Final Examination
"Course 1-1023221-0: Database Systems"
Exam A Answers

Kinneret College School of Engineering

January 28, 2009 9:00am-12:00pm

- Answer the following questions in English or Hebrew.
- You may bring one page of notes to the exam with notes on both sides.
- The number of points for each question is listed next to each one to indicate its weight.
- The number of points for each question is listed next to each one to indicate its weight. You must complete a total of **120** points on the test to receive a 100% grade. Since there are a total of 155 points possible, you may get up to 35 points off and still receive a full grade.
- You will not receive a grade greater than 100%, even if you receive more than 120 points.
- You are encouraged to try all of the problems. I will grade them all.

1 Question 1 (18 points / 3 points each)

Briefly explain the following terms as they relate to databases:

1. Foreign Key
   - A field (or set of fields) in one relation which is constrained to only contain null values or values which appear in the primary key field(s) of another (or in some cases, the same) relation.
2. Transaction
   - One execution of a user program in a database. Transactions are collections of primitive operations whose execution is atomic (all or nothing).
3. Participation Constraint
   - The requirement that all entities in an entity set are associated with some relationship. For instance, for entity sets E_1, E_2 and relationship R with a participation constraint on E_1, $\forall e_1 \in E_1 \exists e_2 \in E_2.(e_1, e_2) \in R$.
4. Recoverable Schedule
   - An interleaving of transactions which allows for full recovery of correct state in case any transaction aborts or does not complete successfully.
5. Forcing Pages (to disk)

- The policy that all dirty pages (writes) from a committed transaction be stored to persistent storage (disk) immediately after its completion.

6. Serializable Schedule

   - An interleaving of transactions whose outcome is equivalent to *some* serial execution of the (non-aborted) transactions.

2 Question 2 (12 points / 6 points each)

Consider the following scenarios about entities, relationships, and constraints. Draw an ERD for each scenario. Each scenario may be considered *independently* of the others in the list.

1. Each student has a national identification number, a name, an address, and a phone number. Younger students often share the same address, and no address has more than one phone.

2. A local library has borrowers and elderly volunteers. Each volunteer has a national identification number, a day he works at the library and a specialty. Each borrower has a national identification number a unique library card number, and an address. A volunteer can borrow books *only* if she is also registered as a borrower. A book can only be borrowed by one borrower at a time.

Answer There are many options for how the ERD could be written. Two potential ideas are shown below. Figure 1 shows a potential ERD for the first scenario. Figure 2 shows a potential ERD for the second scenario.

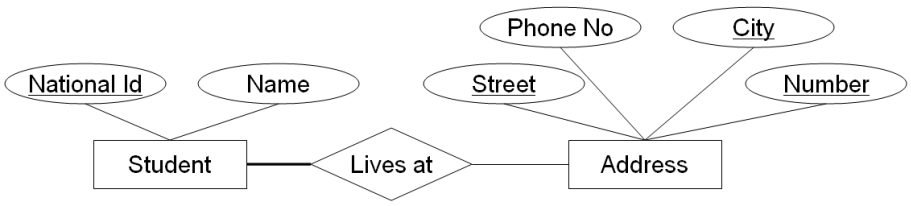


Figure 1: Potential ERD for Question 2 Part 1

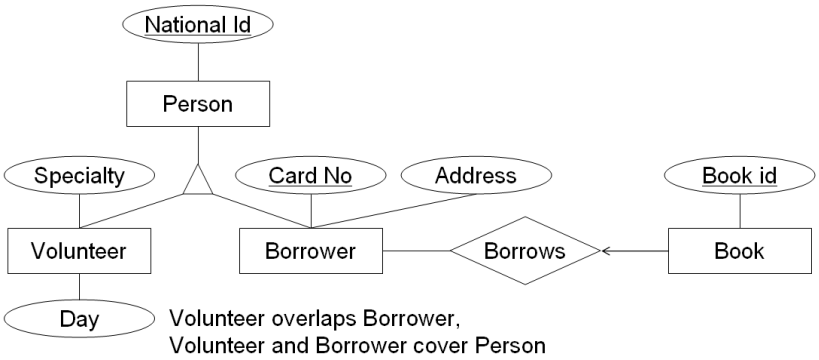
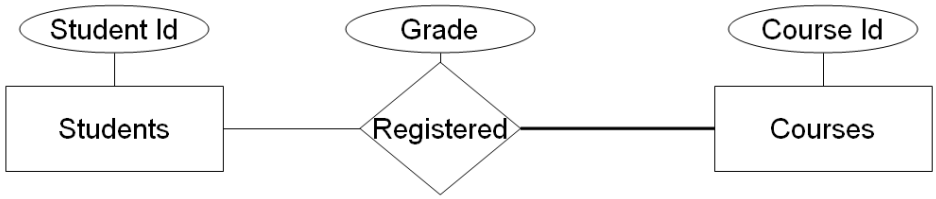


