CS 145 Introduction to Databases
Autumn Quarter 2015
Stanford University

Midterm Examination

Tuesday October 27, 2015
80 minutes

	Problem	Full Points	Bonus Points	Your Score
1	The Love of Money	20	10	
2	If It's All the Same to You...	15		
3	ER for the ER	15		
4	Where are my keys?	20		
5	Bob's Bagel Bite Shop	20		
6	Bonus: MVDs		10	
Total		90	20	

The exam contains **20** pages including this cover page. The last page provides you some information that may be useful in the exam. Feel free to tear the last page out.

Instructions:

- This exam is **closed book i.e. no laptops, notes, textbooks, etc. during the exam**
- In all cases, and especially if you're stuck or unsure of your answers, **explain your work!** We'll give partial credit for good explanations of what you were trying to do
- Keep the point values of questions in mind- don't get too stuck on any one part! **Our advice is to skip the bonus problems on your first pass!**

Name: _____

SUNETID: _____

I have read and will abide by the Stanford honor code:

Signature: _____

1 The Love of Money [20 points]

Imagine the following setup:

- Stocks have some positive or negative change in price each week
- Traders each pick a portfolio of stocks, i.e. they have one or more stocks and they own some non-zero quantity of each one

The database schema for this problem consists of two relations corresponding to the descriptions above:

```
Stock(ticker VARCHAR(3), price_change FLOAT, week INT)
Owns(trader_name TEXT, stock_ticker TEXT, quantity FLOAT)
```

Assume there are no NULL values present unless specified otherwise, and that the same stocks are present for all weeks in the database.

1.1 Part 1: Safe for Work [5 points]

Write a query which returns the ticker and price change for all stocks which had a positive price change in week 4:

```
SELECT ticker, price_change
FROM Stock
WHERE week = 4 AND price_change > 0;
```

1.2 Part 2: Portfolio View [5 points]

Recall that our schema is:

```
Stock(ticker VARCHAR(3), price_change FLOAT, week INT)
Owns(trader_name TEXT, stock_ticker TEXT, quantity FLOAT)
```

Now write a query which returns the ticker and *net price change* over all the weeks in the table, **for each stock**. Here, for example, if our table covered three weeks, and if stock ABC had price_change=-5.45 in week 1, and price_change=1.45 in week 2, and price_change=2.00 in week 3, the *net price change* would be $-5.45 + 1.45 + 2.00 = -2.00$, and we would return a row $(ABC, -2.00)$ for this stock.

```
SELECT ticker, SUM(price_change)
FROM Stock
GROUP BY ticker;
```

1.3 Part 3: Seeking Alpha [10 points]

Recall that our schema is:

```
Stock(ticker VARCHAR(3), price_change FLOAT, week INT)
Owns(trader_name TEXT, stock_ticker TEXT, quantity FLOAT)
```

Write a query which returns the name of each trader and **the total change in value of their portfolio** for week 7, **only for traders who had a positive total portfolio change**. Here, for example, if the stock ABC had price_change=2.00 in week 7, and the stock BXZ had price_change=-1.50 in week 7, and trader Joe had 3 of ABC and 10 of BXZ, then the total change in value of Joe's portfolio for week 7 would be $3 * 2 + 10 * (-1.5) = -9.00$. Note that in this example *we would not return a row for Joe since his total portfolio change was* ≤ 0.

```
SELECT trader_name, SUM(price_change * quantity)
FROM Stock, Owns
WHERE stock_ticker = ticker AND week = 7
GROUP BY trader_name
HAVING SUM(price_change * quantity) > 0;
```

1.4 *Bonus: Complex Correlated Query [10 points]*

Recall that our schema is:

```
Stock(ticker VARCHAR(3), price_change FLOAT, week INT)
Owns(trader_name TEXT, stock_ticker TEXT, quantity FLOAT)
```

