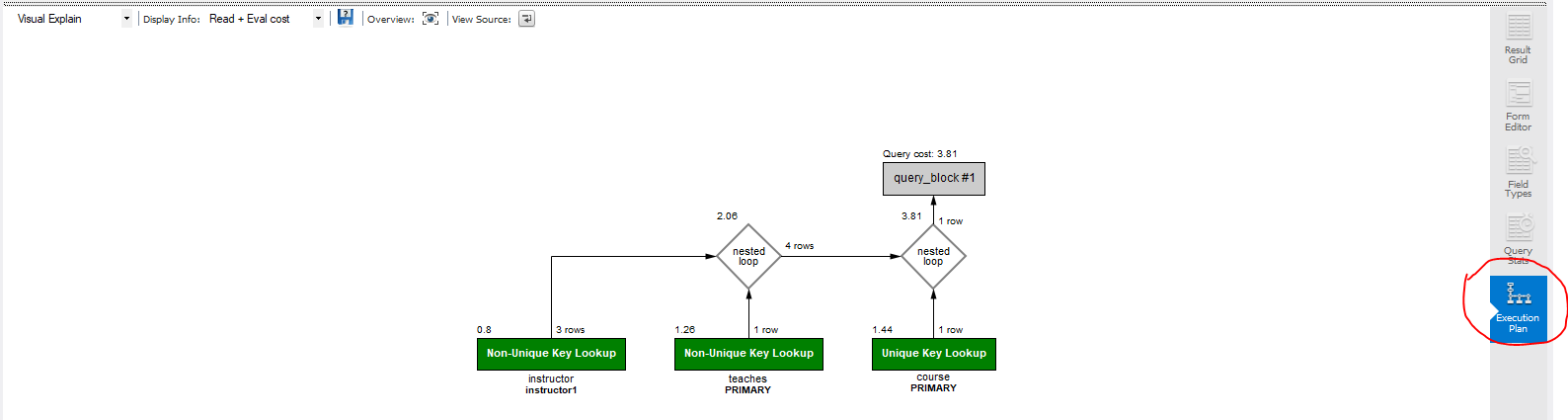
**Query Plan Exercises**

MySQL workbench will show a diagram of the query plan for a SELECT statement if you select the “Execution Plan” icon on the right side of the result panel.



Create the university database by the scripts courses-dll.sql and courses-large.sql.

Then execute the statement and read and study the query plan for the statement

**select name, title**

**from instructor natural join teaches natural join course**

**where instructor.dept\_name='Comp. Sci.';**

**Query 1**

Under the green/red rectangle in the execution plan the index and table name are displayed.

* A “non-unique key lookup” is done when using a non-unique index (such as an index on a foreign key), or a primary key index consisting of multiple columns (such as coursed, sec\_id, semester, year) but only part of the key (course\_id) is specified.
* A “unique key lookup” is done using a unique or primary key index and all columns of the index are specified.
* A “full table scan” is done when the entire table is read and no index is used.
* A “full index scan” indicates that a range predicate is being used along with an index to scan many rows of the tables in sequence based on the index columns.

For more details on the query plan diagram, read “Use the Index Luke” at

<https://use-the-index-luke.com/sql/explain-plan/mysql>

Compare the query plan for the statement above, with the following statement

**select name, title**

**from instructor natural join teaches natural join course**

**where salary > 50000;**

Query 2

Create an index on salary column with the statement

**create index instructor2 on instructor (salary);**

1. Now repeat query 2 above. Is the new index on the salary column being used? Explain why the index is or is not being used.

The new index is not being used because it is not the primary key.

Change the predicates to search on salary and dept\_name.

**select salary, dept\_name**

**from instructor natural join teaches natural join course**

**where salary > 50000;**

**select instructor.name, title**

**from instructor natural join teaches natural join course**

**where salary > 80000 and**

**instructor.dept\_name = 'Comp. Sci.';**

Query 3

1. What indexes are being used in query 3?

The indexes being used in query 3 are instructor1 and teaches.

