Benha University
1st Term (December 2013) Final Exam
Class: 4th Year Students **(IS)**
Subject: Management Information System

Faculty of Computers & Informatics
Date: 26 / 12 /2013
Time: 3 Hours
Examiner: Dr. Sahar Fawzy

Question One

1- **"While they won't displace traditional RDBMSs, the easy scalability and programmability of NoSQL databases guarantees them a permanent place in the data center."** in the sight of this statement, answer the following:

a) What is Wrong with RDBMS?

- → For the longest time (and still true today), the big relational database vendors such as Oracle, IBM, Sybase, and a lesser extent Microsoft were the mainstay of how data was stored.
- During the Internet boom, startups looking for low-cost RDBMS alternatives turned to MySQL and PostgreSQL.
- Hooking your RDBMS to a web-based application was a recipe for headaches, they are OLTP in nature. Could have hundreds of thousands of visitors in a short-time span.
- To mitigate, began to front the RDBMS with a read-only cache such as memcache to offload a considerable amount of the read traffic.
- As datasets grew, the simple memcache/MySQL model (for lower-cost startups) started to become problematic.
- Nothing. One size fits all? Not really.
- Impedance mismatch.
  - Object Relational Mapping doesn't work quite well.
- Rigid schema design.
- Joins across multiple nodes? Hard.
- How does RDMS handle data growth? Hard.
- Need for a DBA.
- ACID limits scaling.
- RDBMS were not designed to be distributed, Began to look at multi-node database solutions Known as 'scaling out' or 'horizontal scaling' Different approaches include:
  - Master-slave & Sharding

b) Show How BASE is an Opposed to ACID.

- IN Cloud computing: ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.
- Availability
  - Traditionally, thought of as the server/process available 99.999 % of time
  - For a large-scale node system, there is a high probability that a node is either down or that there is a network partitioning
- Partition tolerance ensures that write and read operations are redirected to available replicas when segments of the network become disconnected

- Eventual Consistency
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
  - For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) properties, as opposed to ACID
  - Soft state: copies of a data item may be inconsistent
  - Eventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item
  - Basically Available – possibilities of faults but not a fault of the whole system

---

c) "MongoDB is an example of document based databases that has a flexible schema" Show with examples how MongoDB can be used to model tree structures.

➔ Consider the following example that keeps a library book and its checkout information. The example illustrates how embedding fields related to an atomic update within the same document ensures that the fields are in sync.

Consider the following book document that stores the number of available copies for checkout and the current checkout information:

```
book = {
      _id: 123456789,
      title: "MongoDB: The Definitive Guide",
      author: [ "Kristina Chodorow", "Mike Dirolf" ],
      published_date: ISODate("2010-09-24"),
      pages: 216,
      language: "English",
      publisher_id: "oreilly",
      available: 3,
      checkout: [ { by: "joe", date: ISODate("2012-10-15") } ]
        }
```

You can use the db.collection.findAndModify() method to atomically determine if a book is available for checkout and update with the new checkout information. Embedding the available field and the checkout field within the same document ensures that the updates to these fields are in sync:

```
db.books.findAndModify ( {
      query: {
          _id: 123456789,
          available: { $gt: 0 }
                },
     update: {
          $inc: { available: -1 },
          $push: { checkout:
                    { by: "abc", date: new Date() } }          }} )
```

➔ The *Child References* pattern stores each tree node in a document; in addition to the tree node, document stores in an array the id(s) of the node's children.

➔ Consider the following example that models a tree of categories using *Child References*:

```
db.categories.insert( { _id: "MongoDB", children: [] } )
db.categories.insert( { _id: "Postgres", children: [] } )
db.categories.insert( { _id: "Databases", children: [ "MongoDB", "Postgres" ] } )
db.categories.insert( { _id: "Languages", children: [] } )
db.categories.insert( { _id: "Programming", children: [ "Databases", "Languages" ] } )
db.categories.insert( { _id: "Books", children: [ "Programming" ] } )
```

The query to retrieve the immediate children of a node is fast and straightforward:

```
db.categories.findOne( { _id: "Databases" } ).children
```

You can create an index on the field children to enable fast search by the child nodes:

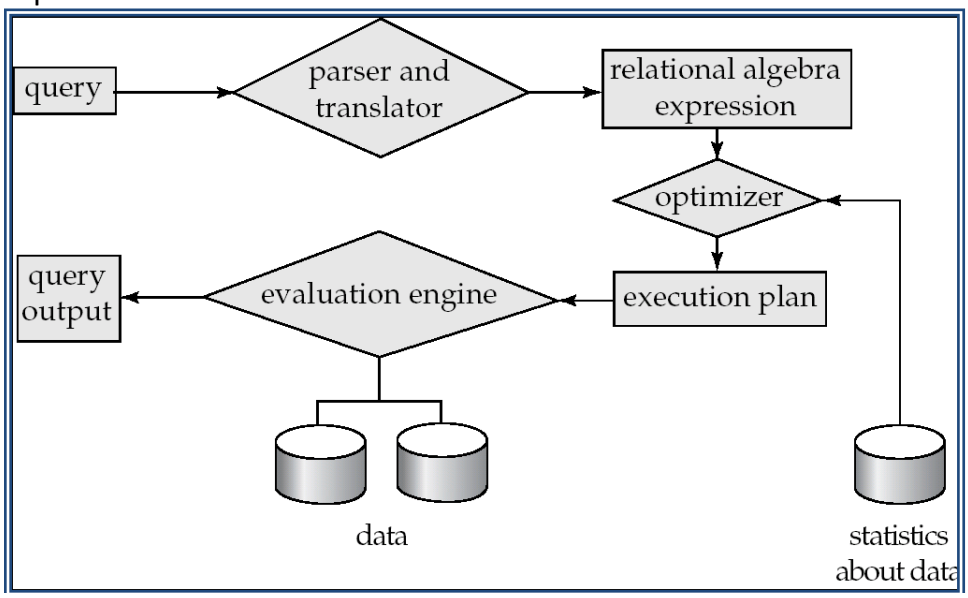
```
db.categories.ensureIndex( { children: 1 } )
```

You can query for a node in the children field to find its parent node as well as its siblings:

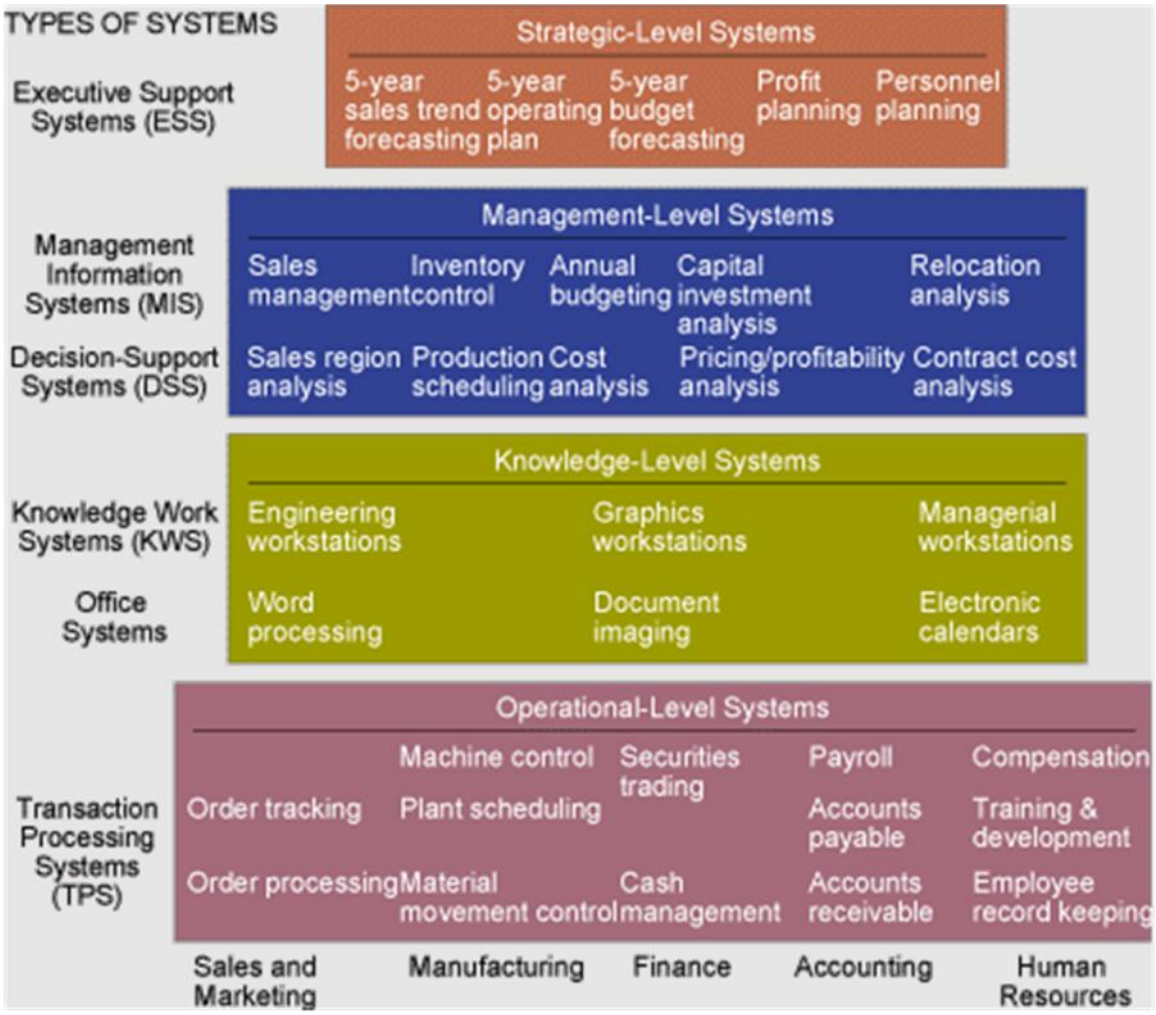
```
db.categories.find( { children: "MongoDB" } )
```

