0/29 Questions Answered
29 questions with unsaved changes

Endterm

STUDENT NAME

Search students by name or email...

Q1 Preliminaries (MUST READ!)

1 Point

Instructions

The allowed time for the exam this time will be 75 minutes (65 minutes plus a 10-minute technology glitch buffer). Be sure to pay attention to the time and budget your exam time accordingly!

The exam is open pre-prepared cheat sheet, open book, open notes, open web browser, and even open MySQL and open AsterixDB. You are just not allowed to communicate with or otherwise interact with other students (or friends) during the course of the exam, and this includes your HW brainstorming buddy. This exam is to be a solo effort!

Read each question carefully, in its entirety, and then answer each part of the question.

If you don't understand something, make your best guess; if you find ambiguities in a question, note the interpretation that you are taking.

Acknowledgement: I certify that I am taking this exam myself, on my own, with honesty and integrity, without interaction with others during the exam, and without having obtained any information about the exam's content from others prior to taking it.

- (●) True
- ( ) False

A Running Example

Consider the following relational schema, where primary keys are indicated in bold (and some are composite primary keys). Assume that the data is going to be stored in a new open source DBMS called **SQLThis!**. You will recall from Quiz 10 that **SQLThis!** supports both clustered and unclustered B+ tree indexes. It stores its data records in a separate heap file and uses RIDs in index leaf page entries to refer to them. It creates *no* indexes by default - not even on primary keys! - so its users must tell it *everything* they want indexed. Here are the tables:

```
CREATE TABLE customers (
    custid VARCHAR(40),
    name VARCHAR(40) NOT NULL,
    street VARCHAR(40) NOT NULL,
    city VARCHAR(40) NOT NULL,
    zipcode VARCHAR(40),
    rating INTEGER,
    PRIMARY KEY (custid)
);

CREATE TABLE orders (
    orderno INTEGER NOT NULL,
    custid VARCHAR(40) NOT NULL,
    order_date VARCHAR(40) NOT NULL,
    ship_date VARCHAR(40),
    PRIMARY KEY (orderno)
);

CREATE TABLE items (
    orderno INTEGER,
    itemno INTEGER,
    qty INTEGER,
    price DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (orderno, itemno)
);
```

The **SQLThis!** syntax for creating indexes is:

```
CREATE CLUSTERED INDEX ixdname ON tblname(fieldlist);
CREATE UNCLUSTERED INDEX idxname ON tblname (fieldlist);
```

Here is some example relational data (for **SQLThis!**):

```
customers (custid, name, street, city, zipcode, rating):
    ("C01", "T. Cruise", "201 Main St.", "St. Louis, MO", "63101", 750 )
    ("C02", "B. Pitt", "360 Mountain Ave.", "St. Louis, MO", "63101", null )
    ("C03", "T. Hanks", "120 Harbor Blvd.", "Boston, MA", "02115", 750 )
    ("C04", "S. Loren", "Via del Corso", "Rome, Italy", null, 625 )

orders (orderno, custid, order_date, ship_date):
    ( 1001, "C01", "2017-05-01", "2017-05-03" )
    ( 1002, "C01", "2017-10-13", null )
    ( 1003, "C02", "2017-08-30", null )
    ( 1004, "C03", "2017-04-29", "2017-05-03" )

items (itemno, qty, price, orderno):
    ( 460, 20, 99.99, 1002 )
    ( 460, 95, 100.99, 1001 )
    ( 680, 150, 8.75, 1001 )
    ( 347, 120, 22.0, 1003 )
    ( 780, 1, 1500.0, 1003 )
    ( 347, 5, 19.99, 1004 )
    ( 193, 2, 28.89, 1004 )
```

