CMR
INSTITUTE OF
TECHNOLOGY

USN	1	C								

Internal Assessment Test 3 – May/June 2017

Sub:	**Database Management System**							**Code:**	16MCA23
Date:	31-05-17	Duration:	90 mins	Max Marks:	50	**Sem:**	II	**Branch:**	**MCA**

Note: Answer any 5 questions. All questions carry equal marks. Total marks: 50

			Marks	OBE CO	OBE RBT
1.	a.	Define the following terms, with an example with respect to ER model: Entity Set, Relationship type, Cardinality ratio, Participation, Weak entity set.	[10]	CO1	L1
2.	a.	Define data independence? Explain the various types of data independence.	[5]	CO1	L1,L4
	b.	Explain the characteristics of database approach	[5]	CO1	L4
3.	a.	Define functional dependency. Write all its inference rules	[5]	CO2	L1, L2
	b.	Explain with example 1NF,2NF and 3 NF	[5]	CO2	L4
4.	a.	Explain the mechanism to control concurrent access to data item.	[10]	CO3	L4
5.	a.	Explain how Dead lock will be handled.	[10]	CO3	L4
6	a.	Define trigger? Explain with example**.**	[5]	CO4	L1
	b.	Define stored procedure? Explain with example.	[5]	CO4	L1
7	a.	Explain the classification of failures.	[5]	CO3	L1
	b.	Explain the different phases of recovery algorithm.	[5]	CO3	L4
8	a.	Explain ACID properties in detail.	[10]	CO3	L4

CMR
INSTITUTE OF
TECHNOLOGY

Internal Assessment Test 3 – May/June 2017

Sub:	Database Management System	Code:	16MCA23

Marks

1.a. **Define the following terms, with an example with respect to ER model:**
Entity Set, Relationship type, Cardinality ratio, Participation, Weak entity set. [10]

Entity Set:
An entity is an object that exists and is distinguishable from other objects. For instance, John Harris with S.I.N. 890-12-3456 is an entity, as he can be uniquely identified as one particular person in the universe. An entity may be concrete (a person or a book, for example) or abstract (like a holiday or a concept). An entity set is a set of entities of the same type (e.g., all persons having an account at a bank).

Relationship type:
A relationship is any association,linkage, or connection between the entities of interest to the business; it is a two directional, significant association between two entities, or between an entity and itself

Cardinality ratio:

In database design, the cardinality or fundamental principle of one data table with respect to another is a critical aspect. The relationship of one to the other must be precise and exact between each other in order to explain how each table links together.

Participation:
A participation constraint defines the number of times an object in an object class can participate in a connected relationship set. Every connection of a relationship set must have a participation constraint. However, participation constraints do not apply to relationships.

Weak entity set:
An entity set that does not have a primary key is referred to as a weak entity set. The existence of a weak entity set depends on the existence of a strong entity set; it must relate to the strong set via a one-to-many relationship set.

2.a. **Define data independence? Explain the various types of data independence**. [5]

A major objective for three-level architecture is to provide data independence, which means that upper levels are unaffected by changes in lower levels.

There are two kinds of data independence:

Logical data independence

Physical data independence

Logical Data Independence:

Logical data independence indicates that the conceptual schema can be changed without affecting the existing external schemas. The change would be absorbed by the mapping between the external and conceptual levels. Logical data independence also insulates application programs from operations such as combining two records into one or splitting an existing record into two or more records. This would require a. change in the external/conceptual mapping so as to leave the external view unchanged.

Physical Data Independence:

Physical data independence indicates that the physical storage structures or devices could be changed without affecting conceptual schema. The change would be absorbed by the mapping between the conceptual and internal levels. Physic 1data independence is achieved by the presence of the internal level of the database and the n, lPping or transformation from the conceptual level of the database to the internal level. Conceptual level to internal level mapping, therefore provides a means to go from the conceptual view (conceptual records) to the internal view and hence to the stored data in the database (physical records).

b. **Explain the characteristics of database approach** [5]

1. Self-describing nature of a database system.
2. Insulation between programs and data, and data abstraction.
3. Support of multiple views of the data.
4. Sharing of data and multiuser transaction processing.