**select \* from instructor where salary > 50000;**

Query 4

1. The query plan for query 4 shows a full table scan and the index on salary is not being used. Why not? What happens if the predicate is changed to salary > 100000 ?

The index being used is instructor2. It's not being used because it's not the primary key. If the predicate is changed to salary >100000 then only salaries over 100000 will be shown

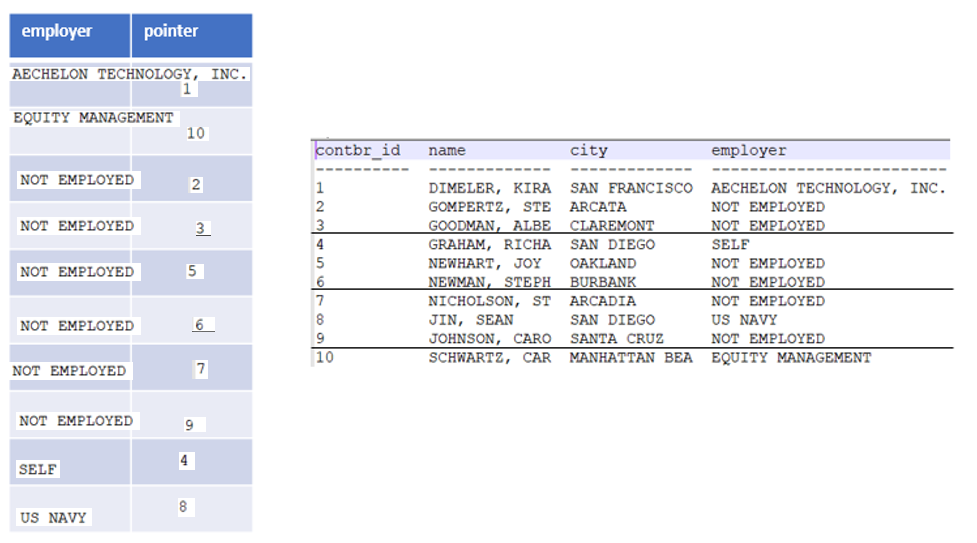
Read about the myth of “slow indexes” at

<https://use-the-index-luke.com/sql/anatomy/slow-indexes>

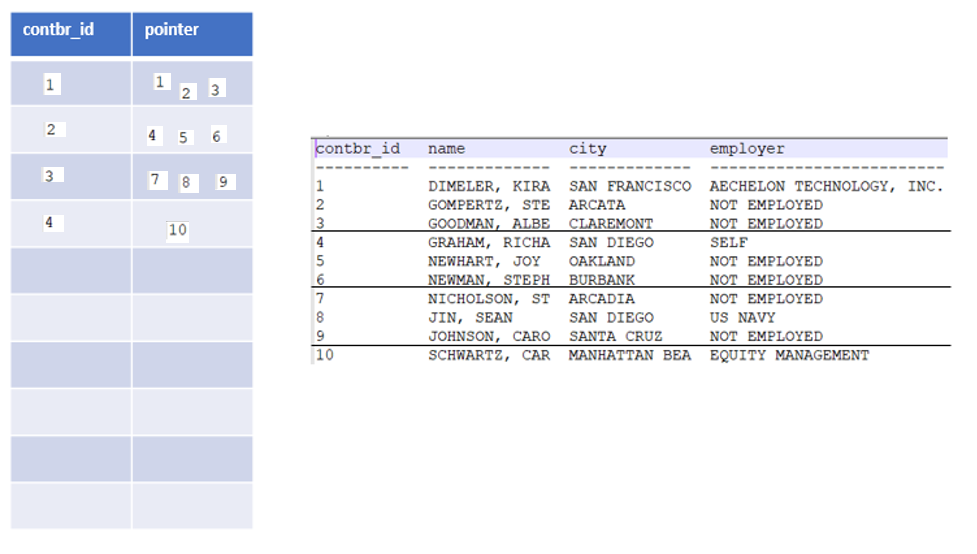
**Index Exercises**

1. Draw a dense ordered index using ‘employer’ as the key. There are 3 rows in each data block of the table file as indicated by the dark lines. Is this index a clustered index?

