CS 122A/EECS 116: Introduction to Data Management
Spring 2016, UC Irvine
Prof. Chen Li
Final Exam (Max. Points: 100)

Sample Solutions

Instructions:

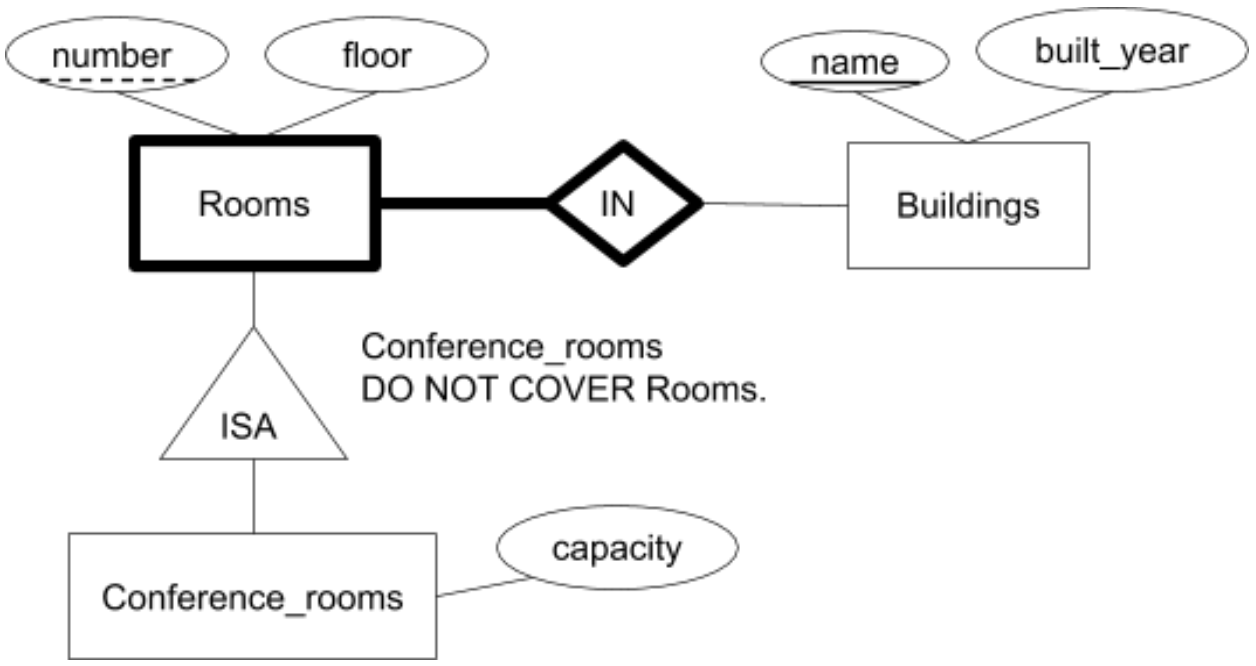
- The total time for the exam is 120 minutes. Be sure to budget your time accordingly.
- The exam is closed book and closed notes.
- Be sure to answer every part of each question.
- If you don't understand something, ask an instructor/TA for clarification.
- If you still find ambiguities in a question, note the (reasonable) interpretation you are taking.

NAME: SOLUTION	**STUDENT ID: SOLUTION**

QUESTION	POINTS	TOPIC	SCORE
1	**10**	ER Diagram	10
2	**15**	SQL I	15
3	**20**	SQL II	20
4	**15**	Functional dependencies	15
5	**20**	Normal forms	20
6	**20**	Indexing	20
TOTAL	**100**		100

Question 1: E-R to Relation (10 points)

Refer to the following E-R diagram for a database that keeps track of buildings, rooms, and in particular, conference rooms.



(a) **(2 pts)** For each of the following statements, specify TRUE or FALSE based on the constraints specified in the E-R diagram above.

I. The number of entities in the Rooms entity set must be greater than or equal to the number of entities in the Conference_rooms entity set.

([✓ **True**] / False)

II. The number of entities in the Rooms entity set must be greater than or equal to the number of entities in the Buildings entity set.

