| | |
|---|---|
| SIES Graduate School of Technology (Autonomous) NAAC A+ (Affiliated to University of Mumbai) | ***End Semester Examination (R-24) SH 2025 Answer Key with marking scheme*** |
| Branch: CSE (IOT & CSIBCT) | Course: Database Management Systems |
| Year/ Semester: SE III | Course code: (CSEC304) |
| Time: 03 hours | Marks: 80 |

| | | Marks |
|---|---|---|
| Q. 1 | Attempt any FOUR. (All questions carry equal marks) | |
| A. | Describe the Three-level of abstraction with a proper diagram.<br>3 Level of Abstraction<br><br>Draw Diagram showing levels | 05 |
| B. | Describe in detail the limitations of conventional databases.<br>Conventional databases are limited by their :-<br>• high costs,<br>• complexity,<br>• poor scalability,<br>• inability to handle unstructured data,<br>• fixed schema, and<br>• potential for performance issues and data redundancy.<br>These limitations stem from their reliance on :-<br>• structured data formats and<br>• their architecture, which can struggle with the large volumes and diverse types of data found in modern applications | 6 |
| C. | Discuss group by and order by clause with example<br>Purpose ofOrder By Clause in Sql .Explain , Write Sql Query and diaplay relation<br>Purpose of Group By Clause in Sql .Explain , Write Sql Query and diaplay relatio | 05 |
| D. | Explain the necessary condition used by time stamp ordering protocol to execute for a read/write operation | 05 |
| E. | Explain different entity types and attribue types in the Entity and Relationship Diagram<br>ntities in an ER diagram are categorized as **Strong**, **Weak**, or **Associative**. A **strong entity** has its own independent existence and is represented by a single rectangle. A **weak entity** depends on a strong entity for its existence and is shown with a double rectangle. An **associative entity**, represented by a diamond inside a rectangle, resolves many-to-many relationships between two or more entities.<br>Here's a breakdown of each type with an example:<br>• Strong Entity | 05 |

| | | |
|---|---|---|
| | ○ **Definition:** A strong entity is a real-world object, concept, or event that has its own independent existence and can be uniquely identified by its own attributes (primary key).<br>○ **Example:** In a university system, a **Student** is a strong entity because a student's existence and identity are independent of any other entity like a course or a professor. Each student can be uniquely identified by a student ID.<br>○ **Representation:** A single rectangle.<br>• <br>Weak Entity<br>• **Definition:** A weak entity is an entity that cannot exist on its own; its existence is dependent on another entity, called the owner or strong entity. It relies on the owner entity's identifier and does not have a unique identifier of its own.<br>• **Example:** In a banking system, a **Transaction** might be a weak entity. A transaction for a specific deposit or withdrawal cannot exist without being associated with an **Account**. The transaction might not have a unique ID on its own but depends on the account it belongs to.<br>• **Representation:** A double rectangle<br><br>**Key-point**: In a Database Management System (DBMS), an attribute is a property or characteristic of an entity that is used to describe an entity. Types of Attributes<br>There are different types of attributes as discussed below-<br>• Simple Attribute<br>• Composite Attribute<br>• Single-Valued Attribute<br>• Multi-Valued Attribute<br>• Derived Attribute | |
| F. | Describe functional,partiall and transitive dependency<br>What is Functional Dependency?<br>Functional dependency states the relationship between two sets of attributes where a value of a set of attributes is dependent on the other set of attributes.<br>It is a relationship that typically exists between two attributes such that with the help of one attribute we can get the values of another attribute. The attribute that is used for finding the values of other attributes is called the primary key attribute.<br>ypes of Dependencies<br>Partial Dependency<br>Full Dependency<br>Transitive Dependency<br>Partial Dependency | 05 |