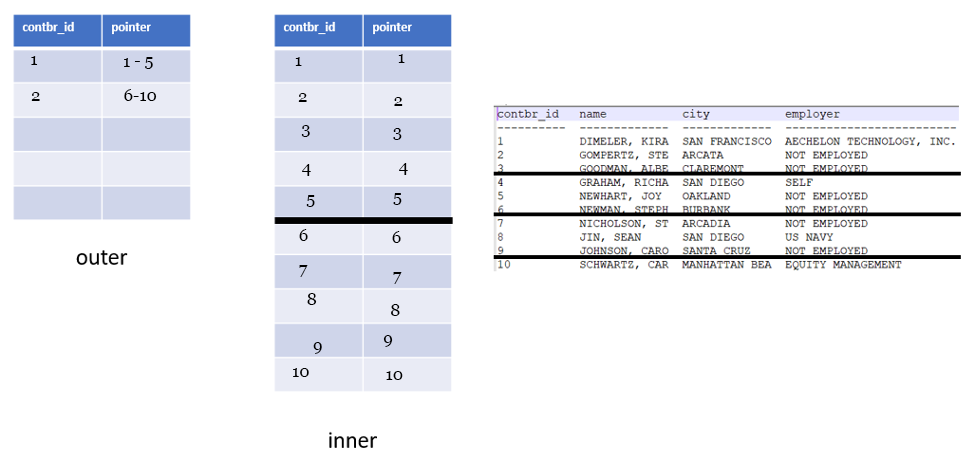
Yes this index is a clustered index, it makes no difference that there is a column with numbers sequentially.



1. Draw a sparse ordered index using ‘contbr\_id’ as key. There should be an index entry for the first row of each data block.



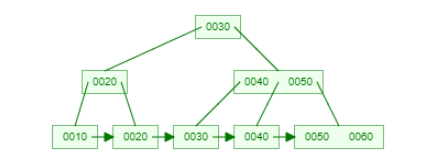
1. Draw a two-level index with ‘contbr\_id’ as key. The inner index is dense and the outer index is (of course) sparse. The inner index has 5 index records per block.

****

**B+ Tree Visualization Exercises**

Use the B+ tree simulator at <https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

* Set MAX DEGREE = 3 Max Degree is the max number of pointers in an internal (not leaf) node. The max number of values in a node is one less than max degree. MAX DEGREE is similar to what we called in lecture FAN OUT. In the simulator we use a small value for MAX DEGREE, but remember in real databases, the FAN OUT is typically on the order of 100.
* Insert the values (one at a time): 10 20 30 40 50 60
* Your diagram should look like



In the diagram above, the leaf node with 0050 0060 is full, as is the parent node 0040 0050. Other nodes are not full.

A B+ tree is efficient for doing key lookup and range queries. However, when new entries have to be inserted or removed from the index due to SQL insert, update or delete statements, there are multiple reads/writes that must be done to maintain the tree nodes in the correct order and the leaf nodes in the correct linked list order.

Pseudo code algorithm to insert a new key.

insert (key) {

start at root, navigate the tree to find the leaf node where key belongs.