(True / [✓ **False**])

SCORE: _________

(b) **(8 pts)** Construct SQL DDL statements to create tables for the above E-R diagram. Be sure to create the columns using appropriate types. Also specify primary keys, foreign keys, NOT NULL constraints, and their DELETE policies.

```
CREATE TABLE Buildings (
        name VARCHAR(20) NOT NULL,
        built_year INTEGER,
        PRIMARY KEY name);

CREATE TABLE Rooms (
        name VARCHAR(20) NOT NULL,
        number INTEGER NOT NULL,
        floor INTEGER NOT NULL,
        PRIMARY KEY (name, number),
        FOREIGN KEY (name) REFERENCES Buildings(name) ON DELETE CASCADE);

CREATE TABLE Conference_rooms (
        name VARCHAR(20) NOT NULL,
        number INTEGER NOT NULL,
        capacity INTEGER NOT NULL,
        PRIMARY KEY (name, number),
        FOREIGN KEY (name, number) REFERENCES
                                Rooms(name, number) ON DELETE CASCADE);
```

SCORE: _________

Question 2 (15 points): SQL (I)

Consider having a database called 'Company' with the following schema:

```
CREATE TABLE Department (
        did       INTEGER NOT NULL,
        dname   VARCHAR(50) NOT NULL,
        PRIMARY KEY(did) );

CREATE TABLE Employee (
        eid       INTEGER NOT NULL,
        name    VARCHAR(50) NOT NULL,
        age       INTEGER,
        did       INTEGER NOT NULL,
        PRIMARY KEY(eid),
        FOREIGN KEY(did) REFERENCES Department(did) );
```

We have the following records in the database (no other records):

Department

did	dname
1	'HR'
2	'IT'
3	'Maintenance'

Employee

eid	name	age	did
10	'Barney'	24	2
11	'John'	39	3
12	'Amy'	55	2

(a) **(7 pts)** According to the above tables, for each of the following SQL statements, circle "True" if the statement can be executed **without violating** the constraints, and "False" otherwise. **Each statement is executed independently.**

1. INSERT INTO Employee VALUES (1, 'Ben', 2); (True / [✓ False])
2. INSERT INTO Employee VALUES (2, NULL, NULL, 1); (True / [✓ False])
3. INSERT INTO Employee VALUES (3, 'Holly', 20, 3); ([✓ True] / False)
4. INSERT INTO Employee VALUES (5, 'Mary', 44, 4); (True / [✓ False])
5. DELETE FROM Department WHERE dname = 'HR'; ([✓ True] / False)
6. DELETE FROM Department WHERE did = 2; (True / [✓ False])
7. DELETE FROM Employee WHERE eid = 12; ([✓ True] / False)

SCORE: _________

(b) **(2 pts)** We have created a user called 'Supervisor'. Write a SQL statement to grant him the privilege to update names of departments.

GRANT UPDATE(dname) ON Department to 'Supervisor';

(c) **(3 pts)** Write a SQL statement grant the ‘Supervisor’ the privilege to insert new employee records and give other users this privilege.

GRANT INSERT ON Employee to 'Supervisor' WITH GRANT OPTION;

(d) **(3 pts)** Create a view named 'IT_Employees' to include the names of employees working for the ‘IT’ department.

CREATE VIEW 'IT_Employees' AS
SELECT dname, name
FROM Department NATURAL JOIN Employee
WHERE dname = ‘IT’;

Question 3 (20 pts): SQL (II)

Consider the following database schema:

- **Company** (cid, cname, year_founded)
- **Worker** (wid, wname, age, phone_number, cid)
- **Project** (pid, pname, budget)
- **Projects_Has_Workers** (pid, wid, working_hours, pay)

Suppose that each company can have more than one worker. Workers work only for one company and can be assigned to different projects. Each project can have multiple workers from different companies working on that project. For each of the following tasks, write a single SQL statement.

