1. Describe three techniques commonly used when developing algorithms for relational operators. Explain how these techniques can be used to design algorithms for the selection, projection, and join operators.

1. The three techniques commonly used are indexing, iteration, and partitioning:
Indexing: If a selection or join condition is specified, use an index to examine just the tuples that satisfy the condition.
Iteration: Examine all tuples in an input table, one after the other. If we need only a few fields from each tuple and there is an index whose key contains all these fields, instead of examining data tuples, we can scan all index data entries.
Partitioning: By partitioing tuples on a sort key, we can often decompose an operation into a less expensive collection of operations on partitions. *Sorting* and *hashing* are two commonly used partitioning techniques.
They can be used to design algorithms for selection, projection, and join operators as follows:
Selection: For a selection with more than one tuple matching the query (in general, at least 5%), indexing like B+ Trees are very useful. This comes into play often with range queries. It allows us to not only find the first qualifying tuple quickly, but also the other qualifying tuples soon after (especially if the index is clustered). For a selection condition with an equality query where there are only a few (usually 1) matching tuple, partitioning using hash indexing is often appropriate. It allows us to find the exact tuple we are searching for with a cost of only a few (typically one or two) I/Os.
Projection: The projection operation requires us to drop certain fields of the input, which can result in duplicates appearing in the result set. If we do not need to remove these duplicates, then the iteration technique can efficiently handle this problem. On ther other hand, if we do need to eliminate duplicates, partitioning the data and applying a sort key is typically performed.
In the case that there are appropriate indexes available, this can lead to less expensive plans for sorting the tuples during duplicate elimination since the data may all ready be sorted on the index (in that case we simply compare adjacent entries to check for duplicates)
Join: The most expensive database operation, joins, can combinations of all three techniques. A join operation typically has multiple selection and projection elements built into it, so the importance of having appropriate indexes or of partitioning the data is just as above, if not more so. When possible, the individual selections and projections are applied to two relations before they are joined, so as to decrease the size of the intermediate table. As an example consider joining two relations with 100,000 tuples each and only 5 % of qualifying tuples in each table. Joining before applying the selection conditions, would result in a huge intermediate table size that would then have to be searched for matching selections. Alternatively, consider applying parts of the selection first. We can then perform a join of the 5,000 qualifying tuples found after applying the selection to each table, that can then be searched and handled significantly faster.

5. What is the goal of query optimization? Why is optimization important?

5. The goal of query optimization is to avoid the worst plans and find a good plan. The goal is usually not to find the optimal plan. The difference in cost between a good plan and a bad plan can be several orders of magnitude: a good query plan can evaluate the query in seconds, whereas a bad query plan might take days!

9. What role do statistics gathered from the database play in query optimization?

9. The query optimizer uses statistics to improve the chances of selecting an optimum query plan. The statistics are used to calculate reduction factors which determine the results the optimizer may expect given different indexes and inputs.

1. Define the term *functional dependency*

1. Let R be a relational schema and let X and Y be two subsets of the set of all attributes of R. We say Y is functionally dependent on X, written X → Y, if the Y-values are determined by the X-values. More precisely, for any two tuples *r*1 and *r*2 in (any instance of) R

$$\pi X(r1) = \pi X(r2) \Rightarrow \pi Y(r1) = \pi Y(r2)$$

2. Why are some functional dependencies called *trivial?*

2. Some functional dependencies are considered trivial because they contain superfluous attributes that do not need to be listed. Consider the FD: $A \rightarrow AB$. By reflexivity, *A* always implies *A*, so that the *A* on the right hand side is not necessary and can be dropped. The proper form, without the trivial dependency would then be $A \rightarrow B$.

3. Give a set of FDs for the relation schema *R(A,B,C,D)* with primary key *AB* under which *R* is in 1NF but not in 2NF.