if leaf node is not full,

insert key

else

split leaf node and insert key,

insert Parent( first key value of new block, ptr to new block),

}

insertParent( key, ptr) {

if node is not full

insert key , ptr

else if node is root

split root node

create new root

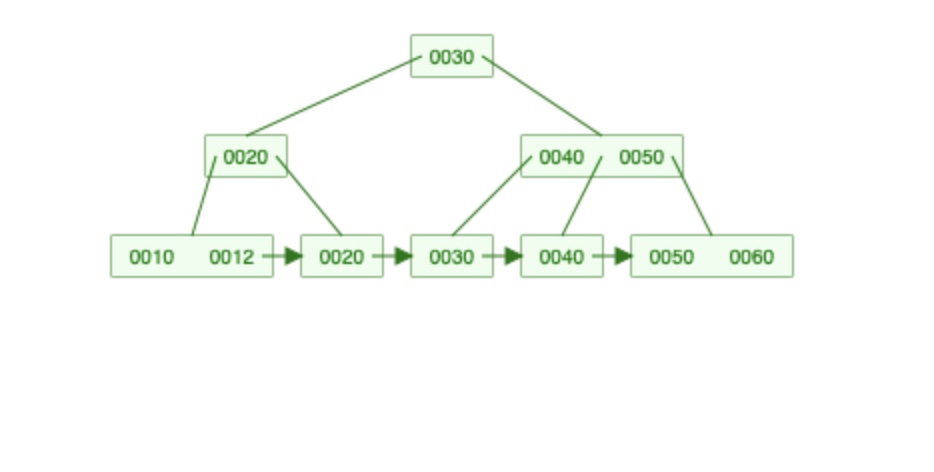
else

split node and insert key,

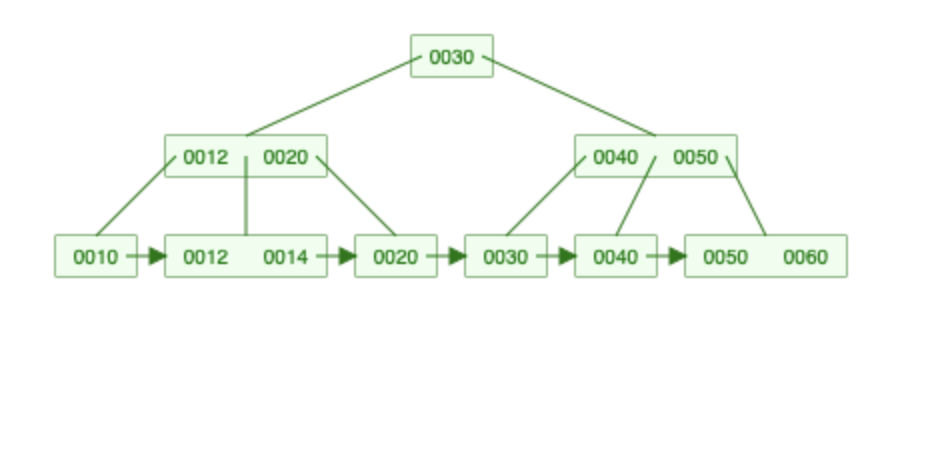
insertParent( first value of new block, ptr to new block),

}

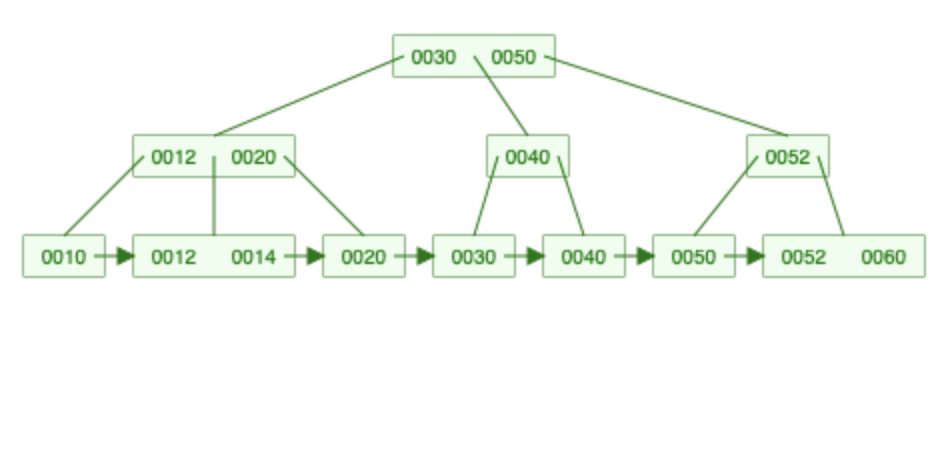
1. Using the pseudo code above and the simulator, draw an updated index diagram (from page 5) after an insert of key value 12.



