CS 245 Midterm Exam
Winter 2022

- Please read all instructions carefully. In case of any ambiguity, write any assumptions you made in your answer. We can't guarantee that we can answer questions from students live during the test, but tell us about anything you found unclear later (in a private Ed post).

- There are five problems for a total of 65 points. You have two hours to **take and upload** the test **and mark which answers** correspond to each question in Gradescope (**please give yourself ~5-10 minutes to upload/mark answers within the 2 hours)**. If you have issues uploading the test on Gradescope within the two hours, please email it to Prof. Zaharia at matei@cs.stanford.edu as quickly as you can after that.

- The test is open-book, but you are not allowed to communicate with other people to do it. This includes asking public questions on Ed. You may also use the Internet during the test, but keep in mind that many online resources might use terms differently from our course, and that directly copying an online resource is considered plagiarism and is not allowed. We don't think you will need resources other than the course materials.

- You may complete this test digitally (e.g., using the Annotate tools in Acrobat Reader), or print it, handwrite your answers legibly and scan it using a scanner or a mobile app such as GeniusScan, or upload a list of answers, with the same numbering, as a separate document. Ensure that the file you upload to Gradescope is sharp, legible, and aligned, and you save time to mark which answers correspond to which questions.

- Solutions will be graded on correctness and clarity. For the long-answer problems, please show your intermediate work. Each problem has a relatively simple to explain solution, and we may deduct points if your solution is much more complex than necessary. Partial solutions will be graded for partial credit.

NAME: ____

SUID: ____

In accordance with both the letter and spirit of the Stanford Honor Code, I have neither given nor received assistance on this test. A typed signature is fine. If uploading answers as a separate document, please include your Name, SUID, and Signature clearly on your submission.

SIGNATURE: ____

Problem 1: Short Answers (16 points)

a) (2 points) Which of the following are drawbacks of the XRM data model used by System R used in phase 0, which motivated the change to indices in phase 1? *(Check all that apply.)*

- ☐ Cost of creating and manipulating TID lists.
- ☐ Separation of data from tuples in TID lists when a query is extremely selective and the data values themselves are relatively large.
- ☐ Separation of data from tuples when data values are relatively small.
- ☐ Since data is stored separately from the records in the TID lists, commonly used values can be stored once.

Answer: box 1 and box 3

b) (2 points) What combination of techniques would most likely perform better in an analytical DBMS? *(Choose 1 answer.)*

(a) B-tree indexes, coarse grained locks, shadow pages for recovery
(b) Row oriented storage, fine grained locks, logging of writes
(c) Column oriented storage, hierarchical locks, logging of queries
(d) Column oriented storage, coarse grained locks, logging of queries

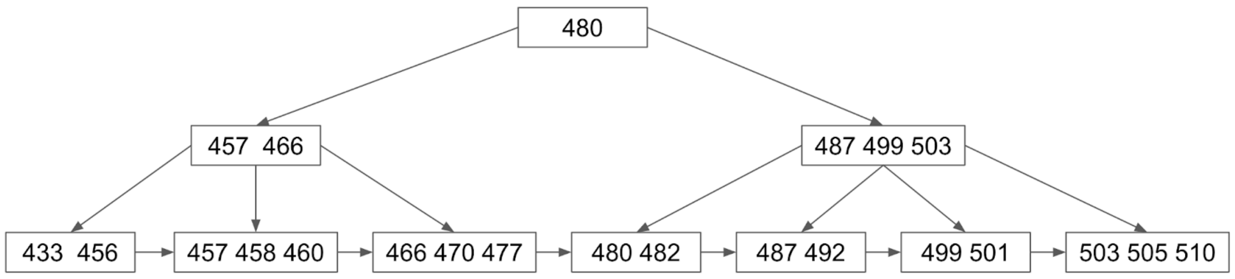
c) (5 points) Consider the following desired behaviors, and check off in each corresponding box whether RAID 0, RAID 1, or RAID 5 provides the given behavior/scenario. Assume a single disk can read/write data sequentially at a rate of 200 MB/s.