If the value of a non-primary attribute can be defined using part of the primary key then it is called a partial dependency. Partial dependency occurs when primary key is formed using more than one attribute.This type of key also called as composite key.
In below given example, the primary key is formed using roll_no + sub_id which can also be called as composite key.When composite key is present and one of the non-primary attribute can is dependent on part of the primary key instead of whole primary key then it is called as Partial Dependency.
Example
Let's take an example, we have a table where we have columns of student roll number, subject ID, sub name, and marks obtained.
Table
roll_no, sub_id, sub_name, sub_mark
1, 121, Science, 80
1, 131, Math, 65
2, 131, Math, 95
2, 141, English, 75
Here primary key will be roll_no+ sub_id because multiple roll_no can have the same sub_id and the same roll_no can have multiple sub_id.In the given example, roll_no 1 has two sub_id i.e. 121 and 131 where as sub_id 131 has two roll_no 1 and 2.So here primary key will be roll_no + sub_id.
But we do have another column of sub_name and the value of sub_name can be easily obtained by only sub_id which is part of the primary key.For example, sub_id = 131 will have the sub_name = 'math' here we required only partial primary key i.e. sub_id.
This type of functional Dependency is known as Partial Dependency.

Transitive Dependency
If the value of a non-primary attribute can be defined using another non-primary attribute then it is called a transitive dependency.
When any attribute does not require primary key and can easily get value using another non-primary attribute then it is called as Transitive Dependency.
Example
Let's take an example, we have a table where we have columns of student roll number, name, city where student live, and zip-code of city .
Table
roll_no, name, city, zip-code
1, abc, pune, 411044
2, jkl, mumbai, 400001
3, uvw, pune, 411044
4, xyz, delhi, 110001
Here the primary key is roll_no, but we can identify the city using zip-code where both city and zip-code are non-primary attributes.
So here roll_no → city and city→zip-code eventually resulting into

| | | |
|---|---|---|
| | roll_no →zip-code. so we can find a non-primary attribute using another non-primary attribute.For example, roll-no = 1 has city=pune and city=pune will have zip-code=411044.So wherever city is pune , zip-code will be 411044<br>This type of functional Dependency is known as Transitive Dependency. | |
| | | |
| Q.2 | Attempt any FOUR. (All questions carry equal marks) | **40** |
| A. | Consider the following schema for Institute Library<br>Student(Rollno,Name,Father_Name,Branch)<br>Book(ISBN,Title,Author,Publisher)<br>Issue(Rollno,ISBN,Date_Of_Issue)<br>Write Relational Algebra expression for the following.<br>(ii)List Roll Number and Name of all students of the branch CSE<br>(iii)Find the name of students who have issued a book published by 'ABC' Publisher<br>(iii)List title of all books and their author issues by student 'Prashant'<br>(iv)List title of all books issued on or before 1st Jan 2014<br>(v)Rename relation Book to Book_Info | 10 |
| B. | Discuss Timestamp-based protocol in detail<br>**Timestamp Ordering Protocol Rules**<br>**Timestamps** follow some rules to perform read or write operations. The rules are also known as Thomas rules**.**<br>**Rule No. 01 is utilized when a transaction requires a Read (A) operation.**<br>If WTS(A) > TS (Ti), then Ti Rollback<br>Else (otherwise) execute R(A) operation and **SET** RTS (A) = MAX {RTS(A), TS(Ti)}<br>**Rules No.2** rules are used when a transaction needs to perform **WRITE (A)**<br>If RTS(A) > TS (Ti), then Ti Rollback.<br>If WTS(A) > TS (Ti), then Ti Rollback.<br>Else (otherwise) execute the W(A) operation and **SET** WTS (A) = TS(Ti).<br>Where **"A"** is some data | 10 |
| C. | Explain Conflict Searlizability with example.<br>Conflict Serializability ensures that a concurrent schedule produces the same result as some serial execution by reordering non-conflicting operations. It maintains data consistency and is stricter than View Serializability, which allows more flexibility but still preserves correctness.<br>Non-conflicting operations: Two operations are considered non-conflicting if they operate on separate data items, or if they involve the | 10 |