Here is the example data in JSON form (for **AsterixDB**):

```
customers:
{ "custid": "C01", "name": "T. Cruise",
    "address": { "street": "201 Main St.", "city": "St. Louis, MO", "zipcode": "63101" },
    "rating": 750 }
{ "custid": "C02", "name": "B. Pitt",
    "address": { "street": "360 Mountain Ave.", "city": "St. Louis, MO", "zipcode": "63101" } }
{ "custid": "C03", "name": "T. Hanks",
    "address": { "street": "120 Harbor Blvd.", "city": "Boston, MA", "zipcode": "02115" },
     "rating": 750 },
{ "custid": "C04", "name": "S. Loren",
    "address": { "street": "Via del Corso", "city": "Rome, Italy" },
    "rating": 625 }

orders:
{ "orderno": 1001, "custid": "C01", "order_date": "2017-05-01", "ship_date": "2017-05-03",
  "items": [ {"itemno": 460, "qty": 95, "price": 100.99},
             {"itemno": 680, "qty": 150, "price": 8.75} ] }
{ "orderno": 1002, "custid": "C01", "order_date": "2017-10-13",
   "items": [ {"itemno": 460, "qty": 20, "price": 99.99} ]}
{ "orderno": 1003, "custid": "C02", "order_date": "2017-08-30",
   "items": [ {"itemno": 347, "qty": 120, "price": 22.00},
              {"itemno": 780, "qty": 1, "price": 1500.00} ] }
{ "orderno": 1004, "custid": "C03", "order_date": "2017-04-29", "ship_date": "2017-05-03",
   "items": [ {"itemno": 347, "qty": 5, "price": 19.99},
              {"itemno": 193, "qty": 2, "price": 28.89} ] }
```

NOTE: You needn't try to load any of this data into anything for this exam! You should be able to look at the questions and data and then answer based on your experience and knowledge. (Plus then you'd have to download and install **SQLThis!**... :-))

Save Answer ***Unsaved Changes**

Q2 Indexing Truth or Consequences

10 Points

For each of the following statements, you should indicate whether the given statement is TRUE or FALSE.

Q2.1
1 Point

With one possible exception, each non-leaf node in a B+ tree of order d = 100 will have between 101 and 201 children.

- (●) TRUE
- ( ) FALSE

Save Answer ***Unsaved Changes**

Q2.2
1 Point

In an ISAM index the number of pages on the search path from the root to any of the index's data entries (and hence the number of pages read) will be the same for every entry in the index.

- ( ) TRUE
- (●) FALSE

Save Answer ***Unsaved Changes**

Q2.3
1 Point

A clustered index is never better than an unclustered index for an exact-match lookup like SELECT * FROM items WHERE orderno = **?**. (The **?** in the query represents a query parameter that an application program will fill in at runtime.)

- ( ) TRUE
- (●) FALSE

Save Answer ***Unsaved Changes**

Q2.4
1 Point

Assuming all pages of a Static Hashed index are on disk at the start of an exact match lookup, the minimum number of page reads required to locate a desired data entry is 2.

- ○ TRUE
- ◉ FALSE

Save Answer ***Unsaved Changes**

Q2.5
1 Point

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

- ◉ TRUE
- ○ FALSE

Save Answer ***Unsaved Changes**

Q2.6
1 Point

Unlike SQL database systems (such as MySQL), NoSQL databases (such as Apache AsterixDB) do not support indexes.

- ○ TRUE
- ◉ FALSE

Save Answer ***Unsaved Changes**

Q2.7
1 Point

If our data was stored in MySQL instead of **SQLThis!**, the primary index for orders would be a clustered index on orders(orderno). Suppose we also have a secondary index on orders(custid) in our MySQL database. In this case, the query SELECT orderno FROM orders WHERE custid = **?** would be index-only.

- (●) TRUE
- ( ) FALSE

Save Answer ***Unsaved Changes**

Q2.8
1 Point

In a real system, the B+ tree indexing code makes use of a constant **d** called the order of the tree to determine when a given index page is too empty after one of its entries is deleted (triggering a rebalancing action if it is left with fewer than d entries).

- ( ) TRUE
- (●) FALSE

Save Answer ***Unsaved Changes**

Q2.9
1 Point

Now that solid-state drive (SSD) technology is becoming commonplace, it is no longer important for a DBMS to do its I/O in units of pages (rather than individual records) like it was when secondary storage was based on hard disk drive (HDD) technology.