The Child References pattern provides a suitable solution to tree storage as long as no operations on subtrees are necessary. This pattern may also provide a suitable solution for storing graphs where a node may have multiple parents.

2- Discuss briefly the steps of query cycle, showing the role of query parser, analyzer and optimizer.



3- In some companies, we usually see some levels of managements, name these levels showing the basic types of Is to support each of them.



4- write Relational Algebra expressions (not SQL) for the following queries with respect to the database below (primary key attributes are boldfaced and underlined:

D: (**DEPT** DNAME BUDGET) fordepartments
T: (**T**# TNAME CITYDEPT) forteachers
S: (**S**# SNAME CITY DEPT DEGREE) for students
C: (**C**# CNAME DEPT T#) for courses
E: (**S**# **C**# GRADE) forenrollments

Note: Here ′ECE′, ′CSC′ are values of the attribute "DEPT" rather than "DNAME".

a- Get S#, SNAME for ′Baton Rouge′ students from ′CSC′ department.

Answer:

$\Pi_{S\#,\ SNAME}(\sigma_{\ CITY\ =\ 'Baton\ Rouge'\ \wedge\ DEPT\ =\ 'CS\ C'}(S))$

Alternatively:

$\Pi_{S\#,\ SNAME}(\sigma_{\ CITY\ =\ 'Baton\ Rouge'}(S)) \cap \Pi_{S\#,\ SNAME}(\sigma_{\ DEPT\ =\ 'CS\ C'}(S))$

b- Get S#, C# pairs for students and courses such that the student is enrolled in the course with a grade ≥ 80.

Answer:

$\Pi_{S\#,C\#}(\sigma_{GRADE \geq 80}(E))$

c- Get C#, Cname for courses which are taken by a student from ′ECE′ department.

Answer:

$R_1 \leftarrow E \text{ JOIN } (\sigma_{DEPT = 'EC\ E'}(S))$ JOIN $(\Pi_{C\#,\ CNAME}(C))$

$result = \Pi_{C\#,CNAME}(R_1)$

Note: here I am using the word "JOIN" to denote the natural join operator

d- Get S# for students who are either from ′New Orleans′ or taking the course (C#=) ′C4′ (or both).

Answer:

$\Pi_{S\#}(\sigma_{CITY = 'New\ Orleans'}(S)) \cup \Pi_{S\#}(\sigma_{C\# = 'C4'}(E))$

e- get S#, T# pairs such that the student and the teacher are from **the same** department and the student takes a course taught by the teacher.