| | | |
|---|---|---|
| | same data item but both are read operations.<br>Conflicting Operations<br>Two operations are said to be conflicting if all conditions are satisfied:<br>They belong to different transactions<br>They operate on the same data item<br>Atleast one of them is a write operation<br><br>1. $I_i$ = read(Q) — $I_j$ = read(Q) — No Conflict<br>2. $I_i$ = read(Q) — $I_j$ = write(Q) — Conflict<br>3. $I_i$ = write(Q) — $I_j$ = read(Q) — Conflict<br>4. $I_i$ = write(Q) — $I_j$ = write(Q) — Conflict | |
| D. | Notown Records has decided to store information about musicians who perform on its albums (as well as other company data) in a database.<br>(i)Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians often share the same address, and no address has more than one phone.<br>(ii)Each instrument used in songs recorded at Notown has a unique identification number, a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).<br>(iii)Each album recorded on the Notown label has a unique identification number, a title, a copyright date, a format (e.g., CD or MC), and an album identifier.<br>(iv)Each song recorded at Notown has a title and an author.<br>(v)Each musician may play several instruments, and a given instrument may be played by several musicians.<br>(vi)Each album has a number of songs on it, but no song may appear on more than one album.<br>(vii)Each song is performed by one or more musicians, and a musician may perform a number of songs.<br>(viii)Each album has exactly one musician who acts as its producer. A musician may produce several albums, of course.<br>Identify Entities<br>Relationshipa<br>Attributes<br><br>Also show relationships<br>Show pimary keys for the entities<br>Design an ER diagram for the above database | 10 |
| E. | Consider the following insurance database and write a SQL queries for: | 10 |

| | | |
|---|---|---|
| | Person(driver_id,name,address)<br>Car(licence,model,year)<br>accident(report_no,date,location)<br>owns(driver_id,licence)<br>participated(driver_id, car, report_no,damage_amount)<br>(i) Find the total number of people who owned cars that were involved in accidents in 1989<br>(ii) Find no of accidents in which cars belonging to "Scott" were involved.<br>(iii) Add a new record to the database. Assume any values for the attributes.<br>(iv) Delete Mazda belonging to "John" .<br>(v) Update damage_amount for the car with licence no "AABB2000" in the accident with report_no "AR2197" to 3500. | |
| F. | Consider the following relational schemes for a library database:<br>Book (Title, Author, Catalog_no, Publisher, Year, Price)<br>Collection (Title, Author, Catalog_no)<br>With the following functional dependencies:<br>I. Title,Author ->Catalog_no<br>II. Catalog_no -> Title, Author, Publisher, Year<br>III. Publisher Title Year -> Price<br><br>Assume {Author, Title} is the key for both schemes. Check in which Normal form both relations are. Justify your answer<br><br>**Book** (Title, Author, Catalog_no, Publisher, Year, Price, bookCoverType, contractDate)<br>**Collection** (Title, Author, Catalog_no). Assume {Author, Title} is the key for both relations. Additional functional dependencies are :-<br>1. Title,Author --> Catalog_no<br>2. Catalog_no --> Publisher, Year, bookCoverType<br>3. Publisher, bookCoverType --> Price<br>4. Author --> contractDate<br>a. Explain what normal form the relation is in.<br>b. Apply normalization until the 3rd NF. State reasons behind each normalization.<br>**Answer:**<br>a. It's in 1 NF, because no multi valued/composite attribute and no nested relations.<br>b. It is not in 2NF as there is partial dependency due to FD IV.<br><br>Therefore, normalizing to 2NF:<br>Collection (Title,Author,Catalog_no)<br>Book (Title, Author, Catalog_no, Publisher, Year, Price, bookCoverType)<br>Author_Info(Author, ContractDate)<br><br>Now normalizing to 3NF as there is still transitive dependency in "Book" | 10 |