- ( ) TRUE
- (●) FALSE

Save Answer ***Unsaved Changes**

Save Answer **Unsaved Changes**

Q2.10
1 Point

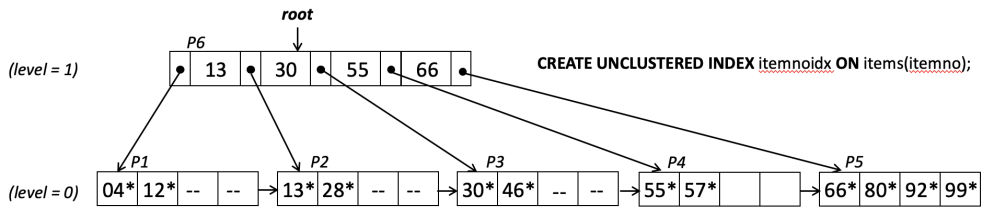
When creating a B+ tree index on a composite key, such as an index on customer(city, street), the order that the components (city and street) of the key are listed in in the CREATE INDEX statement matters.

- (●) TRUE
- ( ) FALSE

Save Answer ***Unsaved Changes**

Q3 Save the Trees
29 Points

Below is an example of a B+ tree index. The pages are numbered for reference in the questions that follow. If new pages are needed for an index operation, assume they will come from the end of the file (so their page numbers will continue in sequence). Answer each question as best you can, making use of the indicated notation for index node content since you can't draw pictures on the exam. (You can draw pictures for yourself and then translate the relevant nodes into this notation to give your answer.) Please show the whole tree in your insert/delete answers -- not just changed nodes -- to simplify grading and enable partial credit if needed.



Example index node notation:

```
P6: [P1|13|P2|30|P3|55|P4|66|P5]
P1: [04*|12*|--|--]
```

```
P2: [13*|28*|--|--]
P3: [30*|46*|--|--]
P4: [55*|57*|--|--]
P5: [66*|80*|92*|99*]
```

Q3.1

5 Points

Consider the SQL query

```
SELECT * FROM items WHERE itemno > 13
```

Which pages, in which order, will the DBMS read if it utilizes the B+ tree index shown above? (Answer in list form, e.g., say P4, P2 if your answer is page P4 and then P2.)

P6, P2, P3, P4, P5

Save Answer ***Unsaved Changes**

Q3.2

8 Points

In addition to reading index pages, of course, the **SQLThis!** query processor will have to read some heap file pages in order to fetch the qualifying records. Assume a tiny buffer pool that can fit only one heap file page at a time, so if two consecutive record accesses are for records on different pages, each will result in a disk read operation. Assume for this problem that itemno's are currently distinct (i.e., that no two orders happen to have the same item).

(a) (3 pts) How many record accesses will occur when the system runs the SELECT query?

9

(b) (3 pts) How many heap file page reads will occur in the worst case when the system runs the aforementioned SELECT query?

9

(c) (2 pts) Briefly explain the reason for your answer to (b):

The index is unclustered, so every record access is likely to result in a disk read

Save Answer ***Unsaved Changes**

Q3.3

8 Points

Show the results of performing the SQL insert operation:

```
INSERT INTO items (itemno, qty, price, orderno)
          VALUES (70, 10, 99.99, 777);
```

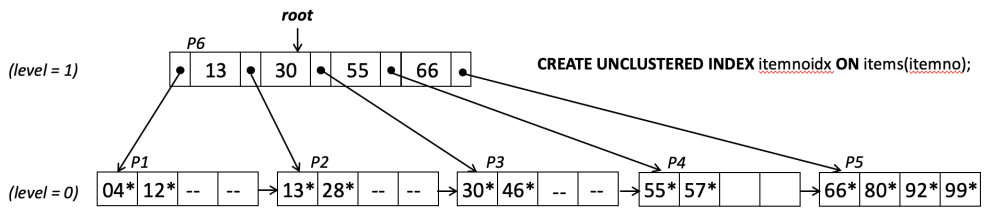
using the node notation shown underneath the figure. Assume the insertion strategy (from lecture) that favors **splitting over redistribution**. Please show the whole new tree (even the unaffected nodes). When numbering pages, be careful to allocate new pages only when needed, and number them in the order in which they are allocated (to fully demonstrate your understanding of insertions).