3.a. **Define functional dependency. Write all its inference rules** [5]

In relational database theory, a functional dependency is a constraint between two sets of attributes in a relation from a database. In other words, functional dependency is a constraint that describes the relationship between attributes in a relation.

Let A, B and C and D be arbitrary subsets of the set of attributes of the giver relation R, and let AB be the union of A and B. Then,⇒→

Reflexivity
If B is subset of A, then A → B

Augmentation
If A → B, then AC → BC

Transitivity:
If A → B and B → C, then A → C.

Projectivity or Decomposition Rule
If A → BC, Then A → B and A → C

```
Proof  :

Step 1 : A → BC (GIVEN)

Step 2 : BC → B (Using Rule 1, since B ⊆ BC)

Step 3 : A → B (Using Rule 3, on step 1 and step 2)
```

Union or Additive Rule :
If A→B, and A→C Then A→BC.

```
Proof :

Step 1 : A → B (GIVEN)

Step 2 : A → C (given)

Step 3 : A → AB (using Rule 2 on step 1, since AA=A)

Step 4 : AB → BC (using rule 2 on step 2)
```

```
    Step 5 : A → BC (using rule 3 on step 3 and step 4)
```

Pseudo Transitive Rule :
If A → B, DB → C, then DA → C

```
     Proof :

     Step 1 : A → B (Given)

     Step 2 : DB → C (Given)

     Step 3 : DA → DB (Rule 2 on step 1)

     Step 4 : DA → C (Rule 3 on step 3 and step 2)
```

b. **Explain with example 1NF,2NF and 3 NF** [5]

1NF:
As per First Normal Form, no two Rows of data must contain repeating group of information i.e each set of column must have a unique value, such that multiple columns cannot be used to fetch the same row. Each table should be organized into rows, and each row should have a primary key that distinguishes it as unique.

The Primary key is usually a single column, but sometimes more than one column can be combined to create a single primary key. For example consider a table which is not in First normal form

Student	Age	Subject
Adam	15	Biology, Maths
Alex	14	Maths
Stuart	17	Maths

The following table is in 1NF.

Student	Age	Subject
Adam	15	Biology
Adam	15	Maths
Alex	14	Maths

Stuart	17	Maths

Second Normal Form (2NF)

As per the Second Normal Form there must not be any partial dependency of any column on primary key. It means that for a table that has concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence. If any column depends only on one part of the concatenated key, then the table fails Second normal form.

Third Normal Form (3NF)

Third Normal form applies that every non-prime attribute of table must be dependent on primary key, or we can say that, there should not be the case that a non-prime attribute is determined by another non-prime attribute. So this transitive functional dependency should be removed from the table and also the table must be in Second normal form.

4.a. **Explain the mechanism to control concurrent access to data item.** [10]

Lock based protocols:

1. Locks
2. Granting of locks
3. The 2-phase locking protocol
4. Implementation of locking

A lock is a mechanism to control concurrent access to data item.

Data items can be locked in 2 modes.

1. Shared
2. Exclusive

Exclusive(x) – Data item can be both read as well as written. X-lock is requested using lock-X instructions.

Shared(x) – Data item can only be read s-lock is requested using lock-s instructions.

Lock requests are made to concurrency control manager

The compatibility relation between the two modes of locking discussed in this section appears in the matrix comp of Figure. An element comp (A, B) of the matrix has the value true if and only if mode A is compatible with mode B.

