Question 1 – Dimensional Model **[25 marks]**

a. What is the *grain* in a dimensional model? [3 marks]

b. Explain the difference between storing the full timestamp in a dimensional model versus introducing a time dimension. What are the advantages and disadvantages of the two solutions?

[4 marks]

c. The Train Ltd. company requires the designing of a data warehouse to record the number of passengers, sales on their trains. It also needs to monitor train punctuality. The starting database is composed of the following tables:

CUSTOMER (Cust_Code, Name, Address, Phone, BDay, Gender, CountyCode [FK])
TIMETABLE(Route_ID,Start_station_ID [FK], End_station_ID [FK], Dep_Time, Arr_Time, route_type)
TRAIN(Train_ID,TripDate,Dep_Time,Arr_Time,RouteID [FK])
TICKETS(Ticked_ID,Train_ID [FK], Cust_Code [FK], Num_tickets_full, Num_tickets_reduced,Date, Start_Station_ID [FK], End_Station_ID [FK], TicketClass [FK])
TICKET_PRICE(TicketClass,Date,Full_Price,Reduced_Price)
STATIONS(Station_ID,Station_Name,Latitude,Longitude,City, CountyCode [FK])
COUNTY(CountyCode,Population,Province)

The DB contains information about customers and trains. There is a timetable containing information about each train connection offered (for example the route_id=5 is "Dublin-Cork leaving at 8:40 arriving at 11:20"). The field route_type is either "daily", "weekdays" or "weekends".
The table train stores information about a single actual train trip, when the train left and arrived to the final destination and the route of the train (for instance, train with route id 5 (=Dublin Cork 8:40-11:20), left at 8:50 and arrived at 11:40 on the 6th June 2015.
The DB contains information about stations and their geographical location. Finally, the DB contains information about tickets sold for every train. Customer buys tickets by specifying the number of "full price" tickets and the number of "reduced price" tickets, and the departure and destination station. The price of each ticket is based on the date of the purchase and on the class of the ticket. The class of each ticket is computed by the system based on the departure and destination station (for instance, a Dublin-Waterford ticket is class "E"). The prices of tickets depend on the date the ticket is issued, the class of tickets and if it is a reduced or full price ticket.

Produce one or more star schema for the above ER diagram. The diagram(s) should support the following queries and reports:

(i) A weekly report showing the total revenue for each train route
(ii) A weekly report showing the distribution of customers by county and train route

(iii) Show the list of trains with a number of passengers above 500
(iv) Show the average number of tickets sold for each train in each county
(v) Understand the demography of the customer base (gender / age)
(vi) Show the average delay of each train route
(vii) Show the percentage of trains that arrived at the final destination punctually(=arrival time <= expected arrival time as stored in the Timetable Table).
(viii) Show the name of the bestselling route each week
(ix) Show, for every month, the number of trains that arrived more than 1 hour late during the weekdays in all the counties with more than 100000 persons.

Justify all your design choices. If a field in the fact or dimension table is not in the ER diagram, explain how to derive it, where it should be derived and why.
[13 marks]

d. Using your dimensional model, write the SQL query at (viii).
[5 marks]

1.

a. In a dimensional model, the grain represents the level of detail at which data is stored in the database. The grain of a dimensional model is often determined by the business processes that the model is intended to support. For example, a sales dimensional model might have a grain of “sales transaction”, meaning that each row in the fact tables represents a single sales transaction, with the associated dimensions providing context about the transaction such as the customer, product and location. The choice of grain will depend on the specific needs of the business and the types of questions that the model is intended to support.
b. In a dimensional model, storing the full timestamp in a fact tables means that the timestamp is stored as a column in the table along with the other

measure columns. This approach can be useful if the timestamp is an important aspect of the data itself and needs to be included in queries and analyses.

On the other hand, introducing a time dimension means creating a separate table that represents the different elements of time (such as year, month, day, hour) and linking this table to the fact table through a foreign key. This approach can be useful if the data needs to be analysed or aggregated by time period, or if the full timestamp is not important for the analyses being performed.

There are a few advantages and disadvantages to consider when deciding between storing the full timestamp in a fact table or introducing a time dimension.

Advantages of storing the full timestamp in the fact table

- Simplicity: there is no need to create and maintain an additional table.
- Performance: Queries may be faster, as there is no need to join another table to access the timestamp.

Disadvantages of storing the full timestamp in the fact table:

- Redundancy: The same timestamp value may be repeated many times in the table, taking up additional space and potentially causing data inconsistencies.

- Flexibility: It can be more difficult to perform time-based aggregations or analyses if the timestamp is stored in the fact table.

Advantages of introducing a time dimension:

- Flexibility: It is easier to perform time-based aggregations and analyses using a time dimension.
- Data integrity: A time dimension can help to ensure that consistent and valid time values are used throughout the model.

