

The homework is due on Nov.7 (Thursday). Each student must submit one copy of the homework and one copy of the spool file.

The following shows the file name format:

- ☐ Zno_FirstName_LastName_HW02.pdf
- ☐ Zno_FirstName_LastName_HW02_spool.txt

Example)

Z00000001_KwangSoo_Yang_HW02.pdf

Z00000001_KwangSoo_Yang_HW02_spool.txt

- ☐ The pdf file should contain all answers for both Question and Lab Parts.
- ☐ Please do not use the zip file to upload the files. Upload two separate files.
- ☐ After submitting the homework, please download it and see if the submission is correct.
- ☐ Late submissions are not accepted after two days (i.e., Nov. 9 (Saturday)).
- ☐ Late penalty
 - 1st day: You lose -25 points.
 - 2nd day: You lose -50 points.

Question Part (50 points)

1. (24 points) Consider the following relations, where the primary keys are underlined.

- ☐ Employee (eid, ename, age, sup_eid)
- ☐ WorkOn (eid, pid, date)
- ☐ Project (pid, pname, budget, lid)
- ☐ Location(lid, street_name, city, state, zip_code)

Employee.sup_eid refers to Employee.eid, WorkOn.eid refers to Employee.eid, WorkOn.pid refers to Project.pid, and Project.lid refers to Location.lid.

Let E be Employee, let W be WorkOn, let P be Project, and let L be Location.

Write the relational algebra for the following queries.

1) (3 points) List the names of employees who worked on the 'Database' project.

π ename (σ pname = 'Database' (P \bowtie W \bowtie E))

2) (3 points) List the names of employees who worked on the project located in the city of 'Boca'.

π ename (σ city = 'Boca' (L \bowtie P \bowtie W \bowtie E))

- 3) (3 points) List the names of employees who worked on the 'Database' project, but never worked on the 'Data Mining' project.

$$\pi_{ename} (\sigma_{Pname = 'Database'} (P \bowtie W \bowtie E))$$

$$- \pi_{ename} (\sigma_{Pname = 'Data Mining'} (P \bowtie W \bowtie E))$$

- 4) (3 points) List the names of employees who never worked on the project located in the city of 'Boca'.

$$\pi_{ename} (E)$$

$$- \pi_{ename} (\sigma_{city = 'Boca'} (L \bowtie P \bowtie W \bowtie E))$$

- 5) (3 points) List the names of employees who worked on the same project as their supervisor.

$$\pi_{ename} (E \bowtie E.sup_cid = E.cid (W \bowtie P))$$

- 6) (3 points) List the names of employees who worked on the same project on different dates.

$$\pi_{ename1, ename2} (P \text{ ename1} \rightarrow \text{ename2} \\ (\omega1 \bowtie \omega2 \bowtie P) \bowtie (\sigma_{\omega1.date \neq \omega2.date}))$$

- 7) (3 points) List the names of employees who worked on all the project located in the city of 'Boca'.