No tricks or puns here: this question involves a correlated query! Write a query which returns the tickers of stocks that, in any given week, out-performed (or matched) all other stocks *which were not in any portfolio with them* that week:

```
SELECT s1.ticker
FROM Stock s1
WHERE NOT EXISTS (
  SELECT s2.ticker
  FROM Stock s2
  WHERE s1.week = s2.week
   AND s2.price_change > s1.price_change
   AND NOT EXISTS (
     SELECT o3.trader_name
     FROM Owns o3, Owns o4
     WHERE o3.trader_name = o4.trader_name
       AND o3.stock_ticker = s1.ticker
       AND o4.stock_ticker = s2.ticker));
```

Solution explanation: We step through each level of nested query:

1. We select all stocks s1 such that there does not exist...
2. ...any stock s2 which in the same week had a greater price change,
3. and such that no trader owns both of these stocks in the same portfolio

2 If It's All the Same to You... [15 points]

The parts of this problem are **multiple choice**. In each of the below questions, consider the three SQL queries listed, and circle **one** of the listed options. Assume there are no NULL values in any of the tables. The queries will all involve either IN, ALL, ANY (recall, equivalent to "SOME"), or EXISTS.

2.1 Part 1 [5 points]

```
SELECT S.A FROM S WHERE S.B NOT IN (SELECT R.B FROM R WHERE R.B > 5);

SELECT S.A FROM S WHERE S.B <> ALL (SELECT R.B FROM R WHERE R.B > 5);

SELECT S.A FROM S WHERE NOT EXISTS (SELECT R.B FROM R WHERE R.B > 5 AND R.B = S.B);
```

- **(A)** The first query is not equivalent to any of the other queries
- **(B)** The first query is equivalent to the *second* query only
- **(C)** The first query is equivalent to the *third* query only
- (**D**) All three queries are equivalent.

Solution: D

Explanations:

- The first query returns all $S.A$ such that $S.B$ is not equal to any of the Rs with $R.B > 5$
- The second query returns all $S.A$ such that for all Rs with $R.B > 5$, $S.B$ is not equal to $R.B$
- The third query returns all $S.A$ such that there are no cases where for Rs with $R.B > 5$, $S.B = R.B$

The three are all the same!

2.2 Part 2 [5 points]

```
SELECT S.A FROM S WHERE S.B NOT > ANY (SELECT R.B FROM R WHERE R.B > 5);

SELECT S.A FROM S WHERE S.B <= ALL (SELECT R.B FROM R WHERE R.B > 5);

SELECT S.A FROM S WHERE EXISTS (SELECT R.B FROM R WHERE R.B > 5 AND R.B >= S.B);
```

- **(A)** The first query is not equivalent to any of the other queries
- **(B)** The first query is equivalent to the *second* query only
- **(C)** The first query is equivalent to the *third* query only
- **(D)** All three queries are equivalent.

NOTE: The first query is not actually syntactically correct- we're very sorry about this! Everyone will receive +5 points here. However if interested, consider replacing "NOT >" with the more syntactically correct "<=", then the below solution will still hold.

Solution: C

Explanations:

- The first query returns all $S.A$ such that $S.B$ is not greater than (i.e. is less than) *at least one* of the Rs with $R.B > 5$
- The second query returns all $S.A$ such that $S.B$ is less than or equal to *all* Rs with $R.B > 5$, $S.B$ is not equal to $R.B$
- The third query returns all $S.A$ such that there exists some R where $R.B > 5$ and $S.B$ is less than $R.B$

Be careful of ANY!!! Try replacing it with "SOME" (which we note is equivalent) and see if it makes sense, and/or see the midterm review slides!

2.3 Part 3 [5 points]

```
SELECT S.A FROM S WHERE S.B IN (SELECT R.B FROM R WHERE R.B > 5);

SELECT S.A FROM S WHERE S.B = ANY (SELECT R.B FROM R WHERE R.B > 5);

SELECT S.A FROM S WHERE S.B >= ALL (SELECT R.B FROM R WHERE R.B > 5);
```

- **(A)** The first query is not equivalent to any of the other queries
- **(B)** The first query is equivalent to the *second* query only
- **(C)** The first query is equivalent to the *third* query only
- **(D)** All three queries are equivalent.

