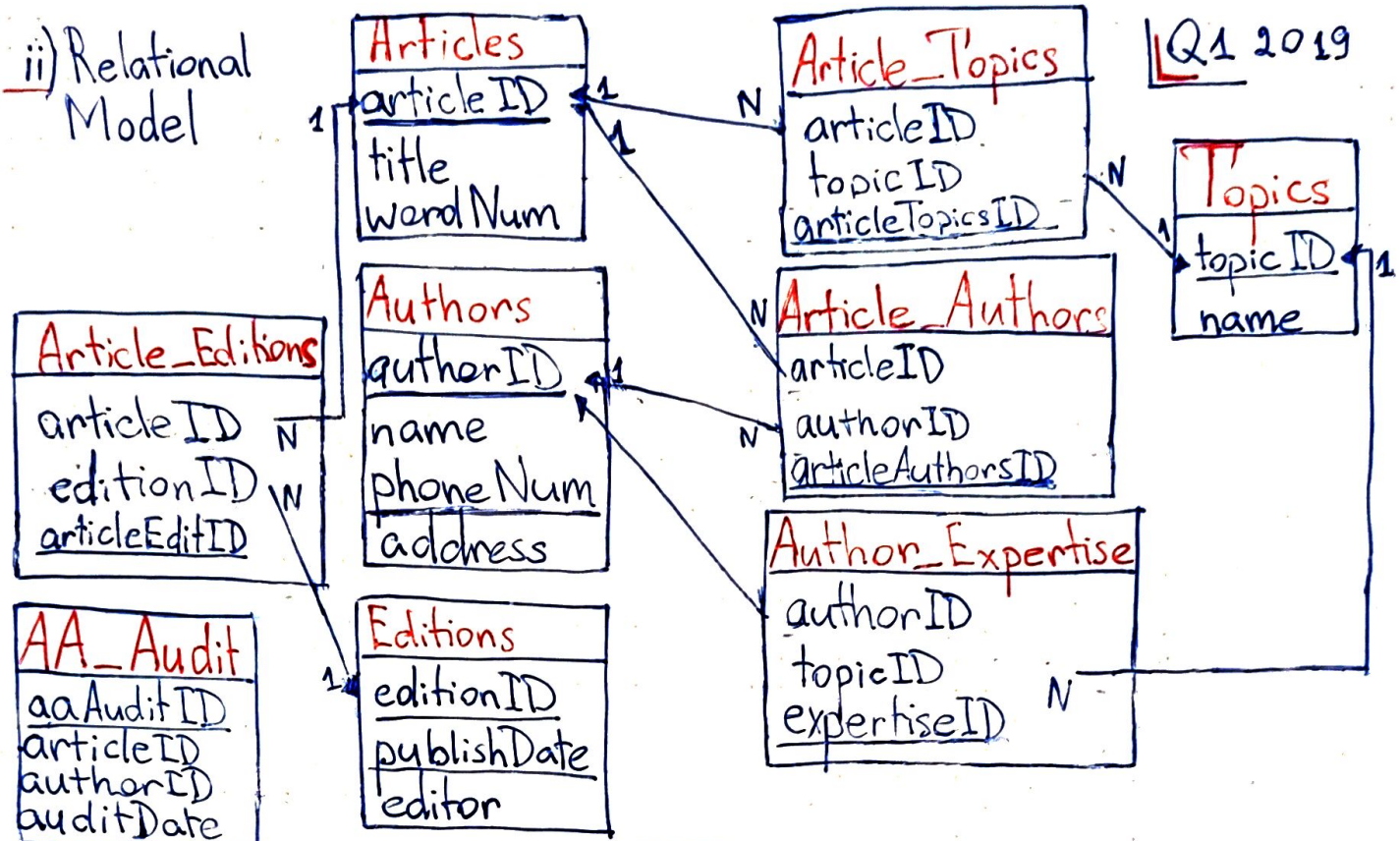
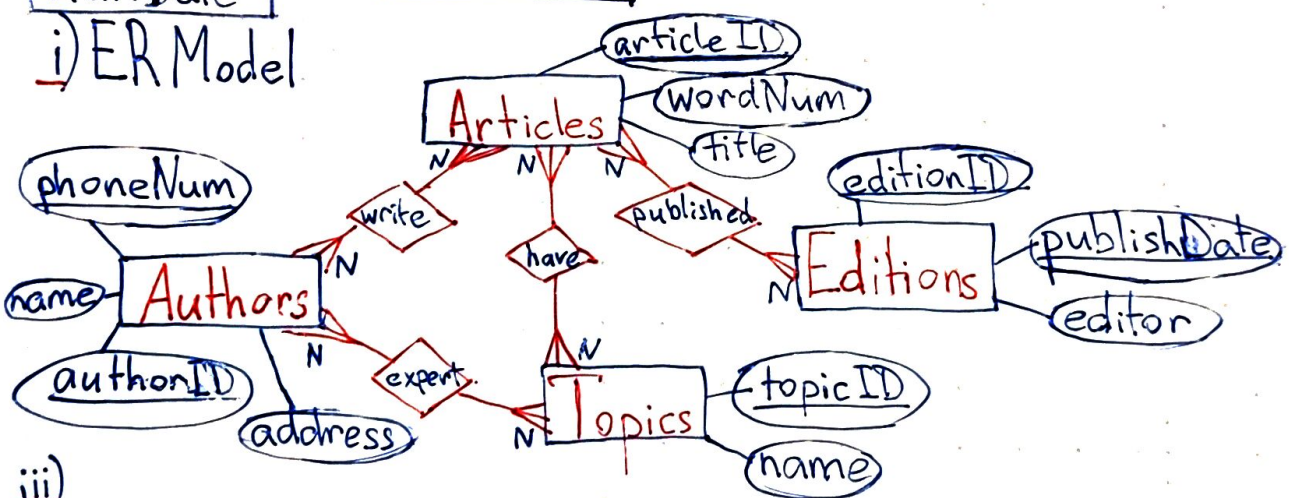
Q1 2019