Figure 2: Potential ERD for Question 2 Part 2

3 Question 3 (20 points / 5 points each)

Consider the following ERD diagrams. Write SQL `CREATE TABLE` commands which create relations which implement them. Include information about primary and foreign keys and `NOT NULL` constraints. If a constraint can not be enforced using primary keys, foreign keys, and `NOT NULL`, write code (`CHECK`, `ASSERTION`, or `STORED PROCEDURE`) which will check it.

3.1 Diagram 1



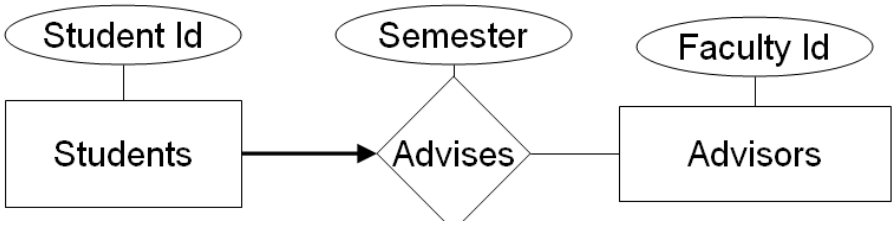
Answer

```
CREATE TABLE Students (studentId INTEGER, PRIMARY KEY (studentId));

CREATE TABLE Courses (courseId INTEGER,
                      PRIMARY KEY (courseId),
                      CHECK NOT EXISTS (SELECT * FROM Courses C WHERE NOT EXISTS
                        (SELECT * from Registered R WHERE R.courseId = C.courseId)));

CREATE TABLE Registered (studentId INTEGER, courseId INTEGER, grade REAL,
                         PRIMARY KEY (studentId, courseid),
                         FOREIGN KEY (studentId) REFERENCES Students,
                         FOREIGN KEY (courseId) REFERENCES Courses);
```

3.2 Diagram 2

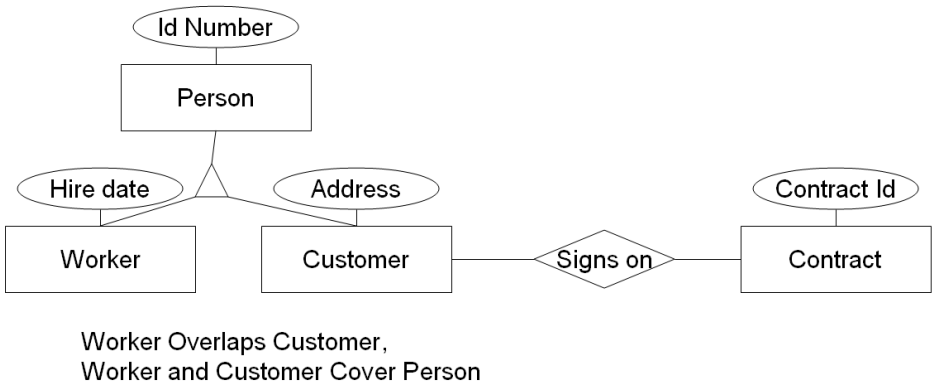


Answer

```
CREATE TABLE Students (studentId INTEGER, advisorId INTEGER NOT NULL,
                       semester VARCHAR(20),
                       PRIMARY KEY (studentId),
                       FOREIGN KEY (advisorId) REFERENCES Advisors);

CREATE TABLE Advisors (facultyId INTEGER, PRIMARY KEY (facultyId));
```

