

Student: Wicaksa Munajat

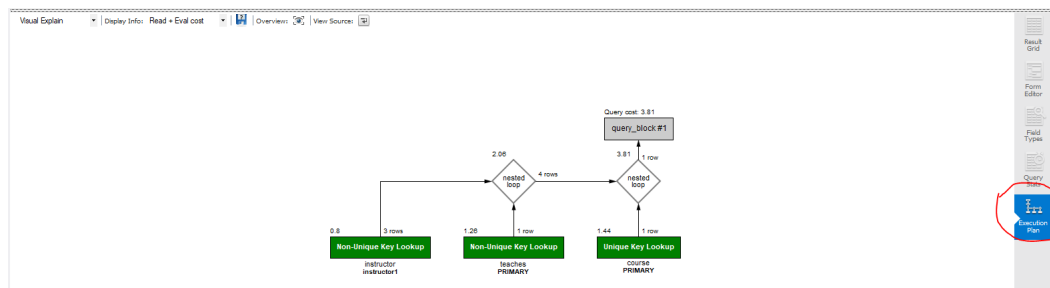
Date: 05/27/2021

Class/Year: CST 363 Summer '21

School: CSUMB CS Online

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 course_id, 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

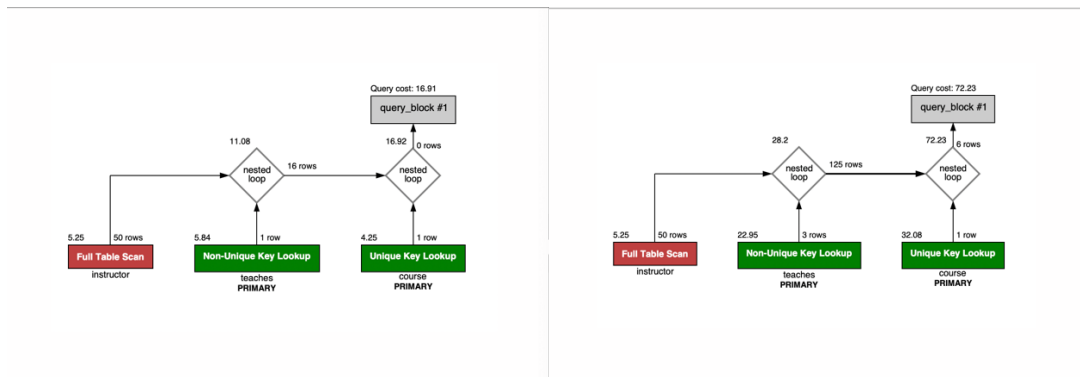
```
select name, title
from instructor natural join teaches natural join course
where salary > 50000;
```

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 salary column being used? Explain why the index is or is not being used.



Answer: The new index on salary column is not being used. Instead, a Full Table Scan is still used because when MySQL plans to do the query, it estimates that it is more efficient to run a full scan vs to use the index that was created. In order for it to use the index, the query should change the predicate that would limit the number of rows returned.

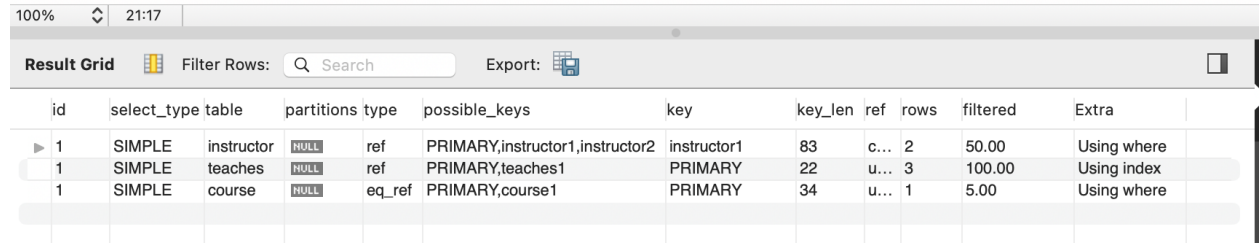
Change the predicates to search on salary and dept_name.

```
select instructor.name, title
from instructor natural join teaches natural join course
where salary > 80000 and
      instructor.dept_name = 'Comp. Sci.';
```