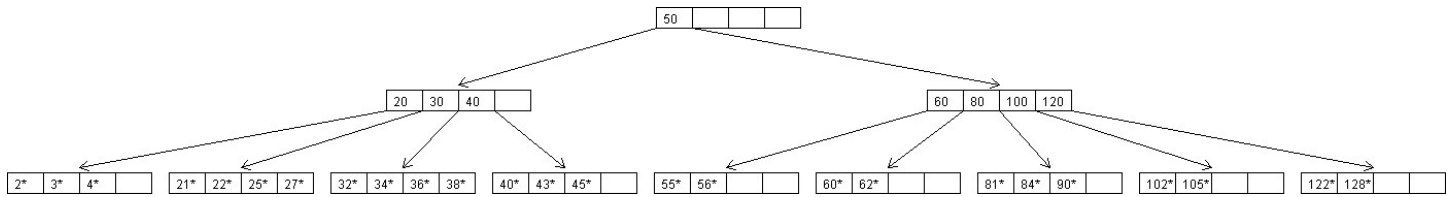
$$\pi_{ename} (E)$$

$$\swarrow \pi_{pid} (\sigma_{city = 'Boca'} (L \bowtie P))$$

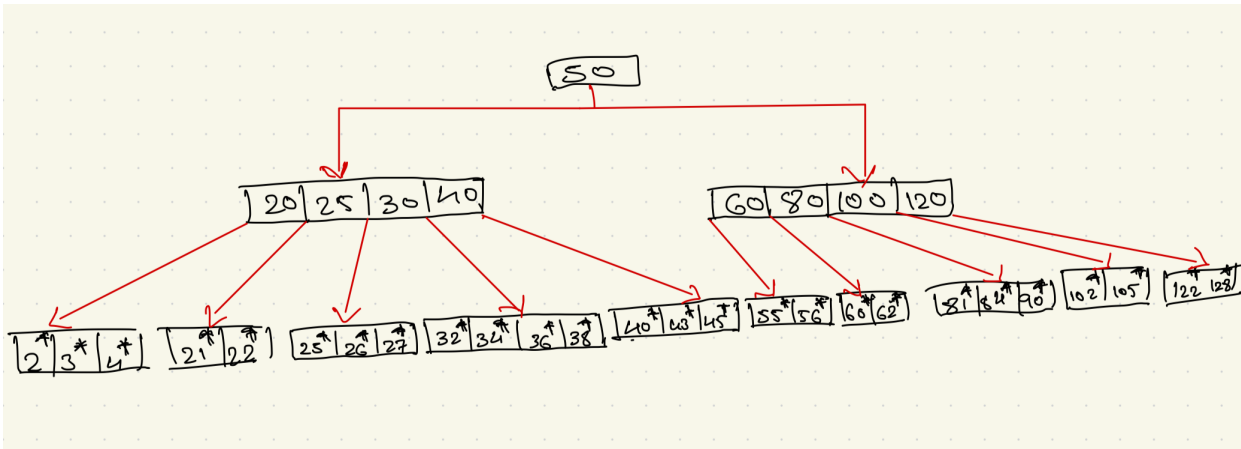
- 8) (3 points) (Do not use the aggregation function) List the names of the oldest employees.

$$\pi_{ename} (\sigma_{age = (\pi_{max(age)} (E))})$$

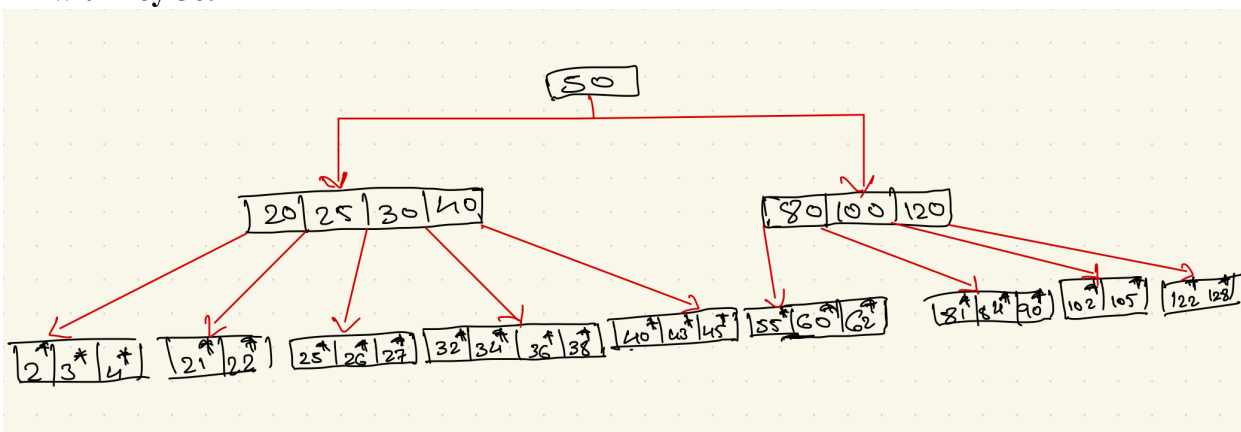
2. (8 points) Consider the following B+ index (Use 2-3 split rule).