	S	X
S	True	false
X	false	false

A transaction to unlock a data item immediately after its final access of that data item is not always desirable, since serializability may not be ensured.

```
T1: lock-X(B);
    Read (B);
    B := B – 50;
    Write(B);
```

```
     Unlock(B);
     Lock-X(A);
     Read(A);
     A := A + 50;
     Write(A);
     Unlock(A);
T2:  lock-S(A);
     Read (A);
     Unlock(A);
     Lock-S(B);
     Read(B);
     Unlock(B);
     Display(A + B);
```

Granting of Lock:

When a transaction requests a lock on a data item in a particular mode, and no other transaction has a lock on the same data item in a conflicting mode, the lock can be granted. However, care must be taken to avoid the following scenario. Suppose a transaction T2 has a shared-mode lock on the data item. Clearly, T1 has to wait for T2 to release the shared-mode lock. Meanwhile, a transaction T3 may request a shared-mode lock on the same data item. The lock request is compatible with the lock granted to T2, so T3 may be granted the shared-mode lock. At this point T2 may release the lock, but still T1 has to wait for T3 to finish. But again, there may be a new transaction T4 that requests a shared-mode lock on the same data item, and is granted the lock before T3 releases it. In fact, it is possible that there is a sequence of transaction releases the lock a short while after it is granted, but T1 never gets the exclusive-mode lock on the data item. The transaction T1 may never make progress, and is said to be **starved.**

THE TWO – PHASE LOCKING PROTOCOL

One protocol that ensures serializability is the **two-phase locking protocol**. This protocol requires that each transaction issue lock and unlock requests in two phases:

1. **Growing phase.** A transaction may obtain locks, but may not release any lock.
2. **Shrinking phase.** A transaction may release locks, but may not obtain any new locks.

Initially, a transaction is in the growing phase. The transaction acquires locks as need. Once the transaction releases a lock, it enters the shrinking phase, and it can issue no more lock request.

Implementation of locking:

A lock-manager can be implemented as a process that receives messages from transactions and sends messages in reply.

The lock manager process requests in the following way:

1. When a lock request message arrives, it adds a record to the end of the linked list for the data item, if the linked list is present. Otherwise it creates a new linked list, containing only the record for the request.
2. When the lock manager receives an unlock message from a transaction, it deletes the record for that data item in the linked list corresponding to that transactions.
3. If a transaction aborts, the lock manager deletes any waiting request made by the transactions.

5.a. **Explain how Dead lock will be handled.** [10]

A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. More precisely, there exists a set of waiting transactions {T_0, T_1 … T_n}such that T_0 is waiting for a data item that T_1 holds, and T_1 is waiting for a data item that T_2 holds, and …, and T_{n-1} is waiting for a data item that T_n holds, and T_n is waiting for a data item that T_0

holds. None of the transactions can make progress in such a situation.

The only remedy to this undesirable situation is for the system to invoke some drastic action, such as rolling back some of the transactions involved in the deadlock. Rollback of a transaction may be partial. That is, a transaction may be rolled back to the point where it obtained a lock whose release resolves the deadlock.

There are two principal methods for dealing with the deadlock problem. We can use a **deadlock prevention** protocol to ensure that the system will never enter a deadlock state. Alternatively, we can allow the system to enter a deadlock state, and then try to recover by using a **deadlock detection** and **deadlock recovery** scheme.

DEADLOCK PREVENTION

Two different deadlock prevention schemes using timestamps have been proposed:

1. The **wait-die** scheme is a nonpreemptive technique. When transaction Ti requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (that is, T_i is older than T_j). Otherwise, T_i is rolled back (dies).

For example, suppose that transactions T_{22}, T_{23}, and T_{24} have timestamps 5, 10, and 15, respectively. If T_{22} requests a data item held by T_{23}, then T_{22} will wait. If T_{24} requests a data item held by T_{23}, then T_{24} will be rolled back.

2. The **wound-wait** scheme is a preemptive technique. It is a counterpart to the **wait-die** scheme.

TIMEOUT BASED SCHEMES

Another approach for dead lock handling is based on lock timeouts. In this case a transaction waits for a lock at most a specified amount of time. If the lock is not granted the transaction rolls itself back.

DEAD LOCK DETECTION AND RECOVERY

In order to recover from a dead lock the system must:

- Maintain information about the current allocation of data items to rtransactions, as well as any outstanding data items requests.
- Provide an algorithm that uses this information to determine whether the system has entered a deadlock state.