Query 3

2. What indexes are being used in query 3?

```
14
15 • explain select instructor.name, title
16 from instructor natural join teaches natural join course
17 where salary > 80000 and
18     instructor.dept_name = 'Comp. Sci.';
```



id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	instructor	NULL	ref	PRIMARY,instructor1,instructor2	instructor1	83	c...	2	50.00	Using where
1	SIMPLE	teaches	NULL	ref	PRIMARY,teaches1	PRIMARY	22	u...	3	100.00	Using index
1	SIMPLE	course	NULL	eq_ref	PRIMARY,course1	PRIMARY	34	u...	1	5.00	Using where

Answer: According my to MySQL, the indexes being used are instructor1 from the instructor table, and the primary keys for the teaches and courses table.

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

Query 4

3. 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?

Answer: The index is not being used because MySQL determines that using a full table scan instead of a full index scan is less costly than first matching the index and then looking up the rows since the result of the query will return more than half of the possible teacher with salaries above 50000. When you change the salary to > 100,000, MySQL is using an Index Range Scan because the number of teachers making > 100,000 is way less compared to those making 50,000 so instead of doing a linear search, making use of the index is faster and more efficient.

Read about the myth of “slow indexes” at

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

Index Exercises

4. 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?

ANSWER: The index we created is NOT a clustered index because there is an index table that we created that has pointers to the physical rows. In a non-clustered index, the index is stored in one place and the table data is stored in another.

employer	pointer
AECHELON TECHNOLOGY, INC.	
EQUITY MANAGEMENT	
NOT EMPLOYED	
SELF	
US NAVY	

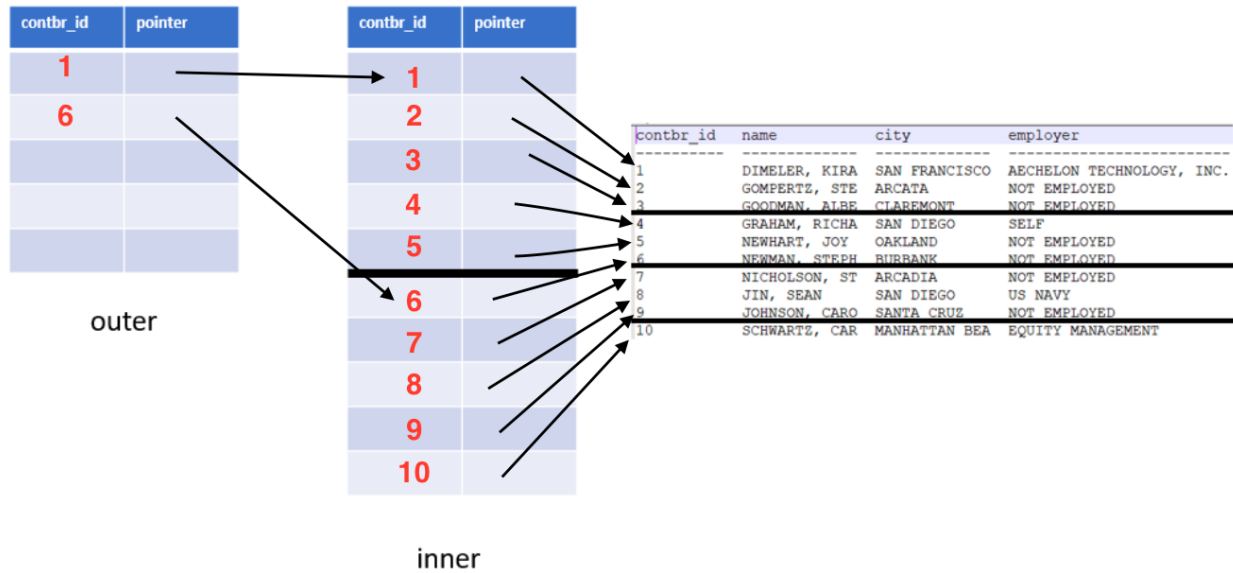
contbr_id	name	city	employer
1	DIMELER, KIRA	SAN FRANCISCO	AECHELON TECHNOLOGY, INC.
2	GOMPERTZ, STE	ARCATA	NOT EMPLOYED
3	GOODMAN, ALBE	CLAREMONT	NOT EMPLOYED
4	GRAHAM, RICHA	SAN DIEGO	SELF
5	NEWHART, JOY	OAKLAND	NOT EMPLOYED
6	NEWMAN, STEPH	BURBANK	NOT EMPLOYED
7	NICHOLSON, ST	ARCADIA	NOT EMPLOYED
8	JIN, SEAN	SAN DIEGO	US NAVY
9	JOHNSON, CARO	SANTA CRUZ	NOT EMPLOYED
10	SCHWARTZ, CAR	MANHATTAN BEA	EQUITY MANAGEMENT