Disadvantages of introducing a time dimension:

- Complexity: An additional table needs to be created and maintained.
- Performance: Queries may be slower, as they will need to join to the time dimension table to access the timestamp.

c.

2.

a.

b.

c.

d. The entity-relationship model and json-like model are both approaches for organizing and storing data in a database. Each has its own advantages and disadvantages, and the appropriate choice will depend on the specific requirements of the application.

   One advantage of the ER model is its ability to clearly represent the relationships between different entities in a database. The ER model uses a

graphical notation to depict these relationships, making it easier for developers and analysts to understand and work with the data. This can be particularly useful in complex databases with many interdependent entities.

Another advantage of the ER model is its ability to enforce data integrity. Because the ER model defines the relationships between entities, it can be used to ensure that data is entered and stored correctly in the database. For example, if an ER model defines a “customer” entity that has a relationship with an “order” entity, the database can be configured to ensure that an order cannot be created without a corresponding customer.

On the other hand, one disadvantage of the ER model is that it can be more difficult to implement than a JSON-like model. The ER model requires the creation of a detailed schema, which can be time-consuming and may require a deeper understanding of database design. In addition, the ER model may not be as flexible as a JSON-like model, which can be more easily modified to accommodate changing requirements.

A JSON-like model, on the other hand, is generally easier to implement and can be more flexible than an ER model. JSON is a simple, human-readable data format that is often used for storing and exchanging data. A JSON-like model can be used to represent complex data structures, such as nested arrays and

objects, in a way that is easy to understand and work with.

One disadvantage of a JSON-like model is that it may not be as efficient for storing and querying large amounts of data as other database models. JSON data is typically stored as a single entity in the database, which can make it more difficult to index and query efficiently. In addition, the lack of a defined schema in a JSON-like model can make it more difficult to enforce data integrity and ensure that data is entered and stored correctly.

In summary, the ER model is a powerful and flexible approach for organising and storing data in a database, but it may be more difficult to implement and may not be as flexible as a JSON-like mode. On the other hand, a JSON-like model is generally easier to implement and can be more flexible but may not be as efficient for storing and querying large amounts of data. The appropriate choice will depend on the specific requirements of the application.

3.
4.

a. There are several potential problems that can arise when implementing indexes using binary trees:
    i. Balancing: In order to maintain good performance, it is important to keep the binary tree balanced, meaning that the height of the tree should be kept as small as possible. This can be difficult to achieve, especially in the case of a write-mostly application, where new

data is being added to the tree frequently. Unbalanced trees can lead to poor performance, as search and insertion operations will take longer.

ii. Space overhead: Binary trees require additional space to store pointers to child nodes which can increase the amount of memory needed for the index. This can be especially problematic for large datasets, where the overhead may become significant.

iii. Insertion performance: Insertion into a binary tree can be slow, especially if the tree becomes unbalanced this can be a problem for write-mostly applications, where new data is being added to the tree frequently.

On the other hand, read mostly applications, where the majority of operations are reads rather than writes, may not be as affected by these issues. However, they may still experience slower performance if the tree becomes unbalanced or if the space overhead becomes too large.

Examples of applications that might be affected by these problems includes databases and search engines, which often require fast iteration and search performance, as well as real-time data processing that need to handle a large volume of writes.

b.

i. Slowly changing dimensions(SCD): this strategy involves creating a new record in the

dimension tables each time the dimension attribute changes. For example, if a customer's address changes, a new record would be created in the customer dimension table with the updated address, while the old record would be preserved with the original address. This approach is useful for preserving the history of changes to dimension attributes, but it can lead to large dimension tables over time.

ii. Rapidly changing dimensions (RCD): this strategy involves overwriting the existing record in the dimension table with the updated attribute value, rather than creating a new record. This approach can be faster and more efficient than SCD, but it does not preserve the history of changes to dimensions attributes.

iii. Type 2 SCD: This strategy is a hybrid approach that combines elements of both SCD and RCD. It involves creating a new record in the dimension table each time the dimension attribute changes, but also adding a column to the table that tracks the effective data range for each record. This allows the data warehouse to track changes to dimension attributes over time, while still maintain a relatively small dimension table.

For example, consider using a customer dimension table that tracks the name and address of each customer. Using the SCD strategy, a new record would be created each time a customer's address changes, resulting in multiple records for each

customer. Whereas using the RCD strategy, the existing record would be updated with the new address, but the original address would be lost, and a column would be added to the table to track the effective data range for each record. This would allow the data warehouse to track changes to the customer's address over time, while still maintaining small dimension table.

c.

5.

a.

ACID is an acronym that stands for four properties of a database transaction: Atomicity, Consistency, Isolation and Durability. These four properties ensure that database transactions are reliable, even in the face of errors, power failures, and other unexpected events.