(a) **(3 pts)** For each project, show its id and number of ***unique*** companies working on the project.
SELECT pid, count(distinct cid)

FROM COMPANY NATURAL JOIN WORKER NATURAL JOIN Projects_Has_Workers

GROUP BY pid

SCORE: _________

(b) **(4 pts)** For each project that still has available fund (i.e., its budget is greater than the total pay of its workers), show its id and name.

```
SELECT p.pid, p.pname FROM Project p, (SELECT pid, SUM(pay) as total FROM
Projects_Has_Workers GROUP BY pid) as t WHERE p.pid = t.pid AND p.budget > t.total;
```

(c) **(4 pts)** For each worker who has worked for more than 200 hours (total) in projects, show his/her id, name, age, and phone number.

```
SELECT w.wid, w.wname, w.age, w.phone_number
FROM Worker as w,      (SELECT wid, SUM(working_hours) as total
                       FROM Projects_Has_Workers
                       GROUP BY wid HAVING total > 200) as p
WHERE p.wid = w.wid;
```

(d) **(4 pts)** For each worker, show his/her id, name, age, and phone number, along with the number of projects and average pay of those projects he/she works on. A worker should be included even if he/she is not working on any project.

```
SELECT w.wid, w.wname, w.age, w.phone_number, count(p.pid), avg(p.pay)
FROM Worker w LEFT OUTER JOIN Projects_Has_Workers p on w.wid = p.wid
GROUP BY w.wid;
```

(e) **(5 pts)** Create a TRIGGER named 'Update_Pay' that will update attribute 'pay' before a tuple in Projects_Has_Workers is updated (for each tuple). The 'pay' attribute value is calculated as 'working_hours' * 500.

```
DELIMITER $$
CREATE TRIGGER Update_Pay
BEFORE UPDATE ON Projects_Has_Workers
FOR EACH ROW
        BEGIN
                SET NEW.pay = NEW.working_hours * 500;
        END$$
DELIMITER ;
```

SCORE: _________

Question 4 (15 points): Functional Dependencies

(a) **(10 pts)** For a relation with attributes ABCD, we have the following instance of the table.

A	B	C	D
6	4	2	9
6	7	5	9
3	4	5	3

For each of the functional dependencies (FDs) below, specify "**Violated (V)**" or "**Not Violated(NV)**" to indicate whether it is **violated** by the instance, and briefly explain the reason by mentioning the violating tuple values.

(1) $A \rightarrow D$

Answer:_**NV**_____ Explain:____

(2) $B \rightarrow C$

Answer:_**V**_____ Explain: **(4, 2), (4, 5)**

(3) $AD \rightarrow B$

Answer:__**V**____ Explain: **(6 9, 4), (6 9, 7)**_

(4) $AB \rightarrow C$

Answer:__**NV**____ Explain:____

(5) $C \rightarrow D$

Answer:__**V**____ Explain: **(5, 9), (5, 3)**

(b) **(5 pts)** Given a relation R(ABCDEFG) with a set of FDs = {$A \rightarrow BD$, $BC \rightarrow E$, $D \rightarrow G$, $GE \rightarrow F$}, prove $AC \rightarrow F$ is also true. If you are using the Armstrong's axioms, make sure to specify the axiom used in this step.

$A \rightarrow D$, $D \rightarrow G$ => $A \rightarrow G$

$A \rightarrow B$ => $AC \rightarrow BC \rightarrow E$ => $AC \rightarrow E$ => $AC \rightarrow EG$ => $AC \rightarrow F$

OR:

$A \rightarrow BD$ => $AC \rightarrow BDE$ => $AC \rightarrow BDGE$ => $AC \rightarrow BDGEF$ => $AC \rightarrow F$

SCORE: _________

Question 5 (20 points): Normal Forms

For each of the following questions, we are given a relational schema and a set of functional dependencies (FDs). Do the following: (i) Identify the relation's candidate key(s); (ii) specify the highest normal form (among 1NF, 2NF, 3NF, and BCNF); and (iii) give the FD(s) that violate(s) the next-level normal form. If it's in BCNF, then there is no need to give such a violation; (iv) check whether a given decomposition is lossless join and dependency preserving (Yes/No). Explain each answer briefly.

(1) **(8 pts)** R = ABCD, FDs = {A → B, BC → A}
(i) Candidate key(s):

Answer: ACD, BCD

Explanation:

(ii) Highest normal form and a **single** FD that violates the next-level normal form:

Answer: 3NF

Explanation: A → B, or BC → A

(iii) Suppose we decompose the relation into two relations, R1(ABC) and R2(BCD). Is this decomposition lossless join? Dependency preserving? Explain why.