Solution: B

Explanations:

- The first query returns all $S.A$ such that $S.B$ is *in* (i.e. is equal to at least one of) the Rs with $R.B > 5$
- The second query returns all $S.A$ such that $S.B$ is equal to *at least one* of the Rs with $R.B > 5$
- The third query returns all $S.A$ such that $S.B \geq$ all $R.B$ such that $R.B > 5$

3 ER for the ER [15 points]

Design an ER diagram for a hospital that contains the following kinds of objects together with the listed attributes:

- Specialty(name, motto, priority_level)
- Room(room_ID, location, capacity)
- Team(name)
- Doctor(name, SSN, age, sex)

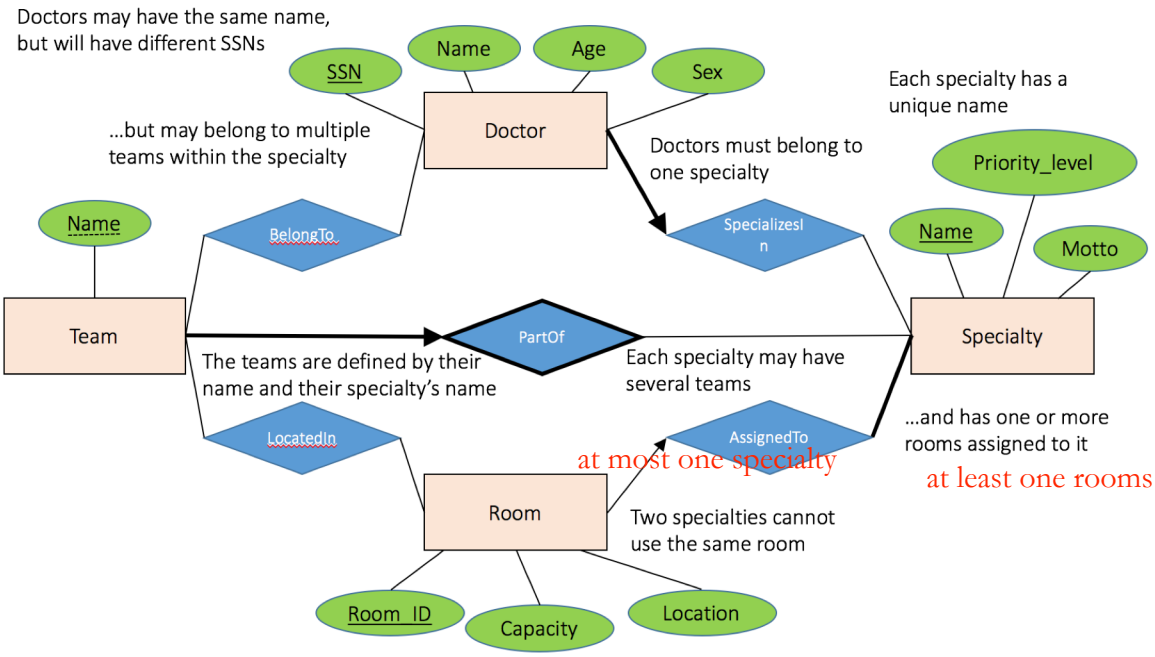
In addition the following must be modeled:

- Each specialty has a unique name (e.g. "cardiology", "neurology", etc.), a motto (for example, rumor has it the Stanford urology group's is "the stream team"), a priority_level, and **has one or more rooms assigned to it**
- Two specialties cannot use the same room
- Each specialty may have several **teams**, which can be spread across various rooms (i.e. more than one team from the same specialty can be in the same room). The teams are **defined by their name and their specialty's name**, i.e. "Urology Team 6" is the team with name "Team 6" that belongs to the "Urology" specialty.
- Doctors must **belong to one specialty but may belong to multiple teams within the specialty**
- Doctors may have the same name, but will have different SSNs

Be sure to specify **keys, multiplicity, and participation constraints** in addition to the ER basics (entity sets, relationships and attributes). If in doubt, just explain what you're trying to do!