ii) Relational Model



i) ER Model



iii)

a)

- INSERT INTO `Articles` (`articleID`, `title`, `wordNum`) VALUES (`12399`, `Boris Johnson election`, `1500`);
- INSERT INTO `Topics` (`topicID`, `name`) VALUES (`921`, `Boris Johnson loses election again`);
- INSERT INTO `Article_Authors` (`articleID`, `authorID`) VALUES (`12399`, `24`);

↖ let's assume that Johns' ID = 24. →

- INSERT INTO `Article_Topics`(`articleID`,`topicID`)
  VALUES(`12399`,`921`);

iii) b) SELECT DISTINCT Topics.name FROM Articles
INNER JOIN Article_Editions ON
Article_Editions.articleID = Articles.articleID
INNER JOIN Article_Topics ON Article_Topics.articleID=
= Articles.articleID
INNER JOIN Editions ON Article_Editions.editionID=
= Editions.editionID
INNER JOIN Topics ON Article_Topics.topicID =
= Topics.topicID
WHERE Articles.wordNum > 100
AND Editions.publishDate = `2091-11-11`;

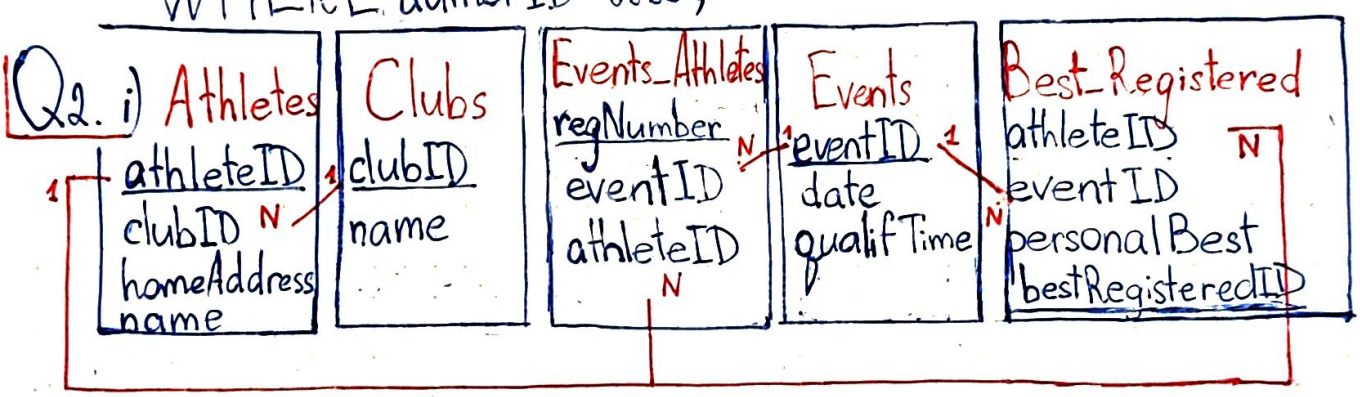
iii) c) CREATE VIEW NewspaperDB.Crime_Experts AS
SELECT DISTINCT Authors.name, Articles.title
FROM Articles
INNER JOIN Article_Authors ON Article_Authors.articleID=
= Articles.articleID
INNER JOIN Authors ON Authors.authorID=
= Article_Authors.authorID
INNER JOIN Authors_Expertise ON Authors_Expertise.
authorID = Article_Authors.authorID

→

INNER JOIN Topics ON Authors_Expertise.topicID =
=Topics.topicID
INNER JOIN Article_Topics ON Article_Topics.topicID =
= Authors_Expertise.topicID
WHERE Topics.name = 'crime';

iii) d) UPDATE Authors SET name = 'Donal Daly'
WHERE authorID = 8888;

Q2. i)



ii) • 1 NF: each table cell contains only a singe value & each record is unique.
All tables listed above are the good examples of 1NF.
• 2 NF: all from 1NF & each non-key attribute is dependent on PK.
Clubs is a 2NF table.
• 3NF: all from 2NF & all non-key attributes are independent of each other.
Clubs is a 3NF table.