Data format	**Raid 0 (with 2 disks)**	**Raid 1 (with 2 disks)**	**Raid 5 (with 4 disks)**
Tolerance of 1 disk failure		✓	✓
Tolerance of 2 disk failures			
>= 400 MB/s maximum read throughput	✓	✓	✓
>= 400 MB/s maximum write throughput	✓		✓
>= 800 MB/s maximum write throughput			

d) (2 points) Suppose you want to store a table where each record contains five string fields, and the fields are rarely null. You also do not plan to compress the table. What kind of storage format for records would you expect to use the *least space* for this table? *(Choose 1 answer.)*

(a) Fixed-format, fixed-length
(b) Fixed-format, variable-length
(c) Variable-format, fixed-length
(d) Variable-format, variable-length

e) (5 points) Consider the following B+ tree with order 3:



Follow the same rules taught in class for changing the tree. Two of the records' keys changed, and we would like to update the tree using these operations: DELETE(433), INSERT(515), DELETE(480), INSERT(450).

During all the operations, there are (fill in a number in each blank):

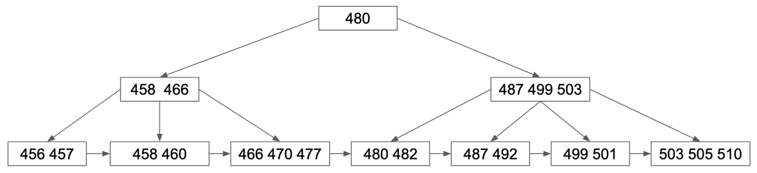
___1____ leaf overflow(s)   ___1____ non-leaf overflow(s)

___1____ leaf coalesce(s)   ___0____ non-leaf coalesce(s)

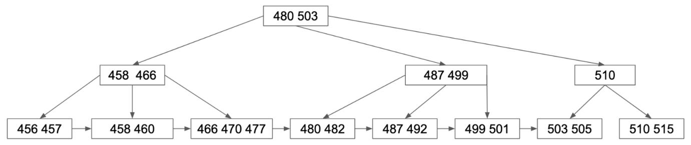
After all the operations, the root node's key value(s) are ________482, 503______ .

Order of updates:

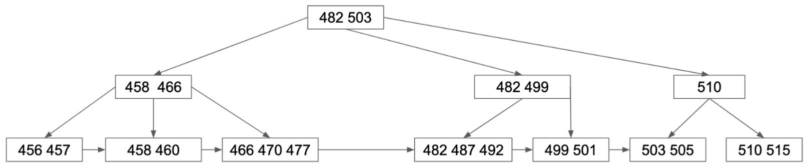
Delete 433: 1 redistributed keys



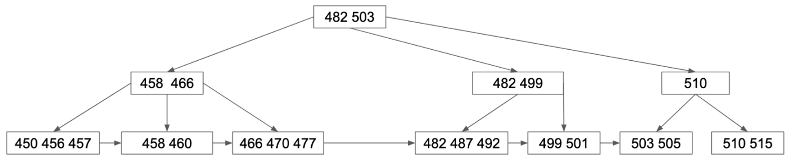
Insert 515: 1 leaf overflow, 1 non-leaf overflow



Delete 480: 1 leaf coalesce



Insert 450:



Problem 2: Indexes (14 points)

Gogo Maps would like to help users pick the best gas station. A gas station is stored with 3 fields (latitude, longitude, gas price in cents), and there are N gas stations on file. We would like to find the best way to index the gas station data to answer the following queries:

- Which gas station whose gas price < user-specified threshold is closest to the user?
- Which gas station is closest to the user?

Suppose that we index the data in a **B+ tree** whose key is the gas price.

a) (2 points) If we insert the keys into the tree one by one, constructing the tree would take a total of [[✓ O(N log(N))] | O(N) | O(log(N))] time. It then takes [O(N log(N)) | O(N) | [✓ O(log(N))]] time to update the tree if one gas station's price changes. *(Circle one choice for each case.)*

b) (2 points) Use X and N to fill in the blanks below. Assume worst-case scenarios, and start at the root node.

Suppose there are X gas stations whose gas price < user-specified threshold. We need to traverse ___X + log(N)___ pointers in the B+ tree to answer the first query.