3. Consider the set of FD: $AB \rightarrow CD$ and $B \rightarrow C$. *AB* is obviously a key for this relation since $AB \rightarrow CD$ implies $AB \rightarrow ABCD$. It is a primary key since there are no smaller subsets of keys that hold over *R(A,B,C,D)*. The FD: $B \rightarrow C$ violates 2NF since:

- $C \in B$ is false; that is, it *is not* a trivial FD
- *B is not* a superkey
- *C is not* part of some key for *R*
- *B is* a proper subset of the key *AB* (transitive dependency)

4. Give a set of FDs for the relation schema *R(A,B,C,D)* with primary key *AB* under which *R* is in 2NF but not in 3NF

4. Consider the set of FD: $AB \rightarrow CD$ and $C \rightarrow D$. *AB* is obviously a key for this relation since $AB \rightarrow CD$ implies $AB \rightarrow ABCD$. It is a primary key since there are no smaller subsets of keys that hold over *R(A,B,C,D)*. The FD: $C \rightarrow D$ violates 3NF but not 2NF since:

- $D \in C$ is false; that is, it *is not* a trivial FD
- *C is not* a superkey
- *D is not* part of some key for *R*

5. Consider the relation schema *R(A,B,C)*, which has the FD $B \rightarrow C$. If *A* is a candidate key for *R*, is it possible for *R* to be in BCNF? If so, under what conditions? If not, explain why not.

5. The only way *R* could be in BCNF is if *B* includes a key, *i.e. B* is a key for R.

6. Suppose we have a relation schema *R(A,B,C)* representing a relationship between two entity sets with keys *A* and *B*, respectively, and suppose that *R* has (among others) the FDs $A \rightarrow B$ and $B \rightarrow A$. Explain what such a pair of dependencies

means (i.e., what they imply about the relationship that the relation models).
6. It means that the relationship is one to one. That is, each A entity corresponds to at most one *B* entity and vice-versa. (In addition, we have the dependency $AB \rightarrow C$, from the semantics of a relationship set.)

Exercise 19.5 Consider the following collection of relations and dependencies. Assume that each relation is obtained through decomposition from a relation with attributes *ABCDEFGHI* and that all the known dependencies over relation *ABCDEFGHI* are listed for each question. (The questions are independent of each other, obviously, since the given dependencies over *ABCDEFGHI* are different.) For each (sub)relation: (a) State the strongest normal form that the relation is in. (b) If it is not in BCNF, decompose it into a collection of BCNF relations.
1. *R1(A,C,B,D,E)*, $A \rightarrow B$, $C \rightarrow D$
2. *R2(A,B,F)*, $AC \rightarrow E$, $B \rightarrow F$
3. *R3(A,D)*, $D \rightarrow G$, $G \rightarrow H$
4. *R4(D,C,H,G)*, $A \rightarrow I$, $I \rightarrow A$
5. *R5(A,I,C,E)*

Answer 19.5
1. 1NF. BCNF decomposition: AB, CD, ACE.
2. 1NF. BCNF decomposition: AB, BF
3. BCNF.
4. BCNF.
5. BCNF.

Exercise 19.7 Suppose you are given a relation *R* with four attributes *ABCD*. For each of the following sets of FDs, assuming those are the only dependencies that hold for *R*, do the following: (a) Identify the candidate key(s) for *R*. (b) Identify the best normal form that *R* satisfies (1NF, 2NF, 3NF, or BCNF). (c) If *R* is not in BCNF, decompose it into a set of BCNF relations that preserve the dependencies.

Answer 19.7

1. $C \rightarrow D$, $C \rightarrow A$, $B \rightarrow C$

1. (a) Candidate keys: *B*
(b) *R* is in 2NF but not 3NF.
(c) $C \rightarrow D$ and $C \rightarrow A$ both cause violations of BCNF. One way to obtain a (lossless) join preserving decomposition is to decompose R into *AC*, *BC*, and *CD*.

2. $B \rightarrow C$, $D \rightarrow A$

2. (a) Candidate keys: *BD*
(b) *R* is in 1NF but not 2NF.
(c) Both $B \rightarrow C$ and $D \rightarrow A$ cause BCNF violations. The decomposition: *AD*, *BC*, *BD* (obtained by first decomposing to *AD*, *BCD*) is BCNF and lossless and join-preserving.