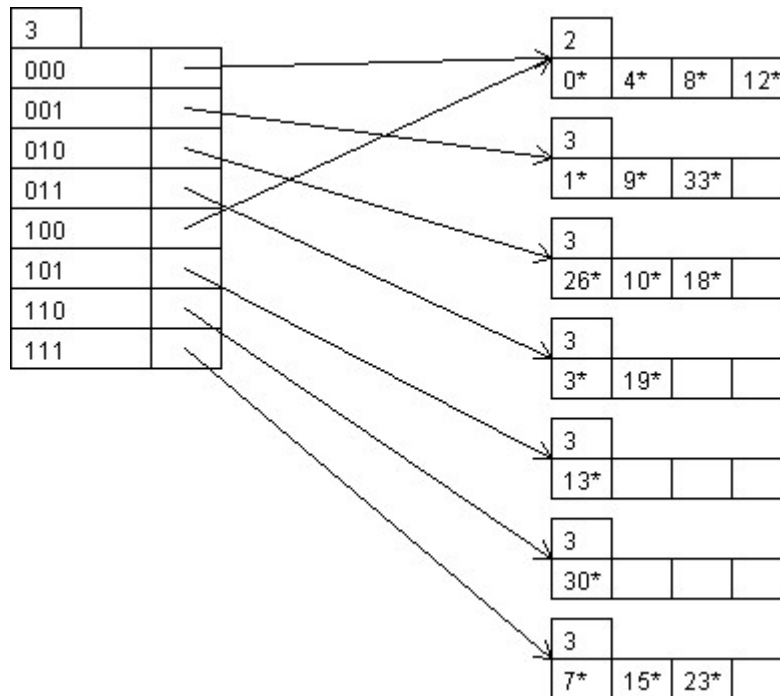
a) (4 points) Show the B+ tree that would result from inserting a data entry with key 26.



b) (4 points) Given the result of (a), show the B+ tree that would result from deleting a data entry with key 56.



3. (6 points) Consider the following extendable hashing index.



a) (3 points) Show the index that would result from inserting a data entry with key 16.

1. **Determine the Binary Representation:**

- Assuming that the hashing function converts keys into binary form and the global depth of the directory is 3 bits, let's find the binary representation of Key 16.
- Key 16 in binary is 10000. To determine the bucket number, we look at the last 3 bits (000) of the key. Since 16 maps to bucket 000, we can proceed.

2. **Check Bucket Capacity:**

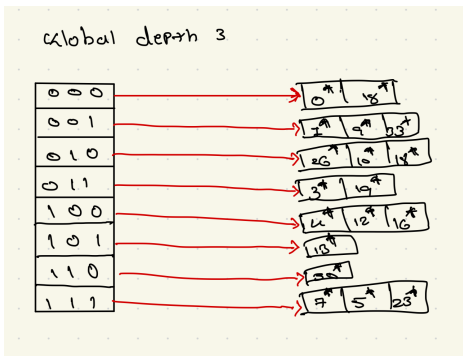
- Bucket 000 currently contains entries [0*, 4*, 8*, 12*]. However, since this bucket already has 4 entries, inserting 16 would exceed its capacity (assuming a maximum of 4 entries per bucket). Therefore, we need to **split** this bucket.

3. **Splitting Bucket 000:**

- To split the bucket, we increase the local depth of bucket 000 from 2 to 3. This means that the bucket can now hold 8 entries instead of 4.
- Next, we distribute the entries based on the new local depth (3 bits):
- Entries with 000 as the last 3 bits remain in bucket 000: [0*, 8*].
- Entries with 100 as the last 3 bits go to a new bucket 100: [4*, 12*, 16*].