| | | |
|---|---|---|
| | table due to Fd II and III.<br>Collection (Title,Author,Catalog_no)<br>Author_Info(Author, ContractDate)<br>Book (Title, Author, Catalog_no)<br>Catalog (Catalog_no, Publisher, year, bookcovertype)<br>Price_info(Publisher, bookcovertype, price) | |
| | | |
| Q.3 | Attempt any FOUR | **20** |
| A. | Differentiate between Deferred Vs. Immediate Database Modification. | 05 |
| B. | Explain all properties of the transaction with example. (ACID Properties)<br>xplain ACID properties related to Transaction management.<br>ACID properties—**Atomicity, Consistency, Isolation, and Durability**—are fundamental guarantees for database transactions, ensuring data integrity, accuracy, and reliability even during failures. Atomicity ensures a transaction is an all-or-nothing operation, Consistency maintains a valid database state, Isolation prevents concurrent transactions from interfering, and Durability makes committed changes permanent.<br><br>Here's a breakdown of each property:<br><br>**Atomicity (All or Nothing)**<br><br>What it means: A transaction is treated as a single, indivisible unit. Either all operations within the transaction are completed successfully, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database reverts to its state before the transaction began.Example: In a banking transaction, transferring money from account A to account B involves two steps: debiting account A and crediting account B. Atomicity ensures that both these steps complete, or neither does, preventing situations where money is debited but not credited.<br>**Consistency**<br><br>What it means: A transaction must bring the database from one valid state to another valid state, adhering to all defined rules, such as constraints, triggers, and data integrity rules.Example: If a database rule prevents negative account balances, a transaction attempting to withdraw more money than available will be canceled to maintain consistency and prevent invalid data.<br>**Isolation**<br><br>What it means: Each transaction executes as if it were the only one running on the system, ensuring that the partial results of concurrent transactions are not visible to others. This prevents interference between | 05 |

transactions running simultaneously.Example: If transaction T1 is updating a customer's balance, transaction T2 should not be able to see the intermediate, uncommitted state of the balance before T1 has completed. Different isolation levels provide varying degrees of protection against issues like dirty reads (reading uncommitted data) and non-repeatable reads (reading different results for the same query multiple times within a single transaction).

**Durability**

What it means: Once a transaction is successfully committed, its changes are permanent and will survive any subsequent system failures, such as power outages or crashes.Example: After a transaction is committed in a banking system, even if the system crashes immediately afterward, the changes to the database will be preserved, and the correct data will be available when the system restarts. This is typically achieved through mechanisms like write-ahead logging.

C. Define serializability, Conflict serializability and view serializability — 05

D. Explain DCL and TCL commands in SQL — 05

**. DCL - Data Control Language**

DCL (Data Control Language) includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system. These commands are used to control access to data in the database by granting or revoking permissions.

| Command | Description | Syntax |
|---|---|---|
| GRANT | Assigns new privileges to a user account, allowing access to specific database objects, actions or functions. | GRANT privilege_type [(column_list)] ON [object_type] object_name TO user [WITH GRANT OPTION]; |
| REVOKE | Removes previously granted privileges from a user account, taking away their access to certain database objects or actions. | REVOKE [GRANT OPTION FOR] privilege_type [(column_list)] ON [object_type] object_name FROM user [CASCADE]; |

**Example:**

*GRANT SELECT, UPDATE ON employees TO user_name;*

This command grants the user user_name the permissions to select and update records in the employees table.

**5. TCL - Transaction Control Language**

Transactions group a set of tasks into a single execution unit. Each

transaction begins with a specific task and ends when all the tasks in the group are successfully completed. If any of the tasks fail, transaction fails. Therefore, a transaction has only two results: success or failure.

| Command | Description | Syntax |
|---|---|---|
| BEGIN TRANSACTION | Starts a new transaction | BEGIN TRANSACTION [transaction_name]; |
| COMMIT | Saves all changes made during the transaction | COMMIT; |
| ROLLBACK | Undoes all changes made during the transaction | ROLLBACK; |
| SAVEPOINT | Creates a savepoint within the current transaction | SAVEPOINT savepoint_name; |

**Example:**

*BEGIN TRANSACTION;*
*UPDATE employees SET department = 'Marketing' WHERE department = 'Sales';*
*SAVEPOINT before_update;*
*UPDATE employees SET department = 'IT' WHERE department = 'HR';*
*ROLLBACK TO SAVEPOINT before_update;*
*COMMIT;*

In this example, a transaction is started, changes are made and a savepoint is set. If needed, the transaction can be rolled back to the savepoint before being committed.