5. Draw a **sparse ordered index** using 'contbr_id' as key. There should be an index entry for the first row of each data block.

contbr_id	pointer
1	
4	
7	
10	

contbr_id	name	city	employer
1	DIMELER, KIRA	SAN FRANCISCO	AECHELON TECHNOLOGY, INC.
2	GOMPERTZ, STE	ARCATA	NOT EMPLOYED
3	GOODMAN, ALBE	CLAREMONT	NOT EMPLOYED
4	GRAHAM, RICHA	SAN DIEGO	SELF
5	NEWHART, JOY	OAKLAND	NOT EMPLOYED
6	NEWMAN, STEPH	BURBANK	NOT EMPLOYED
7	NICHOLSON, ST	ARCADIA	NOT EMPLOYED
8	JIN, SEAN	SAN DIEGO	US NAVY
9	JOHNSON, CARO	SANTA CRUZ	NOT EMPLOYED
10	SCHWARTZ, CAR	MANHATTAN BEA	EQUITY MANAGEMENT

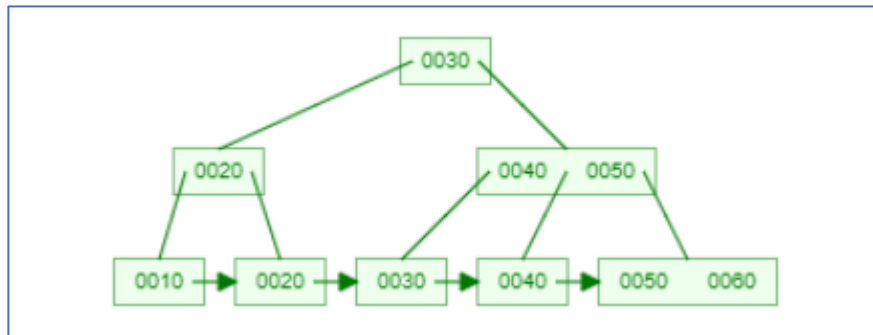
6. Draw a **two-level index** with 'contbr_id' as key. The inner index is dense and the outer index is (of course) sparse. The inner index has 5 index records per block.



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.