Solution on next page...

Extra space:


Doctors may have the same name, but will have different SSNs
SSN
Name
Age
Sex
Doctor
...but may belong to multiple teams within the specialty
Doctors must belong to one specialty
Each specialty has a unique name
Priority_level
Name
Motto
Specializes In
Name
BelongTo
Team
PartOf
Specialty
The teams are defined by their name and their specialty's name
Each specialty may have several teams
...and has one or more rooms assigned to it
LocatedIn
AssignedTo
at most one specialty
at least one rooms
Room
Two specialties cannot use the same room
Room_ID
Capacity
Location

4 Where are my keys? [20 points]

For the first three parts of this problem, consider the relation $R(U, V, W, X, Y, Z)$ with functional dependencies:

1. $\{W, Y\} \rightarrow \{Z\}$
2. $\{U\} \rightarrow \{X, Y\}$
3. $\{V, X\} \rightarrow \{W\}$
4. $\{Y\} \rightarrow \{V\}$

For all parts of this question: keep in mind that explaining your reasoning / showing your work, while not required, is extremely helpful if we are trying to give you partial credit!

4.1 Part 1 [5 points]

Consider the instance of relation R below. Fill in the blanks so that the instance is consistent with the FDs above. When the FDs imply a specific value for a blank space, fill in that value, otherwise if they do not imply a specific value, fill it in with a question mark:

U	V	W	X	Y	Z
0	0	1	2	3	4
1	0		2	2	
	0	1		3	4
2		1	2	2	5

Solution (one of potentially several?):

U	V	W	X	Y	Z
0	0	1	2	3	4
1	0	**1**	2	2	**5**
?	0	1	**?**	3	4
2	**0**	1	2	2	5

4.2 Part 2 [5 points]

Find the closure of $\{X, Y\}$, $\{X, Y\}^+$:

$$\begin{aligned} \{X, Y\}^+ &= \{X, Y, V\} && \text{FD \#4} \\ &= \{X, Y, V, W\} && \text{FD \#3} \\ &= \{X, Y, V, W, Z\} && \text{FD \#1} \end{aligned}$$

4.3 Part 3 [5 points]

List **one** key implied by the given FDs, and **explain why it is a key based on the definition:**

Solution: $\{U\}$

$$\begin{aligned} \{U\}^+ &= \{U, X, Y\} && \text{FD \#2} \\ &= \{U, V, W, X, Y, Z\} && \text{Previous part} \end{aligned}$$

Note that since $\{U\}$ is a key, we know that no other sets of attributes with $\{U\}$ in them will be a key, according to the definition of a key. Since no other attribute sets imply U, there are no other keys.

4.4 Part 4 [5 points]

For this problem, consider the relation $T(A, B, C, D, E)$ with the following functional dependencies:

1. $\{A\} \rightarrow \{B, C\}$
2. $\{C\} \rightarrow \{B\}$
3. $\{B\} \rightarrow \{D\}$
4. $\{E\} \rightarrow \{C\}$

Decompose T into two or more tables so that it is in BCNF, and briefly justify that your end result is indeed in BCNF:

Solution (one of several possible):

1. First, decompose along $\{A\}^+ = \{A, B, C, D\}$:
   - $T_1(A, B, C, D)$
   - $T_2(A, E)$
2. Next, decompose T_1 along $\{C\}^+ = \{B, C, D\}$:
   - $T_{11}(B, C, D)$
   - $T_{12}(A, C)$
3. Finally, decompose T_{11} along $\{B\}^+ = \{D\}$:
   - $T_{111}(B, C)$
   - $T_{112}(B, D)$

So our final decomposed tables are: $T_2(A, E), T_{12}(A, C), T_{111}(B, C), T_{112}(B, D)$.