Lossless Join (Yes/No): YES

Explanation: "BC" are shared by the two relations, and they form a key in R1(ABC).

Dependency Preserving (Yes/No): YES

Explanation: All FDs are still true in R1(ABC).

SCORE: _________

(2) **(12 pts)** R = ABCDEFGH, FDs = $\{A \rightarrow B, A \rightarrow C, A \rightarrow D, AE \rightarrow H, E \rightarrow D, E \rightarrow F, E \rightarrow G\}$

(i) Candidate key(s):

Answer: AE

Explanation:

(ii) Highest normal form and a **single** FD that violates the next-level normal form.

Answer: 1NF

Explanation: $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow D$, $E \rightarrow D$, $E \rightarrow F$, $E \rightarrow G$

(iii) Suppose we decompose the relation into two relations, R1(ABCD) and R2(DEFGH).
Is this decomposition lossless join? Dependency preserving? Explain why.

Lossless Join (Yes/No): No

Explanation: $(ABCD) \cap (DEFGH) = D$ => Lossless join is violated

Dependency Preserving No

Explanation: $AE \rightarrow H$ is not preserved

SCORE: ________

Question 6 (20 pts): Indexing

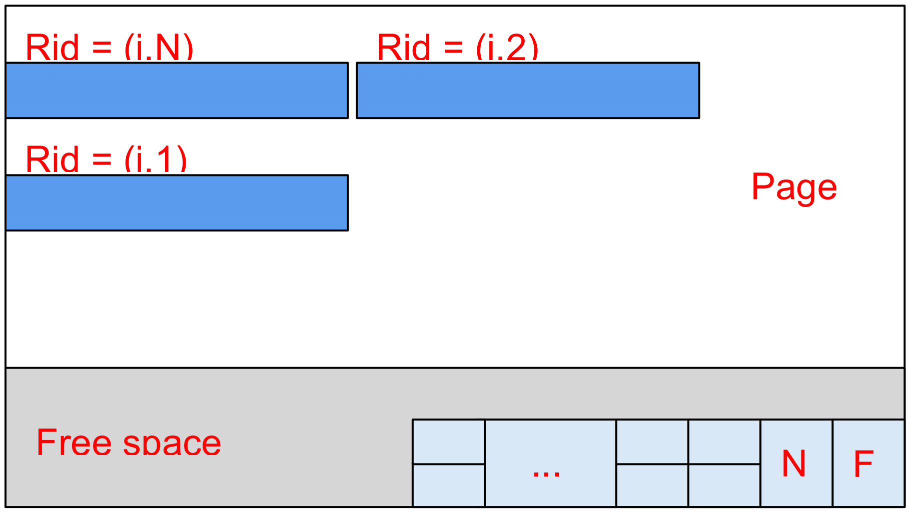
We have a relational table

Author(aid, name, university, publication_count)

For example, a tuple (123, 'John Doe', 'UCI', 356) means that 'John Doe' with an id 123 at 'UCI' has 356 publications.

The table consists of 1 million records. Each record in the table is variable-length. The average size of one record is 100 bytes. A page size on disk is 4KB (4,096 bytes). A record cannot span across two pages.

(a) **(3 pts)** Draw a diagram to show the structure of a page to store records. Specify the format of a record id ("rid"). Also explain a main advantage of this format.



Rid: (page#, index in the directory – slot#). A record can be moved within page without changing RIDs.

(b) **(2 pts)** We store the records in pages. Suppose each page is full of records. Roughly how many pages are in the file for the table?
The number of tuples per page: floor(4096 / 100) = 40.
The number of total pages: 1,000,000 tuples / 40 tuples per page = 25,000 pages

(c) **(3 pts)** We occasionally execute the following query to collect the average number of publications per university:

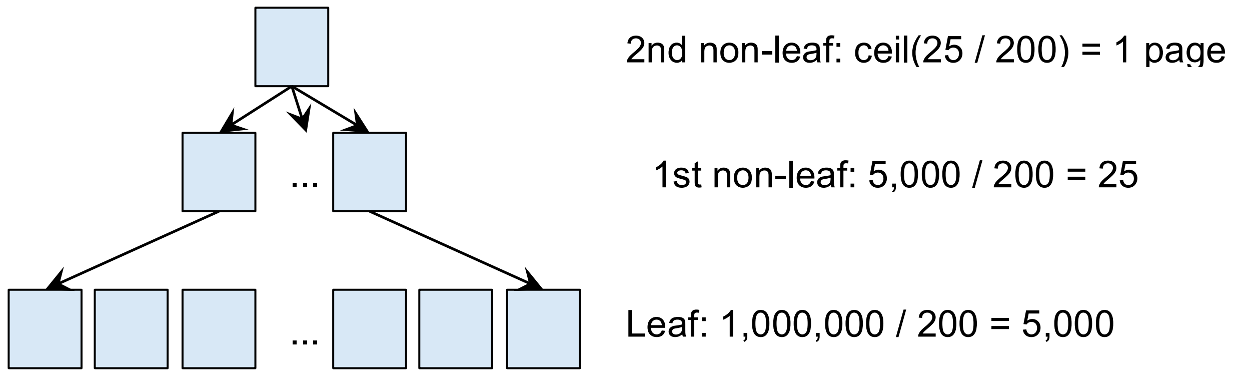
SELECT AVG(publication_count) FROM Author GROUP BY university;

The query takes a long time to finish, so we want to create **a clustered B+ tree index** on **only one attribute** to speedup the query. Which attribute will you choose? Why?

Attribute: university
Reason: The average will be calculated per university. Since this is a clustered B+ tree index on "university", tuples belong to the same university will be placed next to each other. So, it's easy to calculate avg(publication_account) per university. Creating an index on number_publication will not help.

SCORE: __________

(d) **(5 pts)** Suppose we build an unclustered B+ tree index on the "Publication_Count" attribute for the record storage structure described in (b), and each index page is fully occupied. Each index key pair in the leaf pages consists of <publication_count, rid>. Assume that each leaf index page contains 200 such pairs. Each non-leaf index page contains 200 pairs of publication_count and pointer to one of its children nodes. Calculate the number of levels of the B+ tree. Show the details of your calculation. Make sure to draw a diagram to explain the shape of the B+ tree.
There are 1 million <publication_count, rid> pairs in the leaf nodes of B+ Tree and each leaf page can accommodate 200 such pairs.
The number of leaf pages in this B+ Tree is: ceil(1,000,000 / 200) = 5,000 pages.
The number of pages in the first non-leaf level is: ceil(5000 / 200) = 25 pages.
The number of pages in the second non-leaf level is: ceil(25 / 200) = 1. This is the root node.
So, the number of level of this B+ tree is 3.


2nd non-leaf: ceil(25 / 200) = 1 page
1st non-leaf: 5,000 / 200 = 25
Leaf: 1,000,000 / 200 = 5,000

(e) **(4 pts)** Based on (d), calculate the number of disk I/Os for the following query, which is assumed to return 10,000 records. Also assume initially no B+ tree pages are in memory.

SELECT name, university, publication_count FROM Author
WHERE publication_count >= 100;

This query accesses about 50 leaf pages (10,000 / 200 = 50). (If the first pair starts in the middle of the first page, the number of leaf nodes that need to be read is 51.) And in order to access the first leaf page that satisfies the given predicate, we need to have two additional disk I/Os (root -> one of its children that has a leaf page that contains the first <100, rid> pair). So, there will be 52 disk I/Os to read in this index.
In addition to this cost, the query fetches all records that satisfy the condition. So, we need to access the table file. Each record access requires one disk I/O since this is an unclustered index.
Therefore, the total I/O cost is 52 + 10,000 = 10,052

(f) **(3 pts)** Consider the following query:

SELECT publication_count, COUNT(*) FROM Author
WHERE publication_count > 30
GROUP BY publication_count;

Suppose we have an index on the "Publication_count" attribute. Does this query have an index-only plan? Why?

Is this an index-only plan query? (Circle one) [✓ **YES**] / NO
Reason: Yes. We can only use index on the Publication_count attribute to answer this query. No record access is required.

SCORE: _________