3.3 Diagram 3



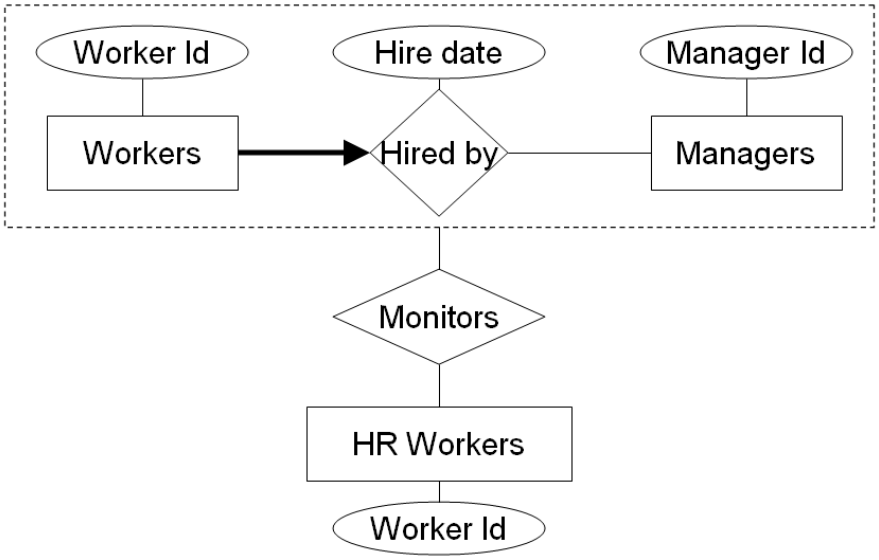

```
CREATE TABLE Worker (idNumber INTEGER, hireDate DateTime,
                     PRIMARY KEY (idNumber));

CREATE TABLE Customer (idNumber INTEGER, address VARCHAR(50),
                       PRIMARY KEY (idNumber));

CREATE TABLE Contract (contractId INTEGER, PRIMARY KEY (contractId));

CREATE TABLE SignsOn (customerId INTEGER, contractId INTEGER,
                      PRIMARY KEY (customerId, contractId),
                      FOREIGN KEY (customerId) REFERENCES Customer,
                      FOREIGN KEY (contractId) REFERENCES Contract);
```

3.4 Diagram 4




```
CREATE TABLE Workers (workerId INTEGER, hiredBy INTEGER NOT NULL,
                      hireDate DateTime,
                      PRIMARY KEY (workerId),
```

```
                    FOREIGN KEY (hiredBy) REFERENCES Manager);

CREATE TABLE Managers (managerId INTEGER, PRIMARY KEY (managerId));

CREATE TABLE HRWorkers (workerId INTEGER, PRIMARY KEY (workerId));

CREATE TABLE Monitors ( workerId INTEGER, hrWorker INTEGER,
                        PRIMARY KEY (workerId, hrWorker),
                        FOREIGN KEY (workerId) REFERENCES Workers,
                        FOREIGN KEY (hrWorker) REFERENCES HRWorkers);
```

4 Question 4 (66 points)

Consider the following relational schema. An employee can work in more than one department; the pct_time field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(*eid*:`integer`, *ename*:`string`, *age*:`integer`, *salary*:`real`)
Works(*eid*:`integer`, *did*:`integer`, *pct_time*:`integer`)
Dept(*did*:`integer`, *dname*:`string`, *budget*:`real`, *managerid*:`integer`)

You will first write relational algebra queries and then similar SQL queries.

A table with all of the relational algebra and set manipulation operators and their meanings is shown in Table 1 to help with this question.

Symbol	Meaning
σ	Select (rows)
π	Project (columns)
$\rho(N(\overline{C}), O)$	Renames table O to be called N. Changes column names as per $\overline{C}$
$\bowtie_c$	Conditional Join / Equijoin
$\bowtie$	Natural Join
$/$	Division
$\times$	Cartesian Cross Product
$\cup$	Set Union
$\cap$	Set Intersect
-	Set Difference

Table 1: Relational Algebra Operators

4.1 Relational Algebra Queries (30 points / 6 points each)

Write relational algebra statements for the following queries. You may use more than one line for the relational algebra statement if necessary.

1. Select the names and ages of each employee who works in both the Hardware department and the Software department.

   $\pi_{(ename,age)}(\sigma_{dname=\text{“}Hardware''}(Emp \bowtie Works \bowtie Dept) \cap \sigma_{dname=\text{“}Software''}(Emp \bowtie Works \bowtie Dept))$

2. Select the *enames* of managers who manage at least two departments.

   $\pi_{ename}(Emp \bowtie_{(managerId=eid)} (\pi_{D_1.managerId}(\sigma_{(D_1.did \neq D_2.did \wedge D_1.managerId=D_2.managerId)}(\rho(D_1, Dept) \times \rho(D_2, Dept)))))$

3. Select the ages of employees who work in *all* departments in which there is some employee

   $\pi_{age}(Emp \bowtie (\pi_{eid}Emp/\pi_{(eid,did)}Works))$

4. Select the names of employees whose *eid* is greater than at least one department in which they work's *did*.

   $\pi_{ename}(Emp \bowtie_{(Emp.eid>Works.did),Emp.eid=Works.eid} Works)$

5. Select the budgets for all departments which employ at least one employee under 18 years old.

   $\pi_{budget}(\sigma_{(age<18)}Emp \bowtie Works \bowtie Dept)$

4.2 SQL Queries (36 points / 6 points each)

Write SQL statements for the following queries. You may use grouping, nesting, or join operators in your statements.