Recover from the deadlock when the detection algorithm determines that a deadlock exists.

6	a. **Define trigger? Explain with example.**	[5]

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. DML triggers execute when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view.

AFTER trigger

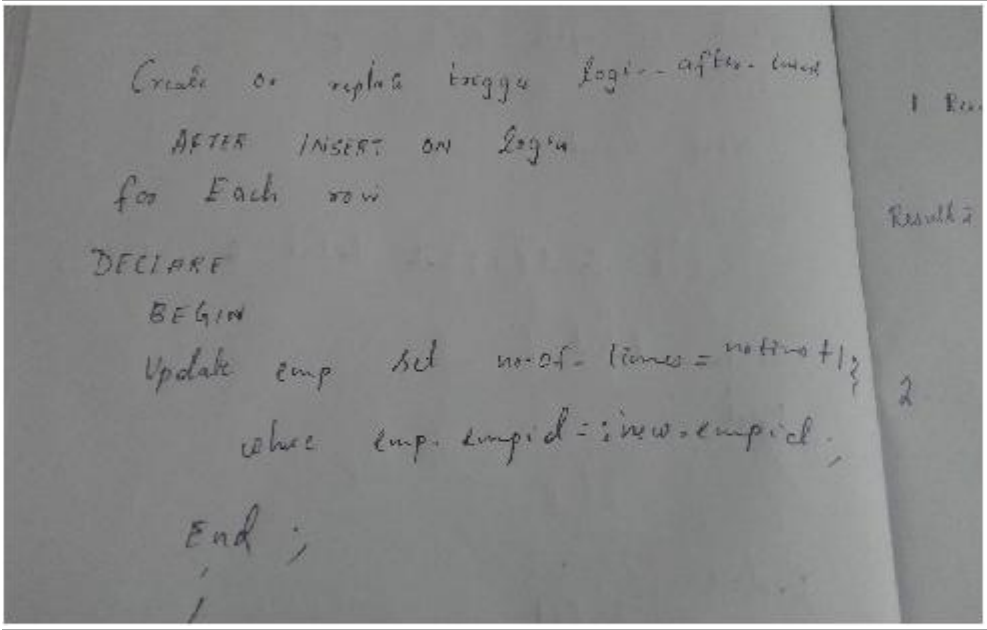
AFTER triggers are executed after the action of the INSERT, UPDATE, MERGE, or DELETE statement is performed. AFTER triggers are never executed if a constraint violation occurs; therefore, these triggers cannot be used for any processing that might prevent constraint violations. For every INSERT, UPDATE, or DELETE action specified in a MERGE statement, the corresponding trigger is fired for each DML operation.

INSTEAD OF trigger

INSTEAD OF triggers override the standard actions of the triggering statement. Therefore, they can be used to perform error or value checking on one or more columns and the perform additional actions before insert, updating or deleting the row or rows. For example, when the value being updated in an hourly wage column in a payroll table exceeds a specified value, a trigger can be defined to either produce an error message and roll back the transaction, or insert a new record into an audit trail before inserting the record into the payroll table. The primary advantage of INSTEAD OF triggers is that they enable views that would not be updatable to support updates. For example, a view based on multiple base tables must use an INSTEAD OF trigger to support inserts, updates, and deletes that reference data in more than one table. Another advantage of INSTEAD OF triggers is that they enable you to code logic that can reject parts of a batch while letting other parts of a batch to succeed.