Answer:

$\Pi_{S\#,\ T\#}(S \text{ JOIN } (\Pi_{T\#,\ DEPT}(T))) \cap \Pi_{S\#,\ T\#}(E \text{ JOIN } C)$

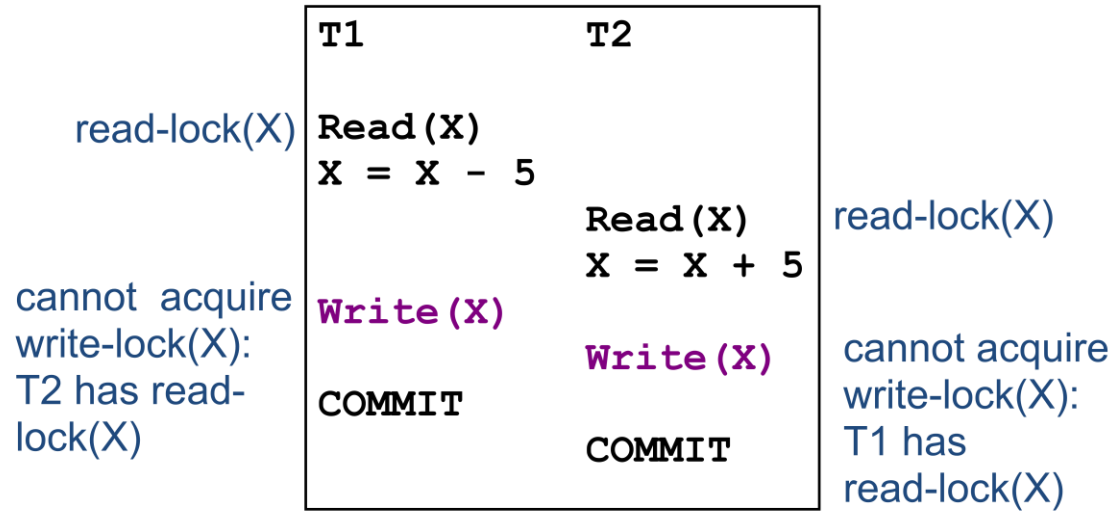
Question Two:

1- Explain with examples the use two phase locking protocols to solve each of the concurrency problems.

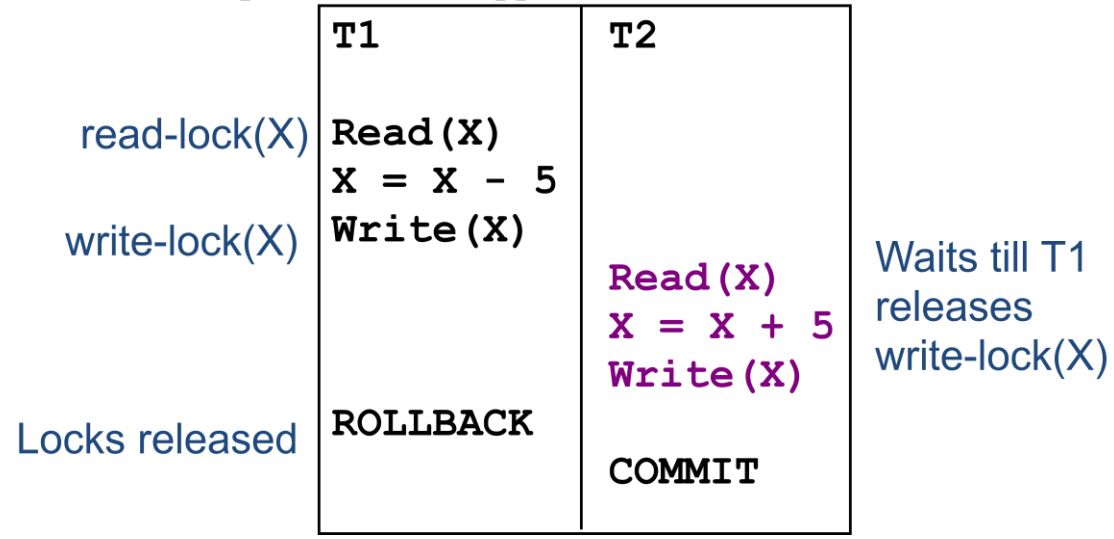
➔ **A transaction follows the *two-phase locking protocol* (2PL) if all locking operations precede the first unlock operation in the transaction. Two phases**

a) **Growing phase where locks are acquired on resources**
b) **Shrinking phase where locks are released**