1. Atomicity: This property ensures that a transaction is either completed in its entirety or not completed at all. In other words, if a transaction involves multiple updates to the database, either all of the updates are applied or none of them are applied. This is known as "all-or-nothing" semantics
2. Consistency: This property ensures that a transaction leaves the database in a consistent state, meaning that the data adheres to all defined rules and constraints. If a transaction violates a

constrain or rule, its in not allowed to complete.

3. Isolation: this property ensure that concurrent transactions do not interfere with each other. Each transaction is executed as if it were the only transaction taking places, even if other transaction is occurring at the same times. This prevents data from being corrupted by concurrent updates.
4. Durability: This property ensures that once a transaction has been committed, it will not be lost. The changes made by the transaction are permanently recorded in the database, even if the database crashes or shuts down.

ii. Base is an acronym that stands for four properties of a NOSQL database: Basically Available, Soft state, Eventual consistency. These properties are a set of design properties for NOSQL databases that prioritize flexibility and scalability over ACID properties of traditional relational databases.

1. Basically Available: This property refers to the ability of the database to continue functioning, even if some parts of the system are unavailable or experiencing errors. This is in contrast to the “all-or-nothing” semantics of the atomicity property in ACID databases, which requires that all parts of the system be available for a transaction to complete.

2. Soft state: This property refers to the fact that the state of a NOSQL database can change over time , even without direct input from users or application. For example, a distributed database may automatically replicate data between nodes, leading to changes in the overall state of the system.
3. Eventual consistency: This property refers to the fact that, over time, all copies of the data in a NOSQL database will eventually converge to the same state. In other words, even if different copies of the data are temporarily inconsistent , they will eventually become consistent as changes are propagated throughout the system. This is in contrast to the consistency property in ACID databases which requires that all copies of the data be consistent at all times.

Together these properties allow NOSQL databases to be highly available, scalable, and flexible, but at the cost of some of the strong consistency guarantees provided by ACID databases.

iii. One key difference between ACID and BASE is the way they handle consistency. ACID databases prioritize strong consistency, meaning that all copies of the data must be consistent at all times. BASE databases, on the other hand, prioritize availability and may allow for

temporary inconsistencies between copies of the data.

In general, ACID databases are more suitable for applications that require strong consistency and transactional guarantees, such as financial system or e-commerce websites. BASE databases are more suitable for applications that require high availability and scalability, such as social media platforms or real-time data processing systems.

It's worth noting that there is a trade-off between consistency and availability, and the appropriate choice will depend on the specific needs of the application.

b. In MongoDB, a replica set is a group of MongoDB instances that maintain the same data set. Replica sets provide redundancy and high availability and are a basis for all production deployments.

A replica set consists of primary member and one or more secondary members. The primary member is responsible for all write operations and can accept read operations. The secondary members are copies of the primary member and are used for reading operations and to provide failover support. In the event that the primary member goes down one of the secondary members is automatically elected as the new primary. This process is known as failover.

An arbiter is a special type of member in a replica set that does not have a copy of the data set. Its

main purpose is to participate in elections for primary members.

Sharding is a method for distributing data across multiple machines. It is used to horizontally scale a MongoDB deployment by partitioning the data across multiple shards. Each shard is a replica set, and each shard is responsible for a portion of the data.

A sharding key is a field in the documents in a sharded collection that determines which shard the document belongs to. The sharding key is used to evenly distribute the documents across the shards in the cluster.

The main difference between replica set and sharding is that replica sets provide redundancy and high availability within a single deployment, while sharding allows MongoDB to scale horizontally by distributing the data across multiple machines.

c.

i. In MongoDB, a hidden member is a secondary member that is not included in the replica set configuration and is not eligible to become primary. It is used as a backup in case all other members of the replica set become unavailable. They do not receive read or write operations and do not participate in elections for primary. They are used as a last resort in case of complete failure of the replica set
ii. A secondary member with priority 0 is a member of a replica set that cannot become

primary, but can still participate in elections for primary. This configuration is useful when you want to exclude a member from becoming primary, but still want it to participate in elections and potentially contribute to the majority of votes needed to determine the new primary. It can receive read operation and participate in elections, but they cannot accept write operation or become primary. They're useful for scenarios where you want to exclude a member from becoming primary, but still want them to contribute to the replica set's availability and read performance.

iii. A delayed member is a secondary member that is configured to have a delay in replication. This means the delayed member receives a copy of the data from the primary member with a specified time lag. Delayed members are useful when you want to have a copy of the data that is not current, but still want to be able to recover from data loss in the event of a disaster. They're useful for scenarios where you want to keep a copy of the data for compliance or auditing purposes, but do not need it to be current