P9: [P8|55|P9|--|--|--|--|--|--]
P6: [P1|13|P2|30|P3|--|--|--|--]
P8: [P4|66|P5|80|P7|--|--|--|--]
P1: [04*|12*|--|--]
P2: [13*|28*|--|--]
P3: [30*|46*|--|--]
P4: [55*|57*|--|--]
P5: [66*|70*|--|--]
P7: [80*|92*|99*|--]

Example index node notation:

Save Answer ***Unsaved Changes**

Q3.4
8 Points



Example index node notation:

```
P6: [P1|13|P2|30|P3|55|P4|66|P5]
P1: [04*|12*|--|--]
P2: [13*|28*|--|--]
P3: [30*|46*|--|--]
P4: [55*|57*|--|--]
P5: [66*|80*|92*|99*]
```

Starting from the original tree again (i.e., not from your answer to the previous question), show the results of performing the SQL delete operation:

```
DELETE FROM items WHERE itemno = 12;
```

Assume the deletion strategy (from lecture) that favors **redistribution over merging**. Please show all of the nodes in the resulting tree (even those not affected).

P6: [P1|30|P3|55|P4|66|P5]
P1: [04*|13*|28*|--]
P3: [30*|46*|--|--]
P3: [55*|57*|--|--]
P5: [66*|80*|92*|99*]

Save Answer ***Unsaved Changes**

Q4 Let's Get Physical
25 Points

SQLThis! supports both clustered and unclustered B+ tree indexes. It stores the data records in a heap file and uses RIDs to refer to them. It creates **no** indexes by default - not even on primary keys! - so users must tell it **everything** they want indexed. Here are the tables again for easy reference - assume that they are much bigger than the sample database at the start of the exam :-):

```
CREATE TABLE customers (
    custid VARCHAR(40),
    name VARCHAR(40) NOT NULL,
    street VARCHAR(40) NOT NULL,
    city VARCHAR(40) NOT NULL,
    zipcode VARCHAR(40),
    rating INTEGER,
    PRIMARY KEY (custid)
);

CREATE TABLE orders (
    orderno INTEGER NOT NULL,
    custid VARCHAR(40) NOT NULL,
    order_date VARCHAR(40) NOT NULL,
    ship_date VARCHAR(40),
    PRIMARY KEY (orderno)
);

CREATE TABLE items (
    orderno INTEGER,
    itemno INTEGER,
    qty INTEGER,
    price DECIMAL(8,2) NOT NULL,
    PRIMARY KEY (orderno, itemno)
);
```

And again, the **SQLThis!** syntax for creating indexes is:

```
CREATE CLUSTERED INDEX ixdname ON tblname(fieldlist);
CREATE UNCLUSTERED INDEX idxname ON tblname (fieldlist);
```

Consider each of the following problems individually - i.e., create the best index (or indexes!) for its given SQL query. Express your answer by writing `CREATE ... INDEX` statements. If the given query doesn't warrant creation of an index, answer "No index" in the box (so we can distinguish such answers from questions left unanswered). The **?**'s in the queries represent query parameters that an application will fill in at runtime.

Q4.1

4 Points

```
SELECT *
FROM customers
WHERE zipcode = ? AND street LIKE '%?%';
```

CREATE CLUSTERED INDEX myidx ON customers(zipcode, street);

Save Answer ***Unsaved Changes**

Q4.2

4 Points

```
SELECT COUNT(DISTINCT street)
FROM customers
WHERE zipcode = ? AND street LIKE '%?%';
```

CREATE UNCLUSTERED INDEX myidx ON customers(zipcode, street);

Save Answer ***Unsaved Changes**

Q4.3

8 Points

```
SELECT c.custid, c.name, i.itemno, i.qty, i.price
FROM customers c, orders o, items i
WHERE c.custid = o.custid AND o.orderno = i.orderno
  AND c.rating = 750;
```

CREATE CLUSTERED INDEX myidx1 ON customers(rating);
CREATE CLUSTERED INDEX myidx2 ON orders(custid);
CREATE CLUSTERED INDEX myidx3 ON items(orderno);

//Another advanced answer

```
CREATE CLUSTERED   INDEX myidx1 ON customers(rating);
CREATE UNCLUSTERED INDEX myidx2 ON orders(custid,
orderno);
CREATE CLUSTERED   INDEX myidx3 ON items(orderno);
```