Lost update cannot happen as follows:



Uncommitted update cannot happen

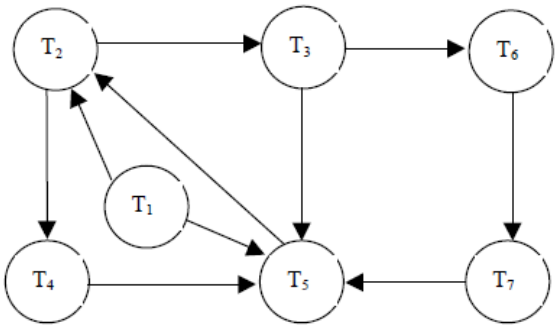


Inconsistent analysis cannot happen as well

	T1	T2	
read-lock(X)	Read(X)		
	X = X - 5		
write-lock(X)	Write(X)		
		Read(X)	Waits till T1 releases write-locks on X and Y
		Read(Y)	
		Sum = X+Y	
read-lock(Y)	Read(Y)		
	Y = Y + 5		
write-lock(Y)	Write(Y)		

2- Produce a wait-for-graph for the following transaction scenario and determine whether deadlock

Transaction	*Data items locked by transaction*	*Data items transaction is waiting for*
T_1	X_2	X_1, X_3
T_2	X_3, X_{10}	X_7, X_8
T_3	X_8	X_4, X_5
T_4	X_7	X_1
T_5	X_1, X_5	X_3
T_6	X_4, X_9	X_6
T_7	X_6	X_5



3- Classify the following relations as either UNNORMALISED, 1NF, 2NF or 3NF. If the relation is not in 3NF, normalise the relation to 3NF.

a. ***EMPLOYEE(<u>empno</u>,empname,jobcode)***
empno -> empname
empno -> jobcode

➔ ***3rd NF***

b. ***EMPLOYEE(<u>empno</u>,empname,(jobcode,years))***
empno -> empname
empno,jobcode -> years

➔ ***Unnormalized.***

➔ ***Employee(Empno , empname), employeeJob(Empno,jobcode,years)***

c. ***EMPLOYEE(<u>empno</u>,empname,jobcode,jobdesc)***
empno -> empname,jobcode
jobcode -> jobdesc

➔ ***Relation is in 2nd NF***

➔ ***Employee(empno,empname,jobcode) & jobs(jobcode , Jobdesc)***

d. ***EMPLOYEE(<u>empno</u>,empname,<u>project</u>,hoursworked)***
empno -> empname
empno,project -> hoursworked

➔ ***Relation is in 1st Nf as there exist partial dependency***

➔ ***Employee(empno,empname) & EmployeeProject(empno,project,hoursworked)***

4- Identify any repeating groups and functional dependences in the PATIENT relation. Show all the intermediate steps to derive the third normal form for PATIENT.

PATIENT(patno,patname,gpno,gpname,appdate,consultant,conaddr,sample)

patno	patname	gpno	gpname	appdate	consultant	conaddr	sample
01027	Grist	919	Robinson	3/9/2004	Farnes	Acadia Rd	blood
				20/12/2004	Farnes	Acadia Rd	none
				10/10/2004	Edwards	Beech Ave	urine
08023	Daniels	818	Seymour	3/9/2004	Farnes	Acadia Rd	none
				3/9/2004	Russ	Fir St	sputum
191146	Falken	717	Ibbotson	4/10/2004	Russ	Fir St	blood
001239	Burgess	818	Seymour	5/6/2004	Russ	Fir St	sputum
007249	Lynch	717	Ibbotson	9/11/2004	Edwards	Beach Ave	none

➔ 1st Nf: Patient (**patno**, patname, gpno, gpname) & Appt (**patno, appdate, consultant**, conaddr, sample)

➔ 2nd NF: Patient (**patno**, patname, gpno, gpname) & Appt (**patno, appdate, consultant**, sample) & Consultant (**consultant**, conaddr)

➔ **3rd Nf:** Patient (**patno**, patname, gpno)& GP (**gpno**, gpname) & App (**patno, appdate, consultant**, sample) & Consultant (**consultant**, conaddr)

Best Wishes & Good Luck

Dr. Sahar