We need to traverse ___N + log(N)___ pointers in the B+ tree to answer the second query.

Suppose instead that we index the data in a **k-d tree** whose keys are (latitude, longitude) pairs.

c) (2 points) Since all gas stations' locations are unique and known beforehand, we would like to construct the tree so that it is balanced. First, we pick the gas station with the [mean | [✓ median]] latitude to be our root node. Constructing the balanced tree with pre-sorting the points in each dimension would take [[✓ O(N log(N))] | O(N) | O(log(N))] time. *(Circle one choice each case.)*

d) (4 points) Circle the correct options to complete this algorithm for finding the single closest gas station to the user using a k-d tree *(Circle one choice for each case)*:

1. Create variables called *closestStation* to store the closest gas station found so far, and *closestDistance* to store the distance from *closestStation* to the user. Set both to *null*.
2. Start at root and then traverse down the tree recursively:
    a. If we reach a *null* node, [[✓ do nothing] | jump to step 3]
    b. Calculate the distance between the non-null node and the user's location.
    c. Update *closest_gas_station* and *closestDistance* if *closestStation* is null or the node is [[✓ closer] | farther] to the user than *closestStation*.
    d. Compare the cutting dimension (level 1 is latitude, level 2 is longitude, level 3 is latitude etc.) value of the node to the user location. Let *diff* = user's cutting dimension's value - the node's cutting dimension. If *diff* <= 0, then we traverse the [[✓ left] | right] subtree first, and [left | [✓ right]] subtree second.
    e. We can skip the second subtree if [[✓ absolute(*diff*)] | 2 * absolute(*diff*) | *diff* * *diff*] is larger than *closestDistance*.
3. Return *closestStation*.

e) (2 points) Use X and N to fill the blanks below. Assume worst-case scenarios, and assume that the search starts at the root node.

Suppose there are X gas stations whose gas price < user-specified threshold. We need to traverse _____N______ nodes in the balanced k-d tree to answer the first query.

We need to traverse ____log(N)____ nodes in the balanced k-d tree to answer the second query.

f) (2 points) Which of the two storage formats do you think is better for realistic scenarios? Provide 2 reasons for your answer.

We accepted several reasons to prefer either format. For example, because the gas stations don't move in location, the k-d tree is probably better in many cases because you never have to update it, and then it can answer location-based queries quickly.

Problem 3: Relational Algebra (11 points)

a) (2 points) Write a SQL query that represents the following relational algebra expression:

$${}_{\text{artist}}G_{\text{SUM(length)}}(\sigma_{\text{year='2022'} \wedge \text{ genre= 'pop'}}(\text{Playlist}))$$

```
SELECT SUM(length)
FROM Playlist
WHERE year = '2022' AND genre = 'pop'
GROUP BY artist
```

For parts b), c) and d), consider the following two tables and the relational algebra expression:

$$\Pi_{\text{age}}(\sigma_{\text{inspection = 'passes'} \wedge \text{ rating >= 4.0} \wedge \text{ gender = 'female'}}(\text{Restaurants} \bowtie \text{Owners}))$$

Restaurants

rid	oid	inspection	rating
1	888	passes	4.8
2	210	fails	3.0
…			

Owners

oid	name	gender	age
1	melody	female	35
2	andrew	male	32
…			

b) (2 points) Explain what the expression finds in one sentence (using words).

The age of every female owner who has a restaurant that passes on inspection and achieves at least a 4.0 rating.

c) (5 points) Which of the following are valid rewrites of the expression? *(Check all that apply.)*

[✓ ☐] $\Pi_{\text{age}}(\sigma_{\text{inspection = 'passes'} \wedge \text{ rating >= 4}}(\text{Restaurants}) \bowtie \sigma_{\text{gender = 'female'}}(\text{Owners}))$

[✓ ☐] $\Pi_{\text{age}}(\sigma_{\text{gender = 'female'}}(\sigma_{\text{inspection = 'passes'} \wedge \text{ rating >= 4}}(\text{Owners} \bowtie \text{Restaurants})))$

