Final Examination
Course 1-02-324: Database Systems
Exam A Answers

Kinneret College School of Engineering

January 30, 2011 9:00am - 12:00pm

- Answer the following questions in English or Hebrew.
- You may not use any material or learning aids to the test.
- The number of points for each question is listed next to each one to indicate its weight.
- There are a total of **100** points on the test. You must answer all of the questions.
- Write all of your answers in the test booklet which you received.
- Marks made on the test sheets will not be counted or graded.
- You must return the test questions sheet at the end of the exam.
- You must use proper SQL or MS SQL Server syntax for all answers.

1 Entity Relational Diagrams (30 points)

Read the following story:

> A pizza store manages its customers and orders in a database. Each customer has a customer ID, name, and address. Customers can place orders for pizzas. Each order is placed by one customer and has an order ID and a delivery date/time. An order is for one or more pizzas. Each pizza has a pizza code (from a list of available pizza types - each with a code and a name), a size, and a price.
>
> The price is determined by many factors (bulk discounts, periodic sales, etc.), so the salesman enters in the price specially for each order (*e.g.* in order 34 a 50cm deep dish pizza cost 40 shekels and in order 35 it cost 42 shekels), but all prices must be non-negative. All pizzas of a single type and size will have the same price in a given order (*i.e.* all 50cm deep dish pizzas in order 21 will cost 40 shekels each).
>
> A single order can contain several size/type combinations (*e.g.* three 50cm deep dish, two 40cm deep dish, and four 40cm Sicilian style).

Based on the above story, perform the following actions:

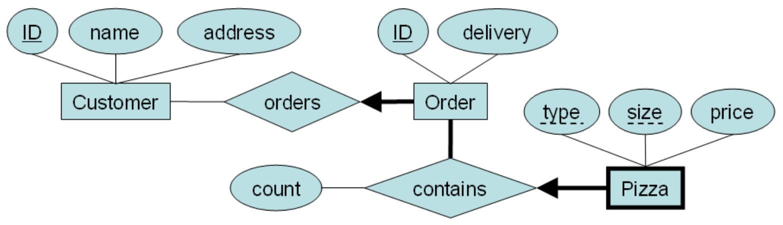
(a) (15 points) Prepare an Entity-Relationship Diagram (ERD) based on the above story. Indicate all attributes, keys, constraints, aggregation, inheritance, or weak entities used (you should not need to use all of them). If you were unable to model some aspect of the story, explain why.

(b) (15 points) Prepare a set of SQL CREATE TABLE definitions based on the ERD you prepared in the previous part. Use correct SQL syntax. Include all primary keys, foreign keys, and constraints. In case you cannot represent a given constraint from the ERD, explain why.

1.1 Answer

The answers per part:

(a) A sample ERD:



Note that the pizza type entry is not found here. It could be added as an extra entity, but it since it is a data type of the "type" field on the pizza entity, it is not necessary to include.

Grading Notes:

- Missing field or attribute **(-2) points**
- Small logical problem (*e.g.* only one pizza size per code, illogical keys) **(-2) points**
- Large logical problem (*e.g.* price is not associated with a pizza, entity with no keys or attributes) **(-3) points**
- Missing key constraint or participation constraint. **(-1) point**

- Reversed key or participation constraint **(-1) point**

(b) A set of tables which implement the above ERD:

```
CREATE TABLE Customers ( id INT PRIMARY KEY,
name CHAR(40),
address CHAR(40))

CREATE TABLE Order ( id INT PRIMARY KEY,
delivery DATETIME,
customerId INT,
FOREIGN KEY (customerId) REFERENCES Customer)

CREATE TABLE PizzaTypes (code INT PRIMARY KEY,
name CHAR(30))

CREATE TABLE Pizza (code INT,
size INT,
price REAL,
orderId INT,
count INT, PRIMARY KEY (orderId, type, size),
FOREIGN KEY (orderId) REFERENCES Order,
FOREIGN KEY (type) REFERENCES PizzaTypes,
CHECK (price >= 0))
```

Note that the participation constraint between Order and Contains could not be included in SQL.

Grading Notes:

- Missing type, missing/extra NOT NULL, spelling error **(-0.5) points**
- Maximum two spelling errors (-1 point) off per table.
- Extra (problematic) fields **(-1) point**
- Missing important fields **(-1) point**
- Missing check for negative prices **(-0.5) points**
- Illogical or missing keys (each)**(-2) point**

2 Relational Algebra and SQL

Consider the following relational schema:

Sailors (<u>sid:`INT`</u>, sname:`VARCHAR(30)`, rating:`INT`, age:`INT`)
Boats (<u>bid:`INT`</u>, bname:`VARCHAR(30)`, color:`VARCHAR(10)`)
Reserves (<u>bid:`INT`, sid:`INT`, day:`DATE`</u>)

2.1 Relational Algebra and SQL (30 points / 10 points each)

Write relational algebra statements and SQL for each of the following queries:

(a) Show the names of all sailors who reserved at least two boats with different colors.

