Endterm Exam (Version B)
CS 122A
Spring 2019
Max. Points: 100
(Please read the instructions carefully)

Instructions:
- The total time for the exam is 50 minutes; be sure to budget your time accordingly.
- The exam is closed book and closed notes but “open cheat sheet”.
- Read each question first, in its entirety, and then carefully answer each part of the question.
- If you don’t understand something, ask one of the exam proctors for clarification.
- If you still find ambiguities in a question, note the interpretation you are taking.
- Please tear off the last page of the exam for reference (and for use as scratch paper).

<table>
<thead>
<tr>
<th>QUESTION</th>
<th>TOPIC</th>
<th>POINTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACIDity</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>Index Basics</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>B+ Trees</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Physical Design</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>NoSQL</td>
<td>25</td>
</tr>
<tr>
<td>TOTAL</td>
<td>All</td>
<td>100</td>
</tr>
</tbody>
</table>
Question 1: ACIDity (10 points)

Refer to the reference data at the end of the exam and look at the following SQL transactions that users might wish to run against it. (Note: SQL DBMSs like MySQL run each SQL statement as a separate transaction when there are no explicit TRANSACTION control statements present.)

**Program P1:**
```
START TRANSACTION;
    INSERT INTO Hobbies VALUES (102, 'soccer');
    INSERT INTO Hobbies VALUES (102, 'tai chi');
COMMIT;
```

**Query Q1:**
```
SELECT COUNT(*) FROM Hobbies
WHERE user = 102;
```

(a) (2 pts) Suppose that you were to run P1 and Q1 concurrently in different windows on your laptop, and suppose that you had wisely chosen **SERIALIZABLE** as the isolation level to run them at when launching this pair of activities. Circle all of the possible numerical outcomes from query Q1 (depending on how Q1’s execution happens to interleave with T1):

```
0 1 2 3 4 5 6
```

(b) (3 pts) Now suppose that you had instead unwisely chosen to configure your database system’s isolation level, probably for performance reasons, to be **READ UNCOMMITTED**. Circle all of the possible numerical outcomes from query Q1 under these less ideal operating conditions:

```
0 1 2 3 4 5 6
```

(c) (1 pt) Suppose that you ran program P1 all by itself but then you accidentally powered your laptop off in the middle of its execution. Which one of the following internal database mechanisms would prevent the DBMS from leaving your program and database in a half-finished state? (Circle your answer.)

- Write-ahead logging (WAL)
- Two-phase locking (2PL)

(v) (4 pts) The acronym of choice for describing what transactions provide to database users is **ACID**. Circle the four terms below that together make up this extremely important acronym.

```
Data  Isolation  Completeness  Atomicity
Implicit  Access  Dynamic  Conflict
Consistency  Indexing  Disk  Accountability
Durability  Absolute  Intelligence  Compatibility
```
Question 2: Index Basics (15 points)

(1 pt each) For each of the following statements, circle the appropriate answer (TRUE or FALSE).

- In real database systems, it is common for a non-leaf index node to contain between 2 and 4 keys.
  TRUE  FALSE

- If a table has more than two indexes defined on it, the third index is called its tertiary index.
  TRUE  FALSE

- In ISAM, the number of page accesses required for an exact-match search, starting from the root and ending when the record of interest has been identified, is the same for all of the indexed records.
  TRUE  FALSE

- In a B+ tree, the number of page accesses required for an exact-match search, starting from the root and ending when the record of interest has been identified, is between $d$ and $2d$.
  TRUE  FALSE

- A major advantage of a hashed index over a tree-based index is that it is possible in a hashed index to determine at which index leaf page (a.k.a. primary bucket page) to begin an exact-match search without performing any actual I/O.
  TRUE  FALSE

- The cost of a disk I/O is about 100 times greater than the cost of a memory access.
  TRUE  FALSE

- A table can have at most one unclustered index.
  TRUE  FALSE

- The root node of a B+ tree must have between $d$ and $2d$ entries.
  TRUE  FALSE

- To search within a B+ tree page, the DBMS must first read the page into the buffer pool in memory.
  TRUE  FALSE

- The largest cost component of a random disk read is the seek time (disk head positioning time).
  TRUE  FALSE

- In a database system (like MySQL or AsterixDB) that stores its data records in the leaves of the table’s or dataset’s primary index, all secondary indexes must be unclustered indexes.
  TRUE  FALSE
Question 2: Index Basics (continued)

- A hashed index is more generally useful, in terms of the kinds of query predicates that it can be used to accelerate, than a tree-based index.

  TRUE  FALSE

- If a secondary index is built on a composite key such as (lastname, firstname), the order of the field list for the key doesn’t matter.

  TRUE  FALSE

- The *’s on the keys in the leaf pages of a picture of a primary index denote the fact that there may be multiple records associated with each of these key values.