1. Select the names and ages of each employee who works in both the Hardware department and the Software department.

```
SELECT ename, age
FROM Emp E1, Dept D1, Works W1
WHERE E1.eid = W1.eid
AND D1.did = W1.did
AND D1.dname = ‘‘Hardware’’

INTERSECT

(SELECT ename, age
FROM Emp E1, Dept D1, Works W1
WHERE E1.eid = W1.eid
AND D1.did = W1.did
AND D1.dname = ‘‘Software’’)
```

2. Select the *enames* of managers who manage at least two departments.

```
SELECT ename
FROM Emp E1, Dept D1, Dept D2
WHERE E1.eid = D1.managedId
AND E1.eid = D2.managerId
AND D1.did <> D2.did
```

3. Select the **MINIMUM age of employees** who work in *all* departments in which there is some employee

```
SELECT MIN(AGE) FROM Emp E1 WHERE NOT EXISTS
    (SELECT did FROM Works W2

    MINUS

    (SELECT W3.did
    FROM Works W3
    WHERE E1.eid = W3.eid)
    )
```

4. Select the average salaries of employees under 18 in departments with at least 2 such employees grouped by the number of employees under 18 in the department.

For example:

average salary	number under 18
100	2
102	3
4	10
...	...

```
SELECT AVG(avg_sal) average, num
FROM (
    SELECT AVG(salary) as avg_sal, COUNT(*) as num
    FROM Works W, Emp E
    WHERE E.eid = W.eid
    AND E.age < 18
    GROUP BY W.did
    HAVING COUNT(*) >= 2) byDept
GROUP BY num
```

5. Find the **SUM of the budgets** for all departments which only employ employees under 18 years old.

```
SELECT SUM(budget) FROM Dept D1
WHERE NOT EXISTS (SELECT *
                  FROM Works W2, Emp E2
                  WHERE W2.did = D1.did
                  AND E2.eid = W2.eid
                  AND E2.age >= 18)
```

6. Print the *eid*'s for employees whose total appointments are greater than 100% (this means that the sum of their associated pct_time > 100).

```
SELECT eid FROM Emp E1
WHERE 100 <
    (SELECT SUM(pct_time)
    FROM Works W2
    WHERE W2.eid = E1.eid);
```

5 Question 5 (24 points)

Consider the following database schema. It is from a factory which has workers and machines. Workers maintain the machines. Information about the machines is stored in the Machines relation, including the last date the machine was maintained. Each maintenance event is recorded in the Maintenance relation, including the date of the maintenance and how many hours it took. Each worker is described in the Workers relation, including the number of hours the worker has worked so far.

Machines (*mid*:`int`, *mname*:`string`, *lastMaintDateTime*:`DateTime`)
Maintenance (*wid*:`int`, *mid*:`int`, *dateTime*:`DateTime`, *numHours*:`int`)
Workers (*wid*:`int`, *wname*:`string`, *hoursWorked*:`int`)

Assume the above tables have been initialized as follows:

Machines:

mid	mname	lastMaintDateTime
1	'AMachine'	2001-1-1 1:01:01
2	'BMachine'	2002-2-2 2:02:02

Workers:

wid	wname	hoursWorked
10	'Chuck'	9
11	'Dan'	10

Maintenance

wid	mid	dateTime	numHours
10	1	2000-1-1 11:11:11	1
10	1	2000-1-1 12:12:12	3
10	1	2001-1-1 1:01:01	5
11	2	2002-2-2 2:02:02	10

5.1 Tables SQL (9 points)

Write SQL code which will create the above tables. Include all relevant primary and foreign key constraints.

```
CREATE TABLE Machines (mid INTEGER,
                       mname VARCHAR(20),
                       lastMaintDateTime DateTime,
                       PRIMARY KEY (mid));

CREATE TABLE Workers (wid INTEGER,
                      wname VARCHAR(20),
                      hoursWorked INTEGER,
                      PRIMARY KEY (wid),
                      CHECK (hoursWorked >= 0));

CREATE TABLE Maintenance (wid INTEGER,
                          mid INTEGER,
                          dateTime DateTime,
                          numHours INTEGER,
    PRIMARY KEY(wid, mid, dateTime),
    FOREIGN KEY (mid) REFERENCES Machines,
    FOREIGN KEY (wid) REFERENCES Workers);
```

