McMaster University
CS/SE 4AD3 Winter 2024
Assignment 1
Due: Feb 8th, 2024 at 10:00pm

January 24, 2024

Part I: Index Modifications (40 marks)

1. Assuming there are 4 slots per (index and leaf) node, and no empty slots are required for future insertions, bulk load a B+ tree index with the following values: {10, 23, 29, 30, 34, 40, 46, 49, 54, 59, 70, 75}. Then perform the following operations, in the given order:

   i) Insert 80
   ii) Insert 24
   iii) Delete 70

   a) Show the resulting index after step 1, and each of (i) through (iii).

   Now return to Step 1, and perform the following operations on your original B+ bulk-loaded tree (in the given order):

   iv) Delete 70
   v) Insert 24
   vi) Insert 80
   vii) Insert 35
   viii) Insert 85
   ix) Delete 59
   x) Delete 54

   b) Show the resulting index after each step (iv) through (x).

2. How many I/Os will you incur to find the entry with search key 80? To determine that 79 is not in the data? To find the entry with search key 40? Provide justification for your answers.

Draw all your indices using a drawing tool software. No handwritten solutions (no photos of handwritten solutions) will be accepted. Include your answers in a file named `PartOneTwo.pdf`.

Part II: Key Compression (10 marks)

1. During lectures, we reviewed prefix key compression as an optimization strategy to increase the fanout of an index, i.e., the number of keys in an index node. An alternative solution is to consider *suffix key compression*, when all index keys have a large common prefix such as {*Daniel L, Daniel P, Danny, Danube*}, and to use *Dan* as a header for the node, and leave only the compressed suffix as keys for the pointers. Figure 1 shows an example. When would this be useful in practice? Provide two examples of indexes showing your suffix key compression, and briefly describe its usage. Again, draw all indices using a drawing tool, no handwritten solutions. Include your answers in the file `PartOneTwo.pdf`.

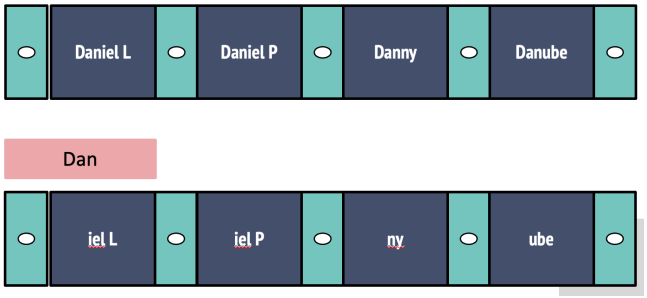

Figure 1: Example: suffix key compression.

Part III: Workload Performance Tuning (50 marks)

Background

The Consumer Electronics Show (CES) is an international global consumer electronics tradeshow held annually in Las Vegas, Nevada, USA. You will work with a smaller version of data replicating similar information to CES. Below you will find a description of the database schema, entity-relationship (E-R) diagram, and a list of queries to help determine profitability, popular products, popular vendors, and attendee interests and demographics. The CES event (henceforth called "the event") consists of vendors, products and persons.

A vendor has a unique identifier, name, street, city, postal code, telephone number, and email contact. There may be many vendors at the event. A product category consists of a set of products that share similar functionality and characteristics (e.g., cameras, televisions, wireless devices, wearable devices). A product sub-category is a specialization of products that share similar characteristics within a product category. For example, the following are examples of sub-categories within the *televisions* category: plasma, LED, LCD, OLED. Each product sub-category contains an ID, name, and description. A product category may have many sub-categories. A vendor sells products, and a product has an id, model number, name, a brief description, the product category ID, sub-category ID, and the retail price.

A person at the event may be an attendee, a vendor representative, or a keynote speaker. For each person, the event assigns a unique person ID, and records the person's last name, first name, telephone number, street, city, postal code, gender, date of birth, and email address. Attendees are persons who attend the event to learn more about the latest products. A vendor representative has a unique ID, and

their vendor affiliation is recorded. The database records a unique keynote speaker ID, their job title, affiliation, the keynote speech title, and the speaker's area of expertise (e.g., wireless devices).

Attendees can attend and listen to keynote speeches. Attendees also have the opportunity to try out and interact with a vendor's latest products. For each attendee to product trial, there exists a unique attendee, product, and vendor information, along with the amount of time (in minutes) that the attendee tried out the specific product. The attendees' email address is also captured in case the vendor wants to follow up with the attendee in the future.

Attendees also have the opportunity to purchase products at the event. Once attendees arrive, they have a choice of whether to accept a passcard or not. If they accept, they can load the passcard with Canadian dollars. The initial amount of money loaded on an attendee's passcard is recorded in their profile. For example, Jane can load $100 on her passcard on the first day of her visit. Passcards cannot be shared. Each passcard contains a unique serial ID, the passcard balance, and the date the card was (re-)loaded. The balance on the passcard cannot go below zero.

As attendees purchase products, a transaction is generated. Each transaction contains an ID, information of who purchased what product, the purchased (product) quantity, the total amount spent in CAD, the payment method (e.g., cash, credit, debit), and the date/time of the purchase. Every product comes with a warranty, which includes an ID, the type of warranty (standard or extended), and the duration of the warranty. Vendors provide a standard warranty for each purchased product. Attendees may optionally purchase an extended warranty. Each warranty covers exactly one product.

Tasks

In the Avenue assignment folder you will find the following files:

- `ER.pdf`: an Entity-Relationship (ER) diagram describing the database.
- `createTables.ddl`: the SQL DDL statements to create all necessary database tables.
- `loadData.ddl`: a set of INSERT statements to populate the database.
- `dropTables.ddl`: a set of DROP TABLE commands, if needed.
- `queries.sql`: SQL queries that comprise your workload.

1. Login to your DB2 instance (database *db4ad3*), create, and populate your database with the above scripts.
2. Using the Linux *time* command, measure the runtime of your query workload. Record this runtime as your baseline performance.
3. Your task is to define at most **six** non-primary key indexes that will improve your baseline runtime. Using the guidelines discussed in class, create a set of indexes that will minimize runtime for your query workload, and is faster than your baseline time. For each of your indexes:
    - Define the search key attributes.
    - The properties of your index, e.g., type of index, clustered/unclustered, composite, etc.
    - Justification of your choice, i.e., why did you define this index? Which queries will this index benefit?

- Explain any special features you may have used in your CREATE INDEX statements, and provide justification.

4. Provide the CREATE INDEX statements in a file `indexes.ddl`. We will use this to verify the performance improvement of your workload over your baseline runtime. **Please double check to ensure your CREATE INDEX statements execute on the server. Non-executable scripts will receive a mark of zero.**
5. Provide the new (improved) runtime of your query workload, and any considerations you would like us to be aware of when verifying your runtime.

Submit all your typed answers to Part III in a file named `tuning.pdf`.

Grading

This assignment is worth 12% towards your final grade.

Submission

All files are to be submitted using the Avenue system. Please ensure you submit all files with the correct names, as described below:

1. For Part I, Part II: Please include all your answers in a file `PartOneTwo.pdf`.
2. For Part III: Submit files `tuning.pdf, indexes.ddl`.