☐ $\sigma_{\text{gender = 'female'}}(\Pi_{\text{age}}(\sigma_{\text{inspection = 'passes'} \wedge \text{ rating >= 4}}(\text{Owners} \bowtie \text{Restaurants})))$

[✓ ☐] $\Pi_{\text{age}}(\sigma_{\text{inspection = 'passes'} \wedge \text{ rating >= 4}}(\sigma_{\text{gender = 'female'}}(\text{Restaurants} \bowtie \text{Owners})))$

[✓ ☐] $\Pi_{\text{age}}(\sigma_{\text{gender = 'female'}}(\text{Owners} \bowtie \sigma_{\text{inspection = 'passes'} \wedge \text{ rating >= 4}}(\text{Restaurants})))$

d) (2 points) Write an equivalent relational algebra expression without using a join operator. *(Hint: use Cartesian product. Feel free to write Greek symbols in words if typing your answer.)*

$$\Pi_{\text{age}}(\sigma_{\text{Restaurants.inspection = 'passes'} \wedge \text{ Restaurants.rating >= 4.0} \wedge \text{ Owners.gender = 'female'}}(\sigma_{\text{Restaurants.oid = Owners.oid}}(\text{Restaurants} \times \text{Owners})))$$

Problem 4: In-Memory Storage (8 points)

In Assignment 1, we saw two storage formats for in-memory data: row stores and column stores. Consider a table stored in RAM with the following specifications:

- The table has **R** rows and **C** columns, holding fields that are each **8 bytes**. The table is laid out contiguously (for example, all rows are next to each other in a row store).
- Every read brings **64 bytes** of data into the cache. The cost of fetching a cache line is 1, whereas the cost of reading a line that's already in cache is 0.
- The cache fits **16 cache lines** in total, and its eviction policy is Least Recently Used.
- **R, C** >> **16** and **R and C are divisible by 8.**

Any query will be implemented as follows, for both the storage formats:

```
for(int i=0; i<numRows; i++) {
      for(int j=0; j<numCols; j++) {
            long element = getTableField(i, j);
            // Code to check for conditions and update output.
      }
}
```

a) (2 points) Consider the query SELECT * FROM TABLE. Compute the cost of executing the query on both row and column stores in terms of R and C.

Row store cost: R*C*8/64 = RC/8

Column store cost: RC

b) (3 points) Consider the query SELECT col$_0$, col$_1$, …., col$_{X-1}$ FROM TABLE. For what values of X is the column store more performant than the row store? *(Briefly explain your answer).*

For this query, row store cost = R*ceil(X/8) and column store cost = RX/8.

Therefore, column cost is less than or equal to the row store cost as long as **X <= 16**, and is equal to it for X = 8 or 16.
We also accepted **X<8** as an answer (when the column cost is strictly less than the row cost).

c) (3 points) Consider the query SELECT * FROM TABLE WHERE col$_0$ = 100. Suppose that **X out of the R** rows have col$_0$ = 100 and col$_0$ is unsorted. What is the cost of this query in both the row store and the column store in terms of R, C and X? *(Briefly explain your answer).*

Row store cost: ___R - X + XC/8___

a. R - X times we do nothing except fetch col0 (when col0 != 100).

b. For X records (when col0 == 100), we have to fetch all columns into the cache.

Therefore, total cost: R - X + X*(C*8/64) = R - X + XC/8

Column store cost: __R + X(C-1)__

a. Cost of checking rows where col0 == 100:
    i. Depends on distribution of matching rows. We can't account for potential cache misses if the rows with col0 == 100 are interspersed with the other rows, and reading the matching rows causes values of col0 to fall out of the cache. Thus, the worst case cost of checking these rows is R. If we were to make a simple optimization of storing the values of col0 for every 8 consecutive rows in registers, we could check col0 for a cost of R/8. However, since we did not mention this optimization in the question, we will accept a row search cost of both R or R/8.

b. Every time col0 == 100, we need to do random accesses to fetch all the other columns for that record.