Save Answer ***Unsaved Changes**

Q4.4
8 Points

```
SELECT c.city, MIN(c.rating) AS rmin, MAX(c.rating) AS rmax,
       COUNT(DISTINCT c.rating) AS rnum
FROM customers c
GROUP BY c.city;
```

```
CREATE UNCLUSTERED INDEX myidx ON customers(city, rating);
```

Save Answer ***Unsaved Changes**

Q4.5
1 Point

```
SELECT * FROM orders WHERE custid != ?;
```

```
No index :-)
```

Save Answer ***Unsaved Changes**

Q5 The Sequel to SQL
25 Points

Time to consider your new favorite DBMS, **Apache AsterixDB** of course, and your new favorite query language, **SQL++**. Here again is the NoSQL (JSON) version of the data:

```
customers:
{ "custid": "C01", "name": "T. Cruise",
    "address": { "street": "201 Main St.", "city": "St. Louis, MO", "zipcode": "63101" },
    "rating": 750 }
{ "custid": "C02", "name": "B. Pitt",
    "address": { "street": "360 Mountain Ave.", "city": "St. Louis, MO", "zipcode": "63101" } }
{ "custid": "C03", "name": "T. Hanks",
    "address": { "street": "120 Harbor Blvd.", "city": "Boston, MA", "zipcode": "02115" },
     "rating": 750 },
{ "custid": "C04", "name": "S. Loren",
    "address": { "street": "Via del Corso", "city": "Rome, Italy" },
    "rating": 625 }

orders:
{ "orderno": 1001, "custid": "C01", "order_date": "2017-05-01", "ship_date": "2017-05-03",
  "items": [ {"itemno": 460, "qty": 95, "price": 100.99},
             {"itemno": 680, "qty": 150, "price": 8.75} ] }
{ "orderno": 1002, "custid": "C01", "order_date": "2017-10-13",
   "items": [ {"itemno": 460, "qty": 20, "price": 99.99} ]}
{ "orderno": 1003, "custid": "C02", "order_date": "2017-08-30",
   "items": [ {"itemno": 347, "qty": 120, "price": 22.00},
              {"itemno": 780, "qty": 1, "price": 1500.00} ] }
{ "orderno": 1004, "custid": "C03", "order_date": "2017-04-29", "ship_date": "2017-05-03",
   "items": [ {"itemno": 347, "qty": 5, "price": 19.99},
              {"itemno": 193, "qty": 2, "price": 28.89} ] }
```

Q5.1

5 Points

What will the following SQL++ query print if the data above is stored in two datasets named customers and orders?
(Recall that every SQL++ SELECT query returns an *array of JSON data model instances*, and be sure to include the right fields and field names in any JSON objects in your answer.)

```
SELECT VALUE c.name
FROM customers c
WHERE c.address.zipcode IS NOT MISSING;
```

"T. Cruise"
"B. Pitt"
"T. Hanks"

Save Answer ***Unsaved Changes**

Q5.2

8 Points

What will the following SQL++ query print given the data above? (Recall that every SQL++ SELECT query returns an *array of JSON data model instances*, and be sure to include the right field names in any JSON objects in your answer.) To maybe save you some typing, here are all the items from the data in copyable form:

```
{ "itemno": 460, "qty": 95, "price": 100.99 }
{ "itemno": 680, "qty": 150, "price": 8.75 }
{ "itemno": 347, "qty": 120, "price": 22.0 }
{ "itemno": 780, "qty": 1, "price": 1500.0 }
{ "itemno": 347, "qty": 5, "price": 19.99 }
{ "itemno": 193, "qty": 2, "price": 28.89 }
```

Now here's the SQL++ query to show the results of:

```
SELECT o.orderno, o.custid, o.items,
       ARRAY_COUNT(o.items) AS numitems
FROM orders o
WHERE SOME i IN o.items
      SATISFIES i.price < 20.00 AND i.qty < 100;
```

{ "orderno": 1004, "numitems": 2, "custid": "C03", "items": [{ "itemno": 347, "qty": 5, "price": 19.99 }, { "itemno": 193, "qty": 2, "price": 28.89 }] }

Save Answer ***Unsaved Changes**

Q5.3

4 Points

(a) (2 pts) Consider what the following SQL++ query produces:

```
SELECT c.name, (SELECT VALUE o.order_date
                FROM orders o
                WHERE o.custid = c.custid) AS dates
FROM customers c;
```

Could you produce a structurally similar query result using SQL and **SQLThis!** on the relational version of the data?