Relational Algebra $\rho(R1, \pi_{(sid,color)}(Reserves \bowtie Boats))$
$\rho(R2, \pi_{(sid,color)}(Reserves \bowtie Boats))$
$\pi_{(sname)}Sailors \bowtie ((R1 \bowtie_{R1.sid=R2.sid \wedge R1.color<>R2.color} R2))$

SQL

```
SELECT sname FROM Sailors S, Reserves R1, Reserves R2, Boats B1, Boats B2
WHERE S.sid = R1.sid AND S.sid = R2.sid
AND R1.bid = B1.sid AND R2.sid = B2.sid
AND B1.color <> B2.color
```

Grading Notes:

- No SQL or no Algebra **(-5) points**
- No reference to color, but searches for two reservations **(-2.5) points**
- SQL that is completely off, but in the general direction **(-4) points**
- Using cross product without renaming the fields with the same name **(-1) point**
- Only using one reservation **(-2) points**.

(b) Show the id and name of each boat which was reserved by each sailor at least once.

Relational Algebra $\pi_{(bid,bname)}Boats \bowtie ((\pi_{(bid,sid)}Reserves)/(\pi_{(sid)}Sailors))$

SQL

```
SELECT B.bid, B.bname FROM Boats B
WHERE NOT EXISTS
(SELECT S.sid FROM Sailors S

MINUS

SELECT R.sid FROM Reserves R
WHERE R.bid = B.bid)
```

Grading Notes:

- No SQL or no Algebra **(-5) points**
- SQL that is completely off, but in the general direction **(-4) points**
- Using cross product without renaming the fields with the same name **(-1) point**
- Division done backwards **(-1) point**
- Minor SQL spelling mistakes - **(-0.5) points**. Maximum one.
- Referring to a field in the algebra which does not exist or has been erased due to cross product. **(-1) point**.

(c) Show the name and rating of each sailor who never reserved boat number 11.

Relational Algebra –

$\pi_{(sname,rating)}(\pi_{(sid,sname,rating)}Sailors - (\pi_{(sid,sname,rating)}Sailors \bowtie (\sigma_{(bid=11)}Reserves)))$

SQL

```
SELECT S.sname, S.rating FROM Sailors S
WHERE S.sid NOT IN (
   (SELECT S2.sid FROM Sailors S2, Reserves R2
   WHERE S2.sid = R2.sid AND R2.bid = 11)
)
```

Grading Notes:

- Showing the sailors who reserved a boat other than 11. **(-5) points**. This was the most common mistake.
- Showing all sailors who reserved at least one boat and didn't reserve 11. **(-1) point**
- Minor SQL or algebra mistakes. **(-0.5) points**

2.2 Aggregate SQL Queries (15 points)

Write SQL for the following query:

(a) Show the boat color which was reserved the most (*e.g.* red, blue, etc.).

```
SELECT CC.color FROM
(SELECT B.color, count(*) as colorcnt FROM Reserves R, Boats B
WHERE R.bid = B.bid GROUP BY B.color) ColorCount CC
WHERE CC.colorcnt >= ALL
   (SELECT count(*) FROM Reserves R1, Boats B1
   WHERE R1.bid = B1.bid GROUP BY B1.color)
```

Grading Notes:

- Minor SQL syntax mistakes (maximum 1). **(-0.5) points**
- SQL with illegal group by actions (columns, illegal SELECT fields, etc.) **(-3) points**
- SQL which calculates color popularity, but doesn't use it properly. **(-10) points (maximum)**
- Some notion of using MAX and GROUP BY, but nothing else. **3 points**
- Some notion of MAX or GROUP BY (but not both). **2 points**

3 Triggers (15 points)

Consider the following modified sailors/reserves/boats schema:

Sailors (<u>sid:`INT`</u>, sname:`VARCHAR(30)`, rating:`INT`, age:`INT`)
Boats (<u>bid:`INT`</u>, bname:`VARCHAR(30)`, color:`VARCHAR(10)`)
Reserves (<u>bid:`INT`, sid:`INT`, day:`DATE`</u>)
Canceled (<u>sid:`INT`, bid:`INT`, day:`DATE`</u>, age:`INT`, color:`VARCHAR(10)`)

The relation "Canceled" contains information about all canceled reservations including: the sailor ID and age of the sailor who canceled, the boat ID and color of the boat, and the day of the canceled reservation. For example, if Horatio (sid = 11, age = 25) canceled a reservation for the boat Interlake (bid = 21, color = red) on January 26, 2011, the relation Canceled will store a new row:

sid	bid	day	age	color
11	21	26/1/2011	25	'red'

(a) (15 points) Write an MS SQL Server trigger which automatically updates "Canceled" whenever a reservation is **deleted** from Reserves. Make sure your trigger fires only when a row is deleted from Reserves (not on updates) and doesn't attempt to insert lines which would violate the primary key constraint of Canceled. If the deletion would cause such an illegal insert, the trigger should just complete successfully without performing any action (and without aborting the action). Assume only one row is deleted from Reserves at a time.