iii) a) SELECT DISTINCT Events.eventID FROM Events_Athletes
INNER JOIN Events_Athletes ON Events_Athletes.eventID =
Events.eventID WHERE Events.date > NOW()

b) SELECT DISTINCT Athletes.name, Clubs.clubID,
Events.eventID FROM Athletes
INNER JOIN Clubs ON Athletes.clubID = Clubs.clubID

→

```
INNER JOIN Best_Registered ON Athletes.athleteID=
=Best_Registered.athleteID
INNER JOIN Events ON Best_Registered.eventID=
=Events.eventID;
```

d)

```
ALTER TABLE Athletes
  ADD COLUMN QualStatus VARCHAR(15);
  UPDATE Athletes
  SET QualStatus=CASE WHEN Best_Registered.personalBest<=(
      SELECT qualifTime FROM Events
      WHERE Events.eventID = Events_Athletes.eventID
      )THEN 'QUALIFIED'
      ELSE 'NOT QUALIFIED'
  END;
```

Q3. i) Integrity in DB ensures that stored data is mantained in a valid & consistent state at all times. It refers to the accuracy, completeness and reliability of the data in the DB.
Integrity types: entity integrity, referential integrity, domain integrity.

ii) Entity integrity can be specified by defining a primary key for each row.

iii) Triggers in SQL execute predefined actions in response to certain events. For instance, this trigger will ensure that when a new article is inserted in Article_Authors, there will be created audit for it in AA_Audit.

```
CREATE TRIGGER trg_aaaudit
ON Article_Authors AFTER INSERT AS
INSERT INTO AA_Audit(articleID, authorID, auditdate)
  SELECT (articleID, authorID, NOW())
FROM INSERTED
```

→

iv) a) • GRANT SELECT, INSERT, UPDATE ON MODULE, LECTURER TO SUBJECT_AREAS;
• GRANT SELECT, UPDATE ON MODULE TO UNIVERSITY_STAFF;
• GRANT SELECT ON STAFF, LECTURER TO UNIVERSITY_STAFF;

b) • REVOKE INSERT, UPDATE ON MODULE, LECTURER FROM SUBJECT_AREAS;

• "STAFF" is a relation, it does not have privileges. STAFF table can still be accessed by DEPARTMENT with all 3 privileges and by UNIVERSITY_STAFF with read privilege.

v) The data about customers must be protected. The GDPR law applies to the EU and EEA — any organisation in these areas is obliged to maintain certain data protection, security and confidentiality principles. Furthermore, the collected data may contain sensitive personal information, which is a subject to additional protection.
As a DB administrator, you must receive users' consent to collect, store, use their data, mentioning to what extent it might be processed. You must provide your customers these rights under GDPR: • access; • to be informed; • rectification; • erasure; • portability; • automated decision making consent; • to object to processing.
Moreover, you have to ensure protection, security and confidentiality of customers' data.

Q4. i) Big data exceeds the processing capacities of conventional database systems: their approach is • solely relational, • makes it hard to scale data: only scale up or out (not both), • hard to partition data.

scale up → • expensive;
• vertical scaling:
• bigger machine
• more processors, storage, memory

out (not both) →
• horizontal scaling:
• data partitioning based: divide the DB across multiple machines
• cheaper
• more resilient
! BUT relational DBs are not designed to
• run on cluster. →

Summing up, SQL DBs are dependent on pre-filter, assume single disk farm, hard to partition, based on 1970's storage assumptions.

ii) As described in Q4. i), DBMS have two approaches to scaling, but unlike NoSQL, they can not be combined.

- NoSQL can store complex objects, which can not be represented in a relational way—that is what "Not only SQL" stands for.
- NoSQL is designed for easy horizontal scaling, which includes data distribution such as replication and sharding.
- NoSQL are non-relational databases; they are schema-less and work well with unstructured data; use non-tabular models like document-oriented, key-value or graph-based. Languages: JSON, XML, YAML.

All of that makes NoSQL DBs a perfect choice for cloud infrastructure.

iii) NoSQL should be preferred over relational DB when there is a need to store & work with the large amounts of semi-structured or non-structured data; when you want to scale data horizontally, for example across multiple geographical locations; when you have a lot of data and/or many different data types; when you are a cloud provider & you want to deploy pay-as-you-go model for your clients; when you are not conserned with a 100% data integrity, but need flexibility in schema design/no schema.

iv) Polyglot persistence is a concept that refers to the use of multiple data storage approaches & technologies within a single system, in order to meet dif. data storage needs. It exists because no single DB storage system can provide the best solutions for all data types.

In NoSQL, polyglot persistence can be implemented by identifying your needs & then choosing available approaches to meet these needs: for example, one company may need to store user profiles (document type DB), analysis data (graph type) and cache data (key-value type). All these requirements can be leveraged using a combination of NoSQL databases, so that each of them will handle one or several of these requirements. It will result in improved performance, better scalability and maintainability.