1. Show an updated diagram after an insert of a key value 14.



1. Show an updated diagram after an insert of a key value 52.



1. Complete the following table. Enter the number of reads/writes in each case.

|  |  |  |
| --- | --- | --- |
|  | reads | writes |
| insert of key 12 | 3 | 1 |
| insert of key 14 | 3 | 2 |
| insert of key 52 | 3 | 3 |

**Concurrency Exercises**

1. **Inconsistent Reads**

The transactions shown below do not use any locking. Transaction 1 reads and writes checking and savings records to do a transfer of $100. Transaction 2 reads checking and savings and incorrectly get a total amount of $1600. Describe what would happens at each time (what data values are read/written to the database and any locks, waits or errors) when **pessimistic locking** is used.

|  |  |  |  |
| --- | --- | --- | --- |
| **Time** | **Transaction 1** | **Transaction 2** | **Database** |
| t1 | read record Checking  as 1000. |  | checking 1000.00  savings 500.00 |
| t2 |  | read record Checking as 1000. |  |
| t3 | read record Savings  as 500 |  |  |
| t4 | subtract $100 from checking  add $100 to savings |  |  |
| t5 | write checking  as 900. |  | checking 900.00  savings 500.00 |
| t6 | write savings  as 600. |  | checking 900.00  savings 600.00 |
| t7 | commit |  |  |
| t8 |  | read record savings  as 600. |  |
| t9 |  | display sum=checking+savings  display 1600. |  |

Transaction one reads that checking is at $1000, the database reflects checking at $1000 and savings at $500. Transaction 2 reads checking is at $1000. Transaction 3 reads savings is at $500. Transaction 4 subtracts $100 from checking and adds $100 to savings but has not yet committed. Transaction 5 writes checking is at $900 and database reflects checking is at $900 and savings is at $500. Transaction 6 writes savings is at $600, database reflects checking is $900, savings is $600. Transaction 7 commits the changes to the database. Transaction 8 reads savings as $600. Transaction 9 displays sum = checking +savings displays 1600. This is because it read the checking as $1000 in transaction 2 before the changes were committed. If pessimistic locking had been used then the transaction wouldn’t have had to roll back.

1. **Inconsistent Writes**

Alice and Bob are both on duty. One of them may go off duty assuming that they first check that the other is still on duty. The following shows the transitions without any locking or versioning. Describe what happens at each time (what data values are read/written to the database and any locks, waits or errors) when **snapshot isolation** is used.

|  |  |  |  |
| --- | --- | --- | --- |
| **Time** | **Transaction Alice** | **Transaction Bob** | **Database** |
| t1 | read records Alice, Bob |  | Alice on  Bob on |
| t2 |  | read records Alice, Bob |  |
| t3 | since Bob is on-duty, update Alice to off-duty | since Alice is on-duty, update Bob to off-duty |  |
| t4 | write record Alice |  | Alice off  Bob on |
| t5 | commit |  |  |
| t6 |  | write record Bob | Alice off  Bob off |
| t7 |  | commit |  |

**Transaction Exercises**

1. Consider this situation: you try to get cash at an ATM, but the ATM fails after updating your account and committing,  but just before cash is dispensed.  As a system designer, how do you cope with this failed transaction?  [ hint:  what do you think “compensating transaction” means? ]

A compensating transaction in SQL is a list of database operations that are able to undo incomplete and inconsistent transactions. They help to get back to the consistent state of the database.

1. Consider this situation: you try to buy an airline ticket at a web site.  The transaction commits on the server, but crashes just afterward, so you cannot see that the purchase has actually been successfully completed.  As a system designer, how would you have the system cope with this situation?

If you haven’t committed the transaction yet you can try and rollback. If you have committed the transaction then you must restore the data from your last backup.

To prevent this in the future you can use set autocommit = 0 and any changes would be kept inside the current transaction until you committed them.

**What to submit for this assignment?**

You should download and edit with file with your answers. You should submit either a Word .docx or a .pdf file.