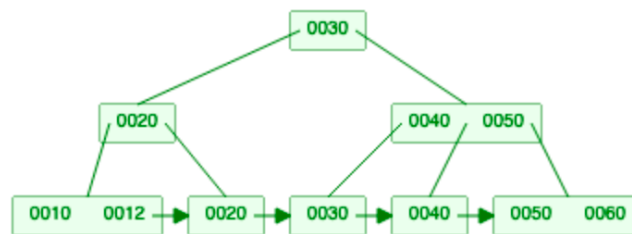
```

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),
}

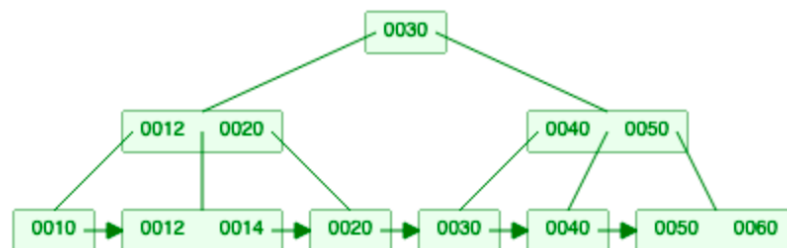
insertParent( key, ptr) {
    if node is not full
        insert key , ptr
    else if node is root
        split root node
        create new root
    else
        split node and insert key,
        insertParent( first value of new block, ptr to new block),
}

```

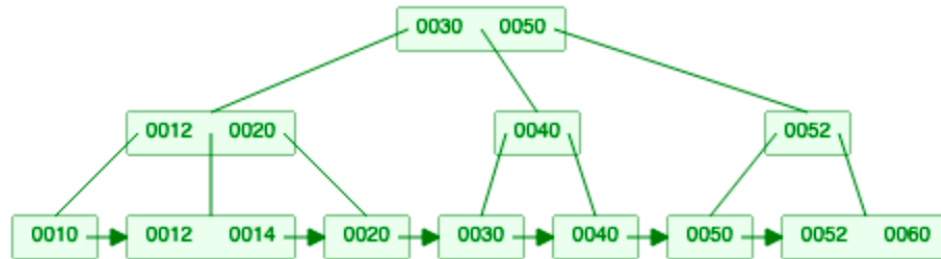
7. Using the pseudo code above and the simulator, draw an updated index diagram (from page 5) after an insert of key value 12.



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



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



10. 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	1

Concurrency Exercises

11. 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 happen 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.	

ANSWER:

T1: As T1 reads the data from checking, it will obtain an S-Lock on the record. Will read checking = \$1000.00.

T2: Concurrently, as T2 reads the same data from checking, it will also obtain an S-Lock. Will read checking = \$1000.00.

T3: As T1 reads the data from savings, it will obtain an S-Lock on the record. Will read savings = \$500.00.

T4 – T6: When T1 tries to subtract \$100 from checking, it will need to upgrade the current S-Lock to an X-Lock. This, however, can't happen since T2 still has an S-Lock on the same data. This will cause a deadlock and T2 is aborted, which releases the S-Lock on T2.

T7: T1 is now able to upgrade from an S-Lock to a X-Lock and perform the write operation.

T8-T9: T2 will try again later and successfully be able to read and display the correct data.

12. 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	

Answer: Each record will have a time stamp of a version number, which gets incremented whenever the record is updated and committed. We start with the original database values of Alice = ON and Bob = OFF and a TS value of 1. When Ta is in the process of committing, it will first check if the version number is still 1 (which it is) If this is true, it will update the value of Alice to OFF and increment the version number to 2 (TS = 2). Now, when Tb is going to write, it will first need to check if the version number is still 1. Since this is not true, this tells Tb that a concurrent program has updated the record between the time that the data was read and going to be written for Tb. In this instance, Tb will abort.