Therefore, total cost = {R or R/8} + X(C-1)

Problem 5: Query Optimization (16 points)

An online shopping company has a database with three tables about shopping activity:

- User, with fields userId (primary key), email, country
- Product, with fields prodId (primary key), name, price
- Order, with fields orderId (primary key), userId, prodId, discount

Managers often runs queries of the following form to see how each product is doing:

```
SELECT SUM(price * discount) FROM Order, User, Product
WHERE Order.userId = User.userId
  AND Order.prodId = Product.prodId
  AND Product.name = <n>
  AND User.country = <c>
```

Here, <n> and <c> are parameters that vary from query to query. In this question, we'll look at how a database can optimize these queries.

a) (1 point) The first step of query optimization would be rule-based optimizations such as selection and projection pushdowns. Which fields should be *projected* out of the User table in this query as the first operator on that table?

We accepted EITHER (userID, country) OR (email) – depending on your interpretation of whether "projected out" means to keep the fields or remove them.

b) (3 points) Note that the query has two joins (Order ⋈ User and Order ⋈ Product) and they can be done in any order. One way to decide the order is using statistics on the number of values in each table. Suppose that there are U users, P products, O orders, and C countries in the data in total. Also suppose that each product has a unique name, and that each user and product appears in at least one order. Fill in the following expressions for the number of tuples and distinct values in each relation, using the T/V notation in class, in terms of U, P, O and C:

T(User): ____U___	T(Order): ____O____	T(Product): ____P____
V(User, userId): ___U____	V(Order, userId): __U__	V(Product, prodId): ____P____
V(User, country): ____C___	V(Order, prodId): ____P__	V(Product, name): ____P____

c) (6 points) Now, assume that users are distributed uniformly across countries, and that orders are distributed uniformly across users and products (and there is no correlation between the user and product for a given order). Also assume preservation of value sets. Write estimates for the following statistics for various intermediate tables we could build during this query:

$T(\sigma_{country=c}(User))$: __U/C__
$V(\sigma_{country=c}(User, userId))$: __U/C__

$T(\sigma_{name=n}(Product))$: __1____
$V(\sigma_{name=n}(Product, prodId))$: __1____

$T(\sigma_{country=c}(User) \bowtie Order)$: __O/C__ (computed as O * (U/C) / max(U, U/C) = O/C)
$V(\sigma_{country=c}(User) \bowtie Order, prodId)$: __P____ (by preservation of value sets)

$T(\sigma_{name=n}(Product) \bowtie Order)$: __O/P___ (computed as O * 1 / max(P, 1) = O/P)
$V(\sigma_{name=n}(Product) \bowtie Order, userId)$: __U____ (by preservation of value sets)

$T(\sigma_{country=c}(User) \bowtie Order \bowtie \sigma_{name=n}(Product))$: __O/(CP)__ (= (O/C) * (1) / max(P, 1))

d) (4 points) Suppose that we perform the joins using hash joins, and therefore model the cost of joining two tables as the *sum* of their sizes (i.e. $T(table_1) + T(table_2)$). Ignore the costs of the selections and projections in the query plan, because we'll always do them. What will be the cost of the joins for each of the following two plans?

i. $(\sigma_{country=c}(User) \bowtie Order) \bowtie \sigma_{name=n}(Product)$: U/C + O + O/C + 1
(adding the T() sizes of the tables in each join)

ii. $\sigma_{country=c}(User) \bowtie (Order \bowtie \sigma_{name=n}(Product))$: U/C + O/P + O + 1
(adding the T() sizes of the tables in each join)

Write a simple expression in terms of U, P, O, and/or C that says when plan i above will be cheaper than plan ii:

Plan i is cheaper when: _____C > P_____________
(the difference between the costs is (O/C) - (O/P), so plan i is cheaper when O/C < O/P)

e) (2 points) The analysis above assumes that orders are placed uniformly across users, products, and countries, but real data is often skewed. Intuitively, using the same cost model for hash joins as above, which of the query plans do you think will be better in each of the following cases? Feel free to briefly explain your answer.

Popular country: if the majority of users (>90%) are in the country we query (<c>), which plan do you expect to be cheaper?

Plan i [✓ Plan ii]

Popular product: if the majority of orders are for the product we query (<n>), which is cheaper?

[✓ Plan i] Plan ii