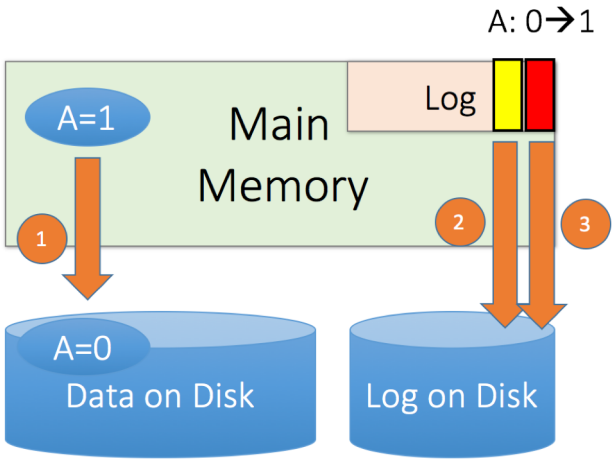
5 Bob's Bagel Bite Shop [20 points]

Bob's Bagel Bite Shop, the premiere purveyor of homemade bagel bites in the bay area, has decided to start a loyalty points program, where customers can earn *"bitecoins"*. Chef Bob is a big fan of transactions and upholding ACID guarantees, but to save on money, he decides to skip buying a full-featured SQL DBMS and instead implement everything himself...

[This space left intentionally blank... draw your favorite animal eating fruit here?]

The solution was polar bear eating watermelon:

5.1 Part 1: Write Mostly Ahead? [10 points]



Bob implements the following three procedures, but isn't quite sure how to order them:

1. The values in main memory written by the TXN are flushed to disk
2. The log of the TXN's actions is flushed to disk
3. The commit record (indicating that the TXN was committed) is flushed to disk

5.1.1 [3 points]

Consider the following order: 1,3,2. List the points at which a system crash would cause issues, and explain why, referring to ACID terms (one sentence each):

Solution: If the system crashed during or after flushing the TXN's values from main memory to disk [1] (*Note: we required that you included the after scenario as well for full credit*), the system would not know if the TXN had committed and should persist, or if it had not and should be undone- nor could it undo the writes, because no log would have been preserved. So this would prevent atomicity. After the commit record had been pushed [3], the system would know the data on disk should persist (durability), and there would be no issues.

5.1.2 [3 points]

Consider the following order: 2,1,3. List the points at which a system crash would cause issues, and explain why, referring to ACID terms (one sentence each):

Solution: If the system crashed during or after flushing the log's values from main memory to disk [1] (*Note: we required that you included the after scenario as well for full credit*), the system would not know if the TXN had committed and should persist- in which case the values on disk could be inferred from the log- or if the TXN had not committed; thus atomicity is not preserved.

If the system crashed after writing data to disk [1], we would have the same exact problem of not knowing if the data had committed!

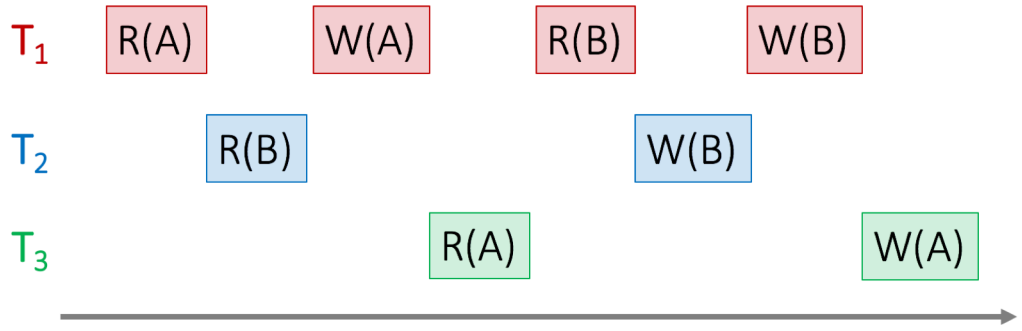
5.1.3 [4 points]

What is the best order, and why? Again, refer to ACID terms, and explain with one sentence.