  TRUE  FALSE

- A hashed index can be a clustered index.

  TRUE  FALSE
Question 3: B+ Trees (25 points)

Consider the following secondary B+ Tree index on Users(age) on the reference data at the end of the exam. (Apparently 12 other users have been added to the Users table since the content shown in the reference data.) The index pages are labeled for use in the questions that follow. As you answer the questions, base your answers on the algorithms (policies) recommended in the lectures and textbook.

(a) (8 pts) AgeIdx resides on disk and it hasn’t been accessed recently. Indicate below which pages the DBMS will have to read from disk, in which order, to run the SQL query `SELECT * FROM Users WHERE age >= 34 AND age <= 54`. To indicate the order, put a “1” by the first page that it will read, a “2” by the next page, and so on. Leave the space next to any untouched page blank.

P1 __ P2 __ P3 __ P4 __ P5 __ P6 __

(b) (8 pts) Suppose that the user’s next operation is `DELETE FROM Users WHERE firstname = ‘Stevie’ AND lastname = ‘Knicks’`. (Refer to the Users reference data to learn more about the target user.) Fill in the information that’s missing in the tree snippet below to show the effects of this operation on AgeIdx.

(Note: P4 and P5 are unaffected for sure.)

(c) (8 pts) Starting from the original index above, suppose the user’s first update operation was instead `INSERT INTO Users VALUES (107, ‘Will’, ‘Smith’, 75, ‘beware the flashy thing’)`. Show below what the root page of the resulting index will look like after this operation has been performed. Be sure to include the relevant key values, page pointers/numbers, and the level in your picture. (Assume that disk pages are allocated in sequence, so the first new page allocated for AgeIdx will be P7, the next will be P8, and so on. Assume that if a page splits, the old disk page is reused and a new page is allocated due to the split.)

(d) (1 pt) What is the order d of AgeIdx?

d = __ 2 ___
Question 4: Physical Design (25 points)

Consider the reference database on the last page of the exam (also summarized below).

Users (uid, firstname, lastname, age, motto)
Likes (user1, user2)
Chirps (mid, user, respto, sent, text)
Hobbies (user, hobby)

Assume the relational DBMS that has been chosen to hold the application’s data stores its records in files (one such file per table) and does best-effort clustering based on the table’s clustered index (if any) when adding records. (This is similar to what IBM’s DB2 system does.) Thus, the index entries in this system are RID-based. Also assume that the chosen system has both hash-based and B+ tree based indexes, so you have lots of options available. For joins, assume that the system always uses nested-loop index joins to compute the required joins. Finally, assume that the system leaves it to the DBA – that would be you! – to decide what fields to define indexes on. Assume that the system creates no indexes by default, so if you want an index, even one on a table’s primary key, it’s on you to create it.

The following questions give an SQL query template and a place to suggest and describe from zero to three indexes that should help to accelerate the query’s execution. Answer each question in isolation; your choices must help with that question’s query. (Also, be careful not to pick a clustered index for a query if clustering won’t help – don’t waste clustering! However, you may consider each problem in isolation – it is okay if you see non-wasted needs here for different clustering – which one to pick would be sorted out later based on the relative frequencies of the different queries in the workload.) Lastly, when you see value, that stands for a parameter that the web application will fill in at runtime before sending the query to the DBMS. Each answer should specify the target table(s), the type(s) of indexes (either B+ tree or hashed whichever is better), whether each index should be clustered or unclustered, and the index key field(s).

(4 pts) SELECT sent, COUNT(DISTINCT user) FROM Chirps GROUP BY sent WHERE sent > value;

<table>
<thead>
<tr>
<th>Table</th>
<th>Kind</th>
<th>Clustered? (y/n):</th>
<th>Indexed field(s):</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chirp</td>
<td>B+</td>
<td>yes</td>
<td>sent (-1 As the next answer is the best option)</td>
</tr>
</tbody>
</table>

Or

| Chirp  | B+    | no               | <sent, user> |

(3 pts) SELECT * FROM Chirps WHERE text LIKE ‘%value%’;

Table: Kind: Clustered? (y/n): Indexed field(s):
No index
Question 4: Physical Design (continued)

Users (uid, firstname, lastname, age, motto)
Likes (user1, user2)
Chirps (mid, user, respto, sent, text)
Hobbies (user, hobby)

(8 pts) `SELECT * FROM Hobbies h, Users u WHERE h.user = u.uid AND h.hobby = value;`

<table>
<thead>
<tr>
<th>Table</th>
<th>Kind</th>
<th>Clustered? (y/n)</th>
<th>Indexed field(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hobby</td>
<td>Hash</td>
<td>yes</td>
<td>hobby</td>
</tr>
<tr>
<td>Users</td>
<td>Hash</td>
<td>no</td>
<td>uid</td>
</tr>
</tbody>
</table>