T1: Ta reads value of Alice = ON, Bob = ON, TS = 1

T2: Tb reads value of Alice = ON, Bob = ON TS = 1

T3: Ta updates Alice = OFF

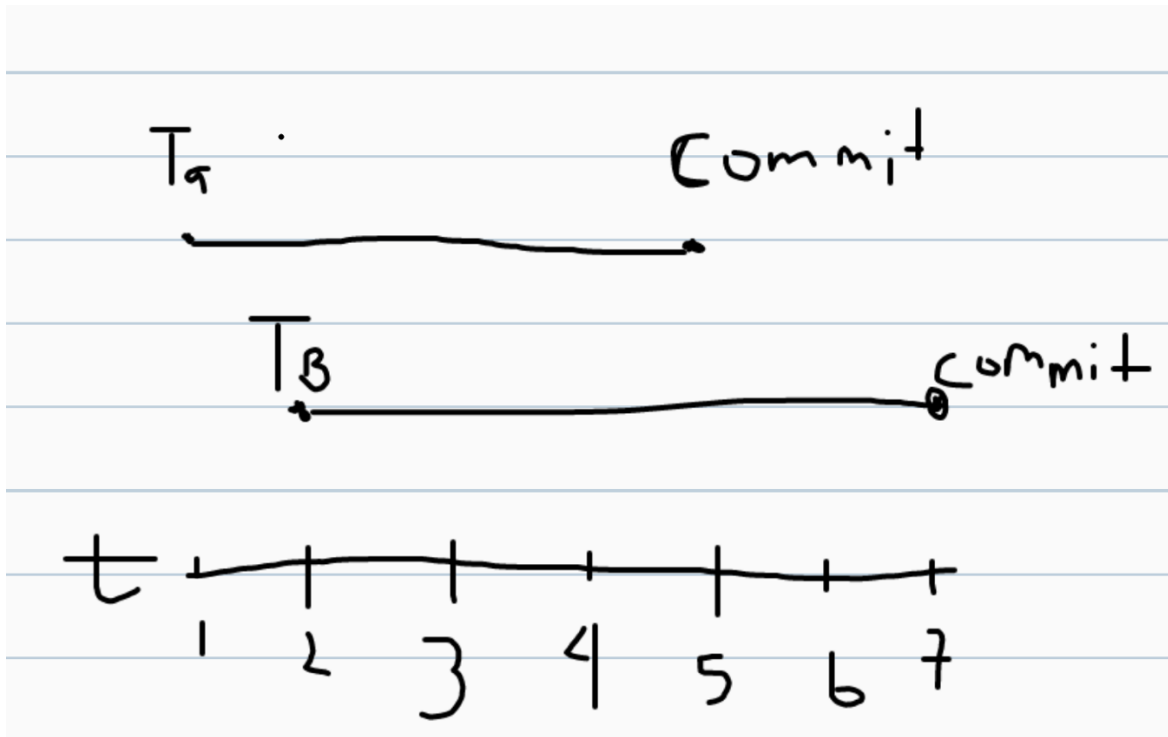
Tb updates Bob = OFF

T4: Ta writes Alice record

T5: Ta COMMITS, Now Alice = OFF, TS = 2

T6: Tb checks if TS = 1, if it's not, abort

T7: Tb aborts



Transaction Exercises

13. 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?]

Answer: First of all, in designing this system, I would have had a system set up for the backup and recovery of data to ensure that in case data is lost during manipulation. For instance, implementing a recovery log/checkpoint or creating a mirrored backup in order to be able to roll back updates in case things go wrong are both good ways to ensure that the database is backed up and can be recovered in case there is a mishap.

However, in this failed transaction, compensating transaction means that you will be trying to undo the changes using explicit database operations. For instance, since you have lost money and not received it, the compensating transaction would simply be re-inserting that amount that you lost back into the database.

This however, does not guarantee that you will be restoring the database to its previous state before it fails since it is likely that other transactions might still be in execution.

14. 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?

Answer: One of the ACID characteristics of databases is durability which states that updates are not lost when a crash occurs. Another is the fact that it is all or none meaning that it will either commit or not commit.

These transactions will normally be written into a recovery log. Checkpoints will be created after a certain amount of time/after a certain amount of data has been recorded. When a checkpoint is made, the updates written to the recovery log get written onto the db on the disk and the recovery log is cleared for new entry.

If the system crashes, the database server after restarting will check the recovery log for transactions that are pending. For each incomplete transaction, it will either remove, reattempt, or roll back the transaction to a previous state.

So in this instance, after the server crashes after committing, the durability of the database system guarantees that the committed transaction will survive permanently because you can implement durability by writing transactions into a log that can be reprocessed or recreated later in case of failure.