Solution: **2,3,1**: WAL! (3,2,1, with reasonable explanation, was given partial credit)

5.2 Part 2: Conflicting Bites [10 points]

Bob also does not have any sort of locking, and must interleave transactions by hand. Consider the time-line below, showing three TXNs operating over two bitecoin accounts, A and B:



5.2.1 [5 points]

What are the conflicts here? List *conflicting* pairs of actions:

Solution:

- $T_1 : R(A)$, $T_3 : W(A)$
- $T_2 : R(B)$, $T_1 : W(B)$
- $T_1 : W(A)$, $T_3 : R(A)$
- $T_1 : W(A)$, $T_3 : W(A)$
- $T_1 : R(B)$, $T_2 : W(B)$
- $T_2 : W(B)$, $T_1 : W(B)$

5.2.2 [5 points]

Describe the serial order this schedule corresponds to. If this schedule is *not serializable*, first circle two operations whose horizontal position could be swapped to make this schedule serializable.

Solution: The schedule is not serializable. However, you can swap $T_1 : R(B)$ and $T_2 : W(B)$, and then it will correspond to the serial order T_2, T_1, T_3. There are other possible correct swaps as well. the schedule

6 BONUS: MVDs [10 points]

Wow. MVDs. Whaaaaat? Who knew these would be on here. Well, sometimes life throws you for a curve...

Consider the instance of relation R below. Fill in the blanks so that the instance is consistent with the following MVDs:

- $\{U\} \twoheadrightarrow \{W, Y\}$
- $\{V\} \twoheadrightarrow \{X, Z\}$

When the MVDs do not imply a specific value for a blank space, fill it in with a question mark:

U	V	W	X	Y	Z
0		2	3		5
	2		4	5	6
0	2	2	4	4	6
0	1	3	3	5	

Solution:

U	V	W	X	Y	Z
0	**1**	2	3	**4**	5
0	2	**3**	4	5	6
0	2	2	4	4	6
0	1	3	3	5	**5**

7 Possibly Useful Information

- **Canonical SQL Statement:**

```
SELECT <attributes>
FROM <tables>
WHERE <conditions>
GROUP BY <attributes>
HAVING <conditions>
```

- **Functional Dependency (FD):** For a relation R, and sets of attributes X and Y, the functional dependency $X \to Y$ holds if for any $t_1, t_2 \in R$, $t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$.

- **Armstrong's Axioms:** Let the A_is, B_js, and C_ks be attributes:

  1. *Split/Combine:* If $A_1, ..., A_n \to B_j$ for $j = 1, ..., m$, then this is equivalent to $A_1, ..., A_n \to B_1, ..., B_m$ and vice-versa
  2. *Reduction/Trivial:* $A_1, ..., A_n \to A_i$ for any $i = 1, ..., n$
  3. *Transitive Closure:* If $A_1, ..., A_n \to B_1, ..., B_m$ and $B_1, ..., B_m \to C_1, ..., C_p$ then $A_1, ..., A_n \to C_1, ..., C_p$

- **Closure:** Given a set of attributes X and a set of FDs F, the closure X^+ is the set of all attributes y such that $X \to y$.

- **Superkey:** Given a relation R, a superkey is a set of attributes X such that X^+ is equal to the full set of attributes of R.

- **Key:** A key is a minimal superkey, i.e. a superkey where no subset of it is also a superkey.

- **Boyce-Codd Normal Form (BCNF):** A relation R is in BCNF if for all sets of attributes X, either $X^+ = X$ (X is trivial) or $X^+ =$ the set of all attributes (X is a superkey).

- **Conflicts:** Two actions conflict if they are part of different TXNs, involve the same variable, and at least one of them is a write.

- **Serializable:** A schedule is serializable if it is equivalent to some serial ordering.

- **Multi-Value Dependency (MVD):** Given a relation R with a set of attributes A, two sets of attributes $X, Y \subseteq A$, we say that the MVD $X \twoheadrightarrow Y$ holds if for any tuples $t_1, t_2 \in R$ such that $t_1[X] = t_2[X]$, there is a tuple t_3 such that:
  - $t_3[X] = t_1[X]$
  - $t_3[Y] = t_1[Y]$
  - $t_3[A \setminus Y] = t_2[A \setminus Y]$