4. **Update the Directory:**

- Since we only increased the local depth for bucket 000, the global depth remains 3. The directory now points to the new bucket 100 for entries that end in 100.



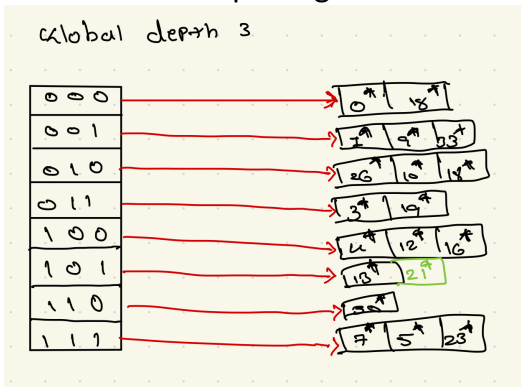
b) (3 points) Given the result of (a), show the index that would result from inserting a data entry with key 21.

1. **Determine the Binary Representation:**

- Key 21 in binary is 10101. The last three bits (101) correspond to bucket 101.

2. **Check Bucket Capacity:**

- Bucket 101 currently contains [13*]. Since it has space (only one entry), we can insert 21 directly without splitting the bucket.



4. (6 points) Consider the following schedules.

S: R3(B);W2(B);R1(C);W1(C);R1(B);R1(C);W3(C);R3(C);R3(A);R3(A);C1;C3;C2

☐ Draw the timetable and precedence graph for the schedule. Is the schedule conflictserializable?

Timetable of Operations:

1. T3: R(B)
2. T2: W(B)
3. T1: R(C)
4. T1: W(C)
5. T1: R(B)
6. T1: R(C)
7. T3: W(C)
8. T3: R(C)
9. T3: R(A)
10. T3: R(A)
11. T1: C
12. T3: C

13. T2: C

conflict on B!

$\hookrightarrow T_3 : R(B) \& T_2 : W(B) \rightarrow T_3 \rightarrow T_2$

$\hookrightarrow T_2 : W(B) \& T_1 : R(B) \rightarrow T_2 \rightarrow T_1$

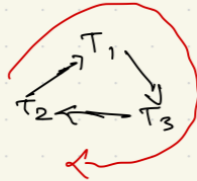
conflict on C!

$\hookrightarrow T_1 : W(C) \& T_3 : W(C) \rightarrow T_1 \rightarrow T_3$

$\therefore T_3 \rightarrow T_2$

$T_2 \rightarrow T_1$

$T_1 \rightarrow T_3$



The schedule is not conflict-serializable because the precedence graph contains a cycle. A cyclic precedence graph implies that the schedule cannot be transformed into a serial schedule without violating the order of conflicting operations.

Is the schedule recoverable?

To verify if the schedule is recoverable, we must ensure that no transaction commits before all transactions it depends on have committed.

- In this schedule:
- T1 depends on T2 because of the conflict on B.
- T2 commits after T1, which could violate recoverability.
- T3 also has conflicts that would need to be checked.

Based on the dependency order and commits, the schedule is **not recoverable** because transactions commit in an order that could lead to cascading aborts if one fails.

5 (6 points) Consider the following schedule.

S: W3(B);W3(C);R3(C);W3(C);W3(B);W1(C);W2(A);R1(C);R2(A);W1(C);C2;C1;C3

Assume that Strict 2PL is applied to the schedules. Draw both deadlock detection table (including shared and exclusive locks) and wait-for-graph. Does the schedule have a deadlock?

1. T3: W(B) → Exclusive Lock on B
2. T3: W(C) → Exclusive Lock on C
3. T3: R(C) → Shared Lock on C (conflict because it already has an exclusive lock from T3)
4. T3: W(C) → Exclusive Lock on C
5. T3: W(B) → Exclusive Lock on B (no conflict)
6. T1: W(C) → Exclusive Lock on C (conflicts with T3)
7. T2: W(A) → Exclusive Lock on A
8. T1: R(C) → Shared Lock on C (conflicts with T3 and T1)
9. T2: R(A) → Shared Lock on A (already locked by T2 as exclusive)

10. T1: W(C) → Exclusive Lock on C (conflicts with T3)

deadlock detection Table !