5.2 Table Updates (6 points)

Write SQL code which will change the third row of Maintenance to: (11, 1, 2001-1-1 1:01:01, 6).

```
UPDATE Maintenance M
    SET wid = 11
    WHERE M.wid = 10
    AND M.mid = 1
    AND M.dateTime = 2001-1-1 1:01:01;

UPDATE Maintenance M
    SET numHours = 6
    WHERE M.wid = 11
```

```
    AND M.mid = 1
    AND M.dateTime = 2001-1-1 1:01:01;
```

Write SQL code which will insert the following tuples into Maintenance: (11, 1, 2002-2-2 2:02:02, 5), (10, 2, 2003-3-3 3:03:03, -1).

```
INSERT INTO Maintenance (wid, mid, dateTime, numHours)
    VALUES (11, 1, 2002-2-2 2:02:02, 5);

INSERT INTO Maintenance (wid, mid, dateTime, numHours)
    VALUES (10, 2, 2003-3-3 3:03:03, -1);
```

5.3 Triggers (9 points)

Consider the following triggers written in SQL syntax:

```
CREATE TRIGGER trigger1 BEFORE INSERT ON Maintenance
REFERENCING NEW ROW AS nrow
FOR EACH ROW
WHEN nrow.numHours < 0
SET nrow.numHours = 0

CREATE TRIGGER trigger2 AFTER UPDATE ON Maintenance
REFERENCING NEW ROW AS nrow, OLD ROW AS orow
FOR EACH ROW
BEGIN
    UPDATE Workers W
        SET W.hoursWorked = W.hoursWorked - orow.numHours
        WHERE W.wid = orow.wid;
    UPDATE Workers W
        SET W.hoursWorked = W.hoursWorked + nrow.numHours
        WHERE W.wid = nrow.wid;
END

CREATE TRIGGER trigger3 AFTER INSERT ON Maintenance
REFERENCING NEW ROW AS nrow
FOR EACH ROW
WHEN nrow.dateTime > (SELECT MAX(lastMaintDateTime)
                      FROM Machines M
                      WHERE M.mid = nrow.mid)
BEGIN
    UPDATE Machines M
        SET M.lastMaintDateTime = nrow.dateTime
        WHERE M.mid = nrow.mid;
    UPDATE Workers W
        SET W.hoursWorked = W.hoursWorked + nrow.numHours
        WHERE W.wid = nrow.wid;
END
```

Use the tables below to show the state of Machines, Workers, and Maintenance after you have run all of the insert and update commands you wrote in 5.2:

Machines:

mid	mname	lastMaintDateTime
1	'AMachine'	2002-2-2 2:02:02
2	'BMachine'	2002-3-3 3:03:03

Workers:

wid	wname	hoursWorked
10	'Chuck'	4
11	'Dan'	21

Maintenance

wid	mid	dateTime	numHours
10	1	2000-1-1 11:11:11	1
10	1	2000-1-1 12:12:12	3
11	1	2001-1-1 1:01:01	6
11	2	2002-2-2 2:02:02	10
11	1	2002-2-2 2:02:02	5
10	2	2003-3-3 3:03:03	0

6 Question 6 (15 points / 5 points each)

Consider the following two transactions sequences.

T_1: R(A), W(B), R(B), W(A), W(B), Commit.

T_2: R(C), R(A), R(B), W(D), Commit.

Consider the following schedules. For each schedule, write whether it is serializable or not and what serial schedule it is equivalent to. If it is not, explain the problems that prevent it from being serializable.

6.1 Schedule 1

T_1	T_2
R(A)	
W(B)	
	R(C)
R(B)	
	R(A)
W(A)	
W(B)	
Commit	
	R(B)
	W(D)
	Commit

Answer The schedule is not serializable. It is not equivalent to $T_1; T_2$ since T_2 reads A before T_1 writes it (RW). It is not equivalent to $T_2; T_1$ since T_2 reads B which is written by T_1 first.

6.2 Schedule 2

T_1	T_2
	R(C)
	R(A)
R(A)	
W(B)	
R(B)	
W(A)	
W(B)	
Commit	
	R(B)
	W(D)
	Commit

Answer The schedule is not serializable. It is not equivalent to $T_1; T_2$ since T_2 reads A before T_1 has written it. It is not equivalent to $T_2; T_1$ since T_2 reads B after T_1 has written it.

6.3 Schedule 3

T_1	T_2
	R(C)
	R(A)
	R(B)
R(A)	
W(B)	
R(B)	
W(A)	
W(B)	
	W(D)
	Commit
Commit	

Answer The schedule is serializable. It is equivalent to $T_2; T_1$.