○ Yes

◉ No

(b) (2 pts) Why or why not? (Explain briefly)

Because the query result is nested, i.e., it's non-1NF.

Save Answer ***Unsaved Changes**

Q5.4
8 Points

Write a SQL++ query to return the items ordered by the customer named "B. Pitt". In your results, don't abbreviate the quantity as "qty" -- call it "quantity" for a better user experience.

```
SELECT i.itemno, i.qty AS quantity, i.price
FROM customers c, orders o, o.items i
WHERE c.name = "B. Pitt"
        AND c.custid = o.custid;
```

Save Answer ***Unsaved Changes**

Q6 Would ACID Kill Coronavirus?
10 Points

You've heard that **SQLThis!** supports the full gamut of SQL transaction options, which is attractive to you. Here once again is the relational version of our data:

```
customers (custid, name, street, city, zipcode, rating):
    ("C01", "T. Cruise", "201 Main St.", "St. Louis, MO", "63101", 750 )
    ("C02", "B. Pitt", "360 Mountain Ave.", "St. Louis, MO", "63101", null )
    ("C03", "T. Hanks", "120 Harbor Blvd.", "Boston, MA", "02115", 750 )
    ("C04", "S. Loren", "Via del Corso", "Rome, Italy", null, 625 )

orders (orderno, custid, order_date, ship_date):
    ( 1001, "C01", "2017-05-01", "2017-05-03" )
    ( 1002, "C01", "2017-10-13", null )
    ( 1003, "C02", "2017-08-30", null )
    ( 1004, "C03", "2017-04-29", "2017-05-03" )

items (itemno, qty, price, orderno):
    ( 460, 20, 99.99, 1002 )
    ( 460, 95, 100.99, 1001 )
    ( 680, 150, 8.75, 1001 )
    ( 347, 120, 22.0, 1003 )
    ( 780, 1, 1500.0, 1003 )
    ( 347, 5, 19.99, 1004 )
    ( 193, 2, 28.89, 1004 )
```

Answer each of the following questions. (Warning: This problem has no partial credit; you'll have to get each one exactly right to earn its two points.)

Q6.1

2 Points

Transaction T1 contains the following SQL statements:

```
BEGIN;
   UPDATE customers
   SET rating = rating + 50
   WHERE rating = 750;
ROLLBACK;
```

Transaction T2 contains the following SQL statements:

```
BEGIN;
   SELECT SUM(rating) AS rsum FROM customers;
COMMIT;
```

What are all the possible rsum values that T2 might produce if these two transactions are executed concurrently and T2's SQL isolation level is set to SERIALIZABLE?

- [ ] null
- [x] 2125
- [ ] 2175
- [ ] 2225

Save Answer ***Unsaved Changes**

Q6.2

2 Points

What are all the possible rsum values that T2 might produce if these two transactions are executed concurrently and T2's SQL isolation level is set to READ UNCOMMITTED?

- [ ] null
- [x] 2125
- [x] 2175
- [x] 2225

Save Answer ***Unsaved Changes**

Q6.3

2 Points

What are all the possible rsum values that T2 might produce if T2 runs after T1 has completed?

- [ ] null
- [x] 2125
- [ ] 2175
- [ ] 2225

Save Answer ***Unsaved Changes**

Q6.4

2 Points

What are all the possible rsum values that T2 might produce if these two transactions are executed concurrently, T2's SQL isolation level is set to SERIALIZABLE, and T1's last statement is changed to COMMIT instead of ROLLBACK?

- [ ] null
- [x] 2125
- [ ] 2175
- [x] 2225

Save Answer ***Unsaved Changes**

Q6.5

2 Points

Would your answer to the previous question be different if T1 was "unwrapped" (with no BEGIN/COMMIT) and instead it just said:

```
UPDATE customers
SET rating = rating + 50
```

```
WHERE rating = 750;
```

- ○ Yes
- ◉ No

Save Answer ***Unsaved Changes**

Save All Answers

Submit & View Submission >