CLR Triggers

A CLR Trigger can be either an AFTER or INSTEAD OF trigger. A CLR trigger can also be a DDL trigger. Instead of executing a Transact-SQL stored procedure, a CLR trigger executes one or more methods written in managed code that are members of an assembly created in the .NET Framework and uploaded in SQL Server.



```
Create or replace trigger login_after_insert
    AFTER INSERT ON login
for Each row
DECLARE
    BEGIN
    Update emp set no-of-times = no-of-times+1;
        where emp.empid=:new.empid;
    End;
    /
```

b.**Define stored procedure? Explain with example.** [5]

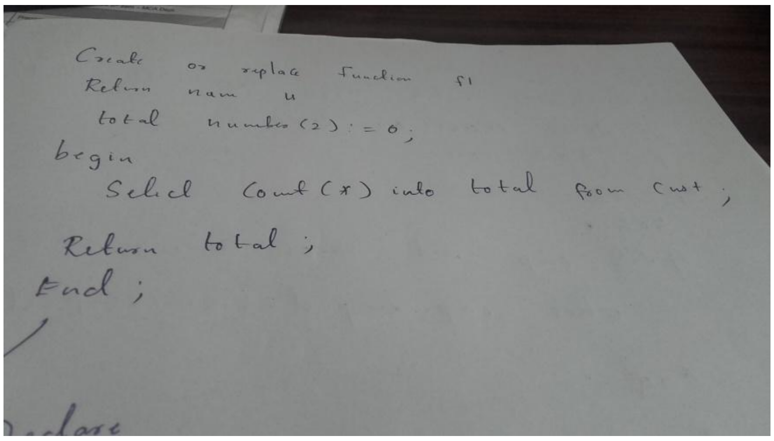
```
CREATE [OR REPLACE] PROCEDURE procedure_name
    [ (parameter [,parameter]) ]
IS
    [declaration_section]
BEGIN
    executable_section
[EXCEPTION
    exception_section]
```

```
END [procedure_name];
```

When we create a procedure or function, you may define parameters. There are three types of parameters that can be declared:

1. IN - The parameter can be referenced by the procedure or function. The value of the parameter can not be overwritten by the procedure or function.
2. OUT - The parameter can not be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.

IN OUT - The parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function



```
Create or replace function f1
Return num is
 total number(2) := 0;
begin
  Select Count(*) into total from Cust;
 Return total;
End;
/
Declare
```

7 a. **Explain the classification of failures.** [5]

Failure of classification:

1. Transaction Failure – Transaction has to abort when it fails to execute or reaches a point where it cannot go further.

   Transaction failures can be either Logical or System errors.
2. System Crash: Interruption to power-supply.

3. Disk-Failure- hard-disk/ Storage drivers used to fail.

b. **Explain the different phases of recovery algorithm.** [5]

LSN- Log Sequence Number

ARIES recovers from a system crash in 3 phase:

1. Analysis Pass
2. Redo Pass
3. Undo Pass

Analysis Pass: This pass determines which transactions to undo, which pages were dirty at the time of crash and the LSN from which the redo pass should start.

Redo Pass: This pass starts from a position determined during analysis pass and performs a redo, repeating history, to bring the database to a state it was in before crash.

Undo Pass: This pass toll-back all transactions that were in-complete at the time of crash.

8 a. **Explain ACID properties in detail.** [10]

Consistency: The consistency requirement here is that sum of A and B be unchanged by the execution of transaction. Without the consistency requirement, money could be created or destroyed by a transaction. It can be verified easily that if the database is consistent before an execution of the transaction, the database remains consistent after the execution of the transaction. Ensuring consistency for an individual transaction if the responsibility of the application programmer, who codes the transaction.

Durability: Once the execution of the transaction completes successfully, and the user who initiated the transaction has been notified that the transfer of funds has taken place, it must be the case that no system failure will result in a loss of data corresponding to this transfer of funds.

Atomicity: That is the reason for the atomicity requirement: if the atomicity property is present, all actions of the transaction are reflected in the database or none are. The database system keeps track of the old values on the disk, on which transaction performs a write and if the transaction does not complete, the database system restores the old values to make it appear that the transaction never occurred. Ensuring atomicity is the responsibility of the database system itself; it is handled by transaction-management component.

- **Isolation:** Even if the consistency and atomicity properties are ensured for each transaction, if several transactions are executed concurrently, their operations may interleave in some undesirable way resulting in an inconsistent state.

The isolation property of a transaction ensures that the concurrent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order.