Transaction	Req. Item	lock type	status	Notes
T ₃	B	X	acquired	1 st req
T ₃	C	X	acquired	2 nd req
T ₃	C	S	Waiting	conflict
T ₃	C	X	Cx	X
T ₃	B	X	Cx	NO conflict
T ₁	C	X	W	conflict
T ₂	A	X	Cx	1 st req
T ₁	C	S	W	conflict
T ₂	A	S	W	conflict
T ₁	C	X	W	conflict

Based on the deadlock detection table, we construct the wait-for graph:

- T1 waits for T3 (on C).
- T2 waits for T2 itself (on A), but this is essentially a lock on itself, so it doesn't create a new edge.
- T3 is not waiting on any other transaction.

There's a cycle here, indicating a **deadlock**.

Lab Part 01 (35 points)

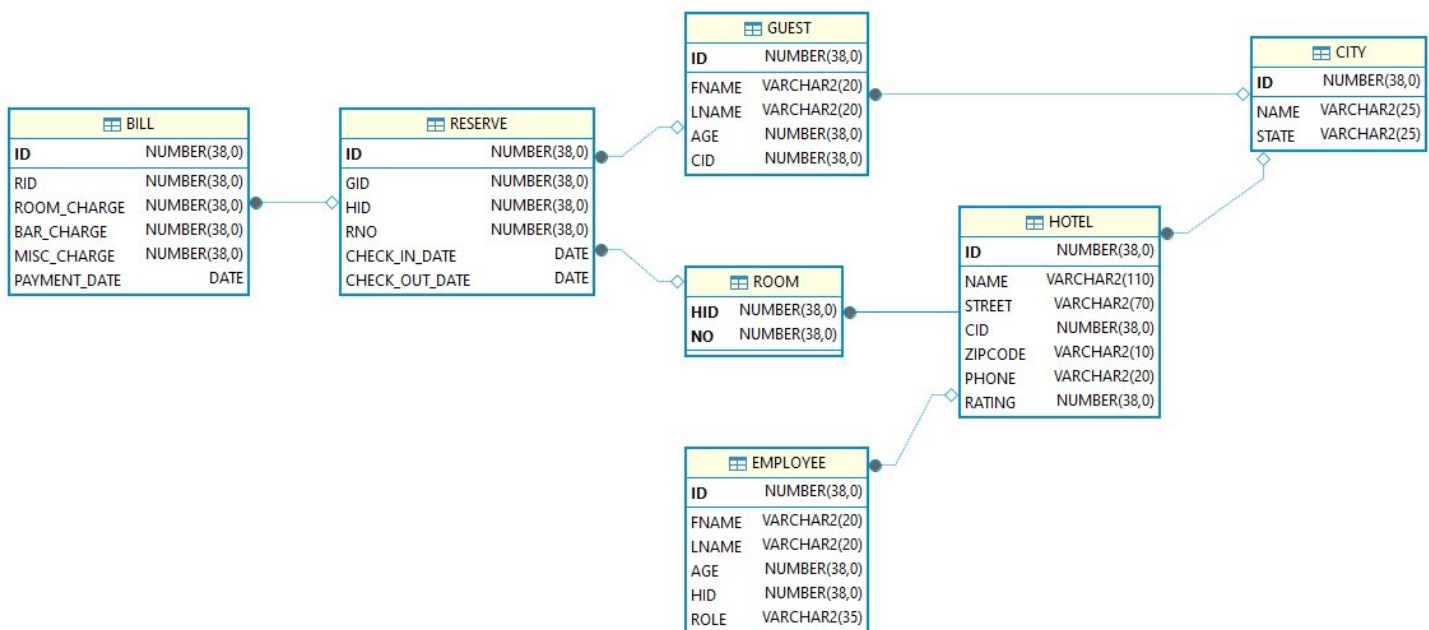
Preliminary

- a. Login into Linux machine (oraclelinux.eng.fau.edu)
- b. Connect to the database (e.g., sqlplus username/password)
- c. **ALTER SESSION SET CURRENT_SCHEMA = COP6731;**
- d. **Please note that the schema name is defined as upper case characters.**
- e. Execute the following SQL and identify all required tables.
 - i. SELECT table_name from all_tables
where owner = 'COP6731'; ☐
Please note that the owner's name is defined as upper case characters.
- f. Change line size and page size:
 - i. SET LINESIZE 400 o SET PAGESIZE 0

Use the “spool” command to create a log file for the output of SQL (e.g., SPOOL filename and SPOOL OFF)

Note: Submit both answers and SPOOL files (i.e., an explanation and a SPOOL file).

Consider the following relational schema.



- ☐ Execute the following SQL and review the indexed keys.
SELECT table_name||', '||index_name||', '||column_name||', '||column_position FROM
all_ind_columns
WHERE table_owner='COP6731'
ORDER BY index_name, column_position;
- ☐ Execute the following SQL and review the index structure.
SELECT index_name || ', ' || index_type || ', ' || blevel || ', ' || leaf_blocks || ', ' || table_name || ', ' ||
avg_leaf_blocks_per_key || ', ' || avg_data_blocks_per_key || ', ' || clustering_factor || ', ' || distinct_keys FROM
all_indexes
WHERE table_owner='COP6731';

Query Evaluation

- ☐ Use the Spool command to log the output of SQL (e.g., SPOOL filename and SPOOL OFF)