(6 pts) `SELECT * FROM Users WHERE lastname > value AND firstname = value;`

<table>
<thead>
<tr>
<th>Table</th>
<th>Kind</th>
<th>Clustered? (y/n)</th>
<th>Indexed field(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>B+</td>
<td>yes</td>
<td>&lt;first name, last name&gt;</td>
</tr>
</tbody>
</table>

(4 pts) `SELECT * FROM Users WHERE age = value;`

<table>
<thead>
<tr>
<th>Table</th>
<th>Kind</th>
<th>Clustered? (y/n)</th>
<th>Indexed field(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>Hash</td>
<td>Yes</td>
<td>age</td>
</tr>
</tbody>
</table>
Question 5: NoSQL Databases (25 points)

Suppose that we have an Endterm dataverse containing the following types and datasets in AsterixDB based on restructuring the reference data for NoSQL.

```
CREATE TYPE ChirpType AS {
    mid: int,
    user: int,
    respto: int?,
    sent: text, -- use ISO stringified dates
    text: string
};
CREATE TYPE UserType AS {
    uid: int,
    name: {first: string, last: string},
    motto: string?,
    hobbies: [string]?,
    likes: [int]?
};
CREATE DATASET Chirps (ChirpType)
    PRIMARY KEY mid;
CREATE DATASET Users (UserType)
    PRIMARY KEY uid;
```

The following statements have been run to insert the data from the last page into the two datasets above:

```
INSERT INTO Chirps (]
    { "mid": 1, "user": 101, "sent": "2017-05-16", "text": "i'm feeling kind of blue" },
    . . . .
    { "mid": 5, "user": 103, "sent": "2018-12-25", "text": "merry christmas y'all" }
]);

INSERT INTO Users (]
    { "uid": 101, "name": {"first": "Miles", "last": "Davis"},
    "motto": "so what", "hobbies": [ "music", "drugs" ], "likes": [ 103, 104 ] }
    . . . .
    { "uid": 106, "name": {"first": "Bill", "last": "Jones"},
    "motto": "don't ask", "hobbies": [ "karate", "running" ], "age": 70 }
));
```

(a) (10 pts) Below are five potential additional Users (the first four) and Chirps records. Indicate for each one, by circling the appropriate letter to its left, whether trying to INSERT that record into the appropriate (initial) dataset would succeed (S) or fail (F) based on the data type and dataset definitions given above.

S F { "uid": 203, "name": {"first": "Diego"}, "age":16,
    "hobbies":[], "likes": [203], "breed": "Blue Russian"}
S F { "uid": 201, "name": {"first": "Donnie", "last": "Frump"},
    "motto": "no collusion", "hobbies": [ "tweeting" ], "likes": [ 201 ], "age": 72 }
S F { "mid": 8, "user": 201, "respto":"congress", "sent": "2019-06-07", "text": "it's a witch hunt" }
Question 5: NoSQL Databases (cont.)

| CREATE TYPE ChirpType AS {               | CREATE TYPE UserType AS {          |
|    mid: int,                           |    uid: int,                        |
|    user: int,                         |    name: {first: string,           |
|    respto: int?,                      |       last: string},               |
|    sent: text, -- use ISO stringified dates |    motto: string?,                  |
|    text: string                       |    hobbies: [string]?,              |
| }                                    |    likes: [int]?                    |
|                                       | }                                    |
| CREATE DATASET Chirps (ChirpType)     | CREATE DATASET Users (UserType)     |
|     PRIMARY KEY mid;                  |     PRIMARY KEY uid;                |

(b) (10 pts) App developer Overly Normal knows SQL and he wants to write a SQL++ query that in a SQL-based relational database system would be written as follows against our reference data:

```sql
SELECT DISTINCT h.hobby FROM Hobbies h, Users u WHERE h.hobby LIKE '%u%' AND u.uid = h.user;
```

Identify the SQL++ query (or queries) below that will return the same answer once Overly has moved to his new AsterixDB database. Circle their query number(s) at the bottom of the list of suggested queries.

**Q1:**

```
SELECT DISTINCT u.hobbies
FROM Users u WHERE SOME hobby IN u.hobbies SATISFIES LIKE '%u%';
```

**Q2:**

```
SELECT DISTINCT hobby
FROM Users u, u.hobbies hobby WHERE hobby LIKE '%u%';
```

**Q3:**

```
SELECT DISTINCT h.hobby
FROM Users u, u.hobbies h WHERE u.uid = h.user AND h.hobby LIKE '%u%';
```

**Q4:**

```
SELECT DISTINCT h AS hobby
FROM Users u UNNEST u.hobbies h WHERE h LIKE '%u%';
```

**Q5:**

```
SELECT hobby
FROM Users u UNNEST u.hobbies hobby WHERE hobby LIKE '%u%';
```

Equivalent SQL++ query (or queries): **Q1 Q2 Q3 Q4 Q5**

(c) (5 pts) Indicate whether each of the following statements are true (T) or false (F) by circling the appropriate letter.