3. $ABC \rightarrow D$, $D \rightarrow A$

3. (a) Candidate keys: *ABC*, *BCD*
(b) *R* is in 3NF but not BCNF.
(c) *ABCD* is not in BCNF since $D \rightarrow A$ and *D* is not a key. However if we split up *R* as *AD*, *BCD* we cannot preserve the dependency $ABC \rightarrow D$. So there is no BCNF decomposition.

4. $A \rightarrow B$, $BC \rightarrow D$, $A \rightarrow C$

4. (a) Candidate keys: *A*
(b) *R* is in 2NF but not 3NF (because of the FD: $BC \rightarrow D$).
(c) $BC \rightarrow D$ violates BCNF since *BC* does not contain a key. So we split up *R* as in: *BCD*, *ABC*.

5. $AB \rightarrow C$, $AB \rightarrow D$, $C \rightarrow A$, $D \rightarrow B$

5. (a) Candidate keys: *AB*, *BC*, *CD*, *AD*
(b) R is in 3NF but not BCNF (because of the FD: $C \rightarrow A$).
(c) $C \rightarrow A$ and $D \rightarrow B$ both cause violations. So decompose into: *AC*, *BCD* but this does not preserve $AB \rightarrow C$ and $AB \rightarrow D$, and *BCD* is still not BCNF because $D \rightarrow B$. So we need to decompose further into: *AC*, *BD*, *CD*. However, when we attempt to revive the lost functioanl dependencies by adding *ABC* and *ABD*, we that these relations are not in BCNF form. Therefore, there is no BCNF decomposition.

13.4 Consider the relations r1(A, B,C), r2(C, D, E), and r3(E, F), with primary keys A, C, and E, respectively. Assume that r1 has 1000 tuples, r2 has 1500 tuples, and r3 has 750 tuples. Estimate the size of r1 ✶ r2 ✶ r3, and give an efficient strategy for computing the join.

Answer:

• The relation resulting from the join of r1, r2, and r3 will be the same no matter which way we join them, due to the associative and commutative properties of joins. So we will consider the size based on the strategy of ((r1 ✶ r2) ✶ r3). Joining r1 with r2 will yield a relation of at most 1000 tuples, since C is a key for r2. Likewise, joining that result with r3 will yield a relation of at most 1000 tuples because E is a key for r3. Therefore the final relation will have at most 1000 tuples.

• An efficient strategy for computing this join would be to create an index on attribute C for relation r2 and on E forr3. Then for each tuple in r1, we do the following:

a. Use the index for r2 to look up at most one tuple which matches the C value of r1.

b. Use the created index on E to look up in r3 at most one tuple which matches the unique value for E in r2.

13.5 Consider the relations r1(A, B,C), r2(C, D, E), and r3(E, F) of Practice Exercise 13.4. Assume that there are no primary keys, except the entire schema. Let V(C,r1) be 900, V(C,r2) be 1100, V(E,r2) be 50, and V(E,r3) be 100. Assume that r1 has 1000 tuples, r2 has 1500 tuples, and r3 has 750 tuples. Estimate the size of r1 ✶ r2 ✶ r3 and give an efficient strategy for computing the join.

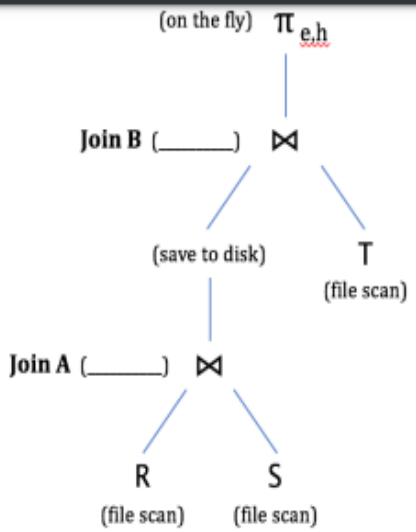
Answer:

The estimated size of the relation can be determined by calculating the average number of tuples which would be joined with each tuple of the second relation. In this case, for each tuple in r1, 1500/V(C,r2) = 15/11 tuples (on the average) of r2 would join with it. The intermediate relation would have 15000/11 tuples. This relation is joined with r3 to yield a result of approximately 10,227 tuples (15000/11 × 750/100 = 10227). A good strategy should join r1 and r2 first, since the intermediate relation is about the same size as r1 or r2. Then r3 is joined to this result.