- ☐ There are two files you should submit: 1) explanation for query execution plans and 2) recorded spool file.

1. (5 points) Execute the following two SQLs and explain which access method is used in each query. Explain why one outperforms another.

```
SELECT /*+ GATHER_PLAN_STATISTICS */ count(G.id)
FROM guest G
WHERE G.id BETWEEN 1 AND 10;

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));

SELECT /*+ GATHER_PLAN_STATISTICS */ count(G.id)
FROM guest G
WHERE G.age BETWEEN 1 AND 10;

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));
```

2. (5 points) Execute the following two SQLs and explain which access method is used in each query. Explain why one outperforms another.

```
SELECT /*+ GATHER_PLAN_STATISTICS */ count(H.id) FROM
hotel H
WHERE H.name LIKE '%opton Inn'

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));

SELECT /*+ GATHER_PLAN_STATISTICS */ count(H.id)
FROM hotel H
WHERE H.name LIKE 'Hamp%';

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));
```

3. (5 points) Execute the following SQL. How many tables were accessed to produce the outcome, and how many indexes were utilized in the process?

```
SELECT /*+ GATHER_PLAN_STATISTICS */ G.fname
FROM guest G, reserve R, hotel H
WHERE G.id = R.gid AND R.hid = H.id
AND G.id = 1047;

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));
```

4. (10 points) Execute the following SQL and [draw the query tree for the SQL](#). Describe what kinds of access methods are used for each table. Explain which join method is used in each join operation.

```
SELECT /*+ GATHER_PLAN_STATISTICS */ H.name
FROM guest G, reserve R, hotel H, city C
WHERE G.id = R.gid AND R.hid = H.id AND H.cid = C.id
AND C.name = 'Blowing Rock' AND G.fname = 'Darryl' AND G.lname = 'Getty';

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));
```

5. (10 points)-Execute the following SQL and [draw the query tree for the SQL](#). Describe what kinds of access methods are used for each table. Explain which join method is used in each join operation.