**T** The SQL++ query **SELECT VALUE c FROM Chirps c** will produce the same result as the SQL query **SELECT * FROM Chirps** would produce in a relational DBMS;
**T  F** It is possible to store first normal form (1NF) data in a NoSQL database.

**T  F** SQL and SQL++ are similar in that both allow queries that do grouping without aggregation.

**T  F** A SQL++ `SELECT` or `SELECT VALUE` query always returns a set (array) of something as its result.

---

**Appendix: Exam Reference Data**

### Users

<table>
<thead>
<tr>
<th>uid</th>
<th>firstname</th>
<th>lastname</th>
<th>age</th>
<th>motto</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Miles</td>
<td>Davis</td>
<td>33</td>
<td>so what</td>
</tr>
<tr>
<td>102</td>
<td>Emily</td>
<td>Smith</td>
<td>33</td>
<td>ask me if I care</td>
</tr>
<tr>
<td>103</td>
<td>Sally</td>
<td>Fourth</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>Bill</td>
<td>Smith</td>
<td>33</td>
<td>you only live once</td>
</tr>
<tr>
<td>105</td>
<td>Stevie</td>
<td>Knicks</td>
<td>32</td>
<td>ask and ye shall receive</td>
</tr>
<tr>
<td>106</td>
<td>Bill</td>
<td>Jones</td>
<td>70</td>
<td>don’t ask</td>
</tr>
</tbody>
</table>

### Likes

<table>
<thead>
<tr>
<th>user1</th>
<th>user2</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>103</td>
</tr>
<tr>
<td>103</td>
<td>101</td>
</tr>
<tr>
<td>102</td>
<td>105</td>
</tr>
<tr>
<td>105</td>
<td>106</td>
</tr>
<tr>
<td>101</td>
<td>104</td>
</tr>
<tr>
<td>104</td>
<td>101</td>
</tr>
<tr>
<td>105</td>
<td>102</td>
</tr>
</tbody>
</table>

### Hobbies

<table>
<thead>
<tr>
<th>user</th>
<th>hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>music</td>
</tr>
<tr>
<td>101</td>
<td>drugs</td>
</tr>
<tr>
<td>102</td>
<td>running</td>
</tr>
<tr>
<td>103</td>
<td>googling</td>
</tr>
<tr>
<td>105</td>
<td>music</td>
</tr>
<tr>
<td>106</td>
<td>karate</td>
</tr>
<tr>
<td>106</td>
<td>running</td>
</tr>
</tbody>
</table>

### Chirps

<table>
<thead>
<tr>
<th>mid</th>
<th>user</th>
<th>respto</th>
<th>sent</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101</td>
<td></td>
<td>2017-05-16</td>
<td>i’m feeling kind of blue</td>
</tr>
<tr>
<td>2</td>
<td>104</td>
<td></td>
<td>2018-07-03</td>
<td>why do cats get nine lives?</td>
</tr>
<tr>
<td>3</td>
<td>101</td>
<td></td>
<td>2018-09-01</td>
<td>it’s around midnight</td>
</tr>
<tr>
<td>4</td>
<td>105</td>
<td>2</td>
<td>2018-10-10</td>
<td>it’s not fair</td>
</tr>
<tr>
<td>5</td>
<td>103</td>
<td></td>
<td>2018-12-25</td>
<td>merry christmas y’all</td>
</tr>
</tbody>
</table>

---

**CREATE TABLE** Chirps (  
  mid INT,  
  user INT,  
  respto INT,  
  sent DATE,  
  text VARCHAR(120),  
  PRIMARY KEY (mid),  
  FOREIGN KEY (user)  
    REFERENCES Users (uid),  
  FOREIGN KEY (respto)  
    REFERENCES Chirps (mid)  
);  

**CREATE TABLE** Hobbies (  
  user INT,  
  hobby VARCHAR(20),  
  PRIMARY KEY (user, hobby),  
  FOREIGN KEY (user)  
    REFERENCES Users (uid)  
);  

**CREATE TABLE** Users (  
  uid INT,  
  firstname VARCHAR(40),  
  lastname VARCHAR(40),  
  age INT,  
  motto VARCHAR(80),  
  PRIMARY KEY (uid)  
);  

-- Has an entry if user1 Likes user2  

**CREATE TABLE** Likes (  
  user1 INT,  
  user2 INT,  
  PRIMARY KEY (user1, user2),  
  FOREIGN KEY (user1)  
    REFERENCES Users (uid),  
  FOREIGN KEY (user2)  
    REFERENCES Users (uid)  
);