E. Explain the design steps of cloud database. 05

**Step 1: Define Requirements and Use Cases**

Before designing a cloud-based database, it's crucial to understand the requirements and use cases. This involves identifying the types of data to be stored, the expected workload, query patterns, and performance requirements.

**Example:**

- **Use Case**: An e-commerce platform needs to store customer information, product catalog, order history, and transaction data.
- **Requirements:** The database must support high read and write throughput, handle concurrent user requests, and scale dynamically based on demand.

**Step 2: Choose the Right Cloud Database Service**

Selecting the appropriate cloud database service is essential for meeting the project's requirements and objectives. Options include relational databases (e.g., Amazon RDS, Google Cloud SQL), NoSQL databases (e.g., Amazon DynamoDB, Google Cloud Firestore), and managed database services (e.g., Amazon Aurora, **Google Cloud Spanne**r).

**Example:**

- Relational Database: Choose Amazon RDS for MySQL or Google Cloud SQL for PostgreSQL if the application requires ACID compliance and relational data modeling.
- NoSQL Database: Opt for Amazon DynamoDB or Google Cloud Firestore for flexible schema, high scalability, and low-latency data access.

**Step 3: Design the Database Schema**

Once the cloud database service is chosen, it's time to design the database schema based on the identified requirements and use cases. This involves defining **tables**, indexes, **relationships**, and **access controls.**

**Example:**

- **Customer Table**: Store customer information such as name, email, address, and phone number.
- **Product Table**: Maintain a product catalog with attributes like ID, name, description, price, and inventory.
- **Order Table**: Track order history, including order ID, customer ID, product ID, quantity, and timestamp.

**Step 4: Optimize for Performance and Scalability**

Performance and scalability are critical factors in cloud-based database design. Utilize features like caching, indexing, partitioning, and sharding to optimize performance and handle increasing workload.

**Example:**

- **Caching**: Use in-memory caching solutions like Amazon ElastiCache or Google Cloud Memorystore to improve read performance and reduce database load.
- **Indexing**: Create indexes on frequently queried columns to speed up data retrieval operations.
- **Partitioning/Sharding**: Distribute data across multiple shards or partitions to distribute load and scale horizontally.

**Step 5: Implement Data Security and Compliance**

Ensure data security and compliance with industry standards and regulations such as GDPR, HIPAA, or PCI DSS. Implement encryption, access controls, auditing, and regular security assessments to protect sensitive data.

**Example:**

- **Encryption**: Encrypt data at rest and in transit using encryption mechanisms provided by the cloud database service (e.g., AWS KMS, Google Cloud KMS).
- **Access Controls**: Define IAM roles, policies, and fine-grained access controls to restrict access to sensitive data based on user

| | | |
|---|---|---|
| | roles and permissions.<br>• **Auditing**: Enable database auditing features to track and monitor user activities, data access, and security events.<br>**Step 6: Test and Monitor Performance**<br>Thoroughly test the cloud-based database solution under various conditions to ensure reliability, performance, and scalability. Implement monitoring and alerting mechanisms to detect and address performance issues proactively.<br>**Example:**<br>• **Load Testing:** Simulate heavy user traffic and workload using load testing tools like Apache JMeter or Gatling to evaluate database performance and scalability.<br>• **Monitoring**: Set up monitoring tools such as Amazon CloudWatch or Google Cloud Monitoring to monitor key performance metrics like CPU utilization, memory usage, and query latency. | |
| F. | Discuss mapping of types of relationships from ER to relation with suitable example Write rules<br>Consider an eaxmple ER dia. And show<br>1..Mapping of Entities<br>2.Mapping of Weak entity<br>3Mapping of Binary Relationship with 1:1 cardinality with total participation of an entity<br>4. Binary Relationship with n: 1 cardinality<br>5.Binary relationship with N:M cardinality | 05 |
| ********* **All the Best**********| | |