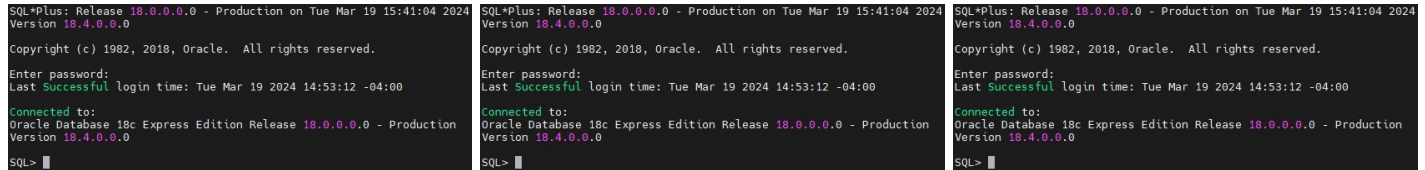
```
SELECT /*+ GATHER_PLAN_STATISTICS */ G.fname
FROM guest G, reserve R, room RM, hotel H, bill B c
WHERE G.id = R.gid AND R.hid = RM.hid AND R.mo = RM.no AND RM.hid = H.id AND R.id = B.rid AND
H.name = 'Michiana Motel' AND B.payment_date='09-JUN-23' AND RM.no = 3;

SELECT * FROM TABLE (DBMS_XPLAN.display_cursor (format=>'ALLSTATS LAST'));
```

Lab Part 02 (15 points)

Preliminary

- ☐ Open three terminals (e.g., putty or XShell) and login Oracle database



- ☐ Assume that each terminal represents each user (i.e., User1 and User2).
- ☐ Create the student table using the following query.

```
CREATE TABLE customer ( id          integer,
age          integer,
                primary key (id) );
```

- ☐ Execute SET AUTOCOMMIT OFF. This command suppresses automatic committing so that you must commit changes manually (Note that # represents the order of the SQL execution).

#	User 1	User 2	User3
1	SET AUTOCOMMIT OFF		
2		SET AUTOCOMMIT OFF	
3			SET AUTOCOMMIT OFF

- ☐ TURN ON the spool.

#	User 1	User 2	User 3
1	SPOOL user1.log		
2		SPOOL user2.log	
3			SPOOL user3.log

- ☐ Execute the SQLs in each question and answer the question.
- ☐ Save the spool files and submit both answers and spool files.

#	User 1	User 2	User 3
1	SPOOL OFF		
2		SPOOL OFF	
3			SPOOL OFF

Transaction Processing and Recovery

1. (5 points) What is the result after STEP 8 and STEP10? Explain why this result makes sense or why it does not make sense?

#	User 1	User 2	User 3
1	DELETE FROM customer;		
2	COMMIT;		
3	INSERT INTO customer (id, age) VALUES (1, 20);		
4		UPDATE customer SET age = age +10;	
5	COMMIT;		
6			UPDATE customer SET age = age +10;
7		COMMIT;	
8			SELECT id, age FROM customer;
9			COMMIT;
10			SELECT id, age FROM customer;

2. (5 points) What is the result of STEP 7? Explain why this result makes sense or why it does not make sense?

#	User 1	User 2	User 3
	DELETE FROM customer;		
	COMMIT;		
1	INSERT INTO customer (id, age) VALUES (1, 20);		
2		INSERT INTO customer (id, age) VALUES (2, 20);	
3	COMMIT;		
4			DELETE FROM customer WHERE id = 1 OR id =2;
5		COMMIT;	
6			COMMIT;
7			SELECT id, age FROM customer;

3. (5 points) What happens after STEP 7 and STEP 8? What is the result of Step 9? Explain why this result makes sense or why it does not make sense?

#	User 1	User 2	User 3
1	DELETE FROM customer;		
2	COMMIT;		
3	INSERT INTO customer (id, age) VALUES (1, 20);		
4		UPDATE customer SET age =30 WHERE id = 1;	
5			INSERT INTO customer (id,age) VALUES (2, 20);
6			INSERT INTO customer (id,age) VALUES (1, 30);
7	INSERT INTO customer (id, age) VALUES (2, 30);		
8		INSERT INTO customer (id, age) VALUES (2, 40);	
9			COMMIT;
10		COMMIT;	
11	COMMIT;		