In the next few problems, we will compute the cost of the query plan on the right, which finds the result of the expression $\pi_{e,h}$ (R ⋈ S ⋈ T) over the tables R(e,f), S(f,g), and T(g,h). (Both joins natural.)

The plan on the right is still *incomplete*, however, because the choice of algorithm for two parts, **Join A** and **Join B**, have not yet been filled in.

In the next few problems (starting on the next page), we will consider the cost of this plan with different choices of algorithms for these two points.



You can assume the following statistics about these tables:

Table	# tuples	# blocks
R	1,000	100
S	5,000	200
T	100,000	10,000
R ⋈ S	5,000	20

Furthermore, you can assume the following statistics about their columns:

Column	# distinct	Low	High
R.f	100	1	1000
S.f	1000	1	2000
S.g	5000	1	2,000
T.g	1000	1	10,000

Useful Formulas

Estimated cost of X join Y:

- Using a block nested loop, the cost is B(X) + B(X) B(Y), where B(X) is # blocks in X.
- Using a clustered index on Y(A), the cost is B(X) + T(X) B(Y) * E, where T(X) is # tuples in X, and E is the selectivity of the condition A = c.
- Using an unclustered index on Y(A), the cost is B(X) + T(X) T(Y) * E.

Estimated selectivity of conditions:

- For A = c, the selectivity is 1 / (# distinct values of A)
- For A < c, the selectivity is (c - lowest value of A) / (highest - lowest value of A)

11. What is the estimated cost of **Join B** in the plan if we implement it with a block nested loop join?

20 + 20 * 10,000 = **200,020**

12. What is the estimated cost of **Join B** the if we use an indexed join? Assume that we have a clustered index on T(g).

20 + 5,000 * 10,000 / 1000 = **50,020**

13. What is the estimated cost of **Join A** in the plan if we implement it with a block nested loop join?

100 + 100 * 200 = **20,100**

14. What is the estimated cost of **Join A** if we use an indexed join? Assume that we have an unclustered index on S(f).

100 + 1,000 * 5,000 / 1000 = **5,100**

15. What is the total cost of this plan if we use the best choice of join algorithm for A and B?

In addition to the two joins above, we need to write R ⋈ S to disk, which costs 20 IOs. The final projection has no cost. Thus, the total cost is 55,140.

14.7 What is a cascadeless schedule? Why is cascadelessness of schedules desirable? Are there any circumstances under which it would be desirable to allow noncascadeless schedules? Explain your answer.

14.7 Answer: A cascadeless schedule is one where, for each pair of transactions Ti and Tj such that Tj reads data items previously written by Ti , the commit operation of Ti appears before the read operation of Tj . Cascadeless schedules are desirable because the failure of a transaction does not lead to the aborting of any other transaction. Of course this comes at the cost of less concurrency. If failures occur rarely, so that we can pay the price of cascading aborts for the increased concurrency, noncascadeless schedules might be desirable.

d) (3 points) Suppose that each of the sequences of actions below is followed by an abort action for transaction T_1. Which of the other transactions would need to be rolled back? (Fill in the appropriate boxes to indicate your answer.)

i) $w_1(A)\ r_2(A)\ w_1(B)\ r_3(B)\ r_4(C)$

☒ T_2 rolls back ☒ T_3 rolls back ☐ T_4 rolls back

ii) $w_1(A)\ w_2(A)\ w_1(B)\ r_3(B)\ r_4(A)\ r_4(C)$

☐ T_2 rolls back ☒ T_3 rolls back ☐ T_4 rolls back

iii) $r_1(A)\ w_1(B)\ r_2(A)\ r_2(B)\ r_3(A)\ r_4(B)$

☒ T_2 rolls back ☐ T_3 rolls back ☒ T_4 rolls back

e) (3 points) For each of the following schedules of read, write, commit and abort actions done by transactions, indicate whether they are recoverable, avoids-conflicting-rollback (ACR) and/or strict by filling in the appropriate boxes. We use the notation from lecture, where $r_i(A)$, $w_i(A)$, c_i and a_i mean that transaction T_i reads A, writes A, commits or aborts respectively.

i) $w_1