(b) (Extra Credit + 5 points) Make your MS SQL Server trigger work even when multiple rows are deleted from Reserves.

3.1 Answer

Here is a trigger which satisfies the requirements:

```
CREATE TRIGGER updateCancel ON Reserves AFTER DELETE AS
DECLARE @sailor INT;
DECLARE @boat INT;
DECLARE @day DATETIME;
DECLARE @already INT;
SELECT @sailor = (SELECT sid FROM deleted);
SELECT @boat = (SELECT bid FROM deleted);
SELECT @day = (SELECT day FROM deleted);
SELECT @already = (SELECT COUNT(*) FROM Canceled WHERE sid = @sailor AND bid = @boat
AND day = @day);

IF (@already = 0)
BEGIN
   INSERT INTO Canceled (sid, bid, day, age, color)
   SELECT DISTINCT d.sid, d.bid, d.day, S.age, B.color
   FROM Sailors S, deleted d, Boats B
   WHERE S.sid = d.sid AND B.bid = d.bid
END
```

To get it to work with multiple rows, a bit more complexity in the insert is required, but we actually use fewer lines:

```
CREATE TRIGGER updateCancel2 ON Reserves AFTER DELETE AS
BEGIN
   INSERT INTO Canceled (sid, bid, day, age, color)
   SELECT DISTINCT d.sid, d.bid, d.day, S.age, B.color
   FROM Sailors S, deleted d, Boats B
   WHERE S.sid = d.sid AND B.bid = d.bid AND NOT EXISTS (
        SELECT * FROM Canceled C
        WHERE C.sid = d.sid AND C.bid = d.bid AND C.day = d.day
    )
END
```

Grading Notes:

- Not checking if the row already exists in Canceled (not checking the primary key constraint). **(-5) points**
- No partial credit given on extra credit.
- Small MS SQL Server syntax errors. **(-0.5) points**
- Trigger with no inserts or impossible assignments. **(-15) points**
- Using *inserted* instead of *deleted*. **(-1) point**
- Trigger with some notion of inserting fields. **5 points**
- Extraneous and incorrect SQL. **(-2) points**

4 Transactions (10 points)

Consider the following transaction schedule:

T1	T2	T3
R(A)		
	R(A)	
	W(B)	
R(B)		
		R(B)
		W(D)
	W(D)	
R(D)		
		W(D)
Commit		
	Commit	
		Commit

Is the above schedule *view serializable*? If yes, to what serial schedule is it *view equivalent*? If not, list all of the conflicts which prevent it from being view serializable.

4.1 Answer

It is view serializable to the serial schedule T2; T1; T3.

Grading Notes:

- Correct answer (yes), but error in calculating the serializability order **(-6) points**
- Correct yes/no answer, but no justification. **(-10) points**

5 Appendix: SQL and MS SQL Server Formats

5.1 Basic SQL Queries:

```
SELECT [select-list]
FROM [table list]
WHERE [condition list]
GROUP BY [grouping-list]
HAVING [group-condition]
```

5.2 Basic SQL Data Manipulation:

Insert `INSERT INTO Table1(field1, field2) VALUES (val1, val2)`

Delete `DELETE FROM Table1 WHERE [condition]`

Update `UPDATE Table1 SET [field assignments] WHERE [condition-list]`

5.2.1 Another version for insert:

```
INSERT INTO Table1 (field1, field2)
SELECT f1, f2 FROM Table2
```

5.3 SQL Data Definition:

Table Creation `CREATE TABLE [table name] ([field] [type], ..., PRIMARY KEY ([field list]), FOREIGN KEY ([field list]) REFERENCES [table])`

View Creation `CREATE VIEW [view name] ([field list]) AS [body] [WITH CHECK OPTION]`

5.4 MS SQL Server Stored Procedures and Triggers:

Trigger format:

```
CREATE TRIGGER triggerName ON
   tableName | viewName
   FOR | AFTER | INSTEAD OF
   [INSERT,] [UPDATE,] [DELETE]
   AS [IF UPDATE (columnName)] [batch-code]
```

Stored Procedure format:

```
CREATE PROCEDURE procedureName [(@parameter1
datatype1 [=DEFAULT], [@parameter2 datatype2
[= DEFAULT], ...])]
   AS batch-code
```

5.5 Relational Algebra

The general mapping from relational algebra to SQL:

Algebra	SQL Term
$\pi_{x,y}$	SELECT x, y
$R \times S$	FROM R, S
$\sigma_{x>y}$	WHERE $x > y$
$R \bowtie_c S$	FROM R, S WHERE c
$R \bowtie S$	S NATURAL JOIN R
$\rho(N(p1 \rightarrow o_1, p2 \rightarrow o_2), E)$	SELECT p_1 AS o_1, p_2 AS o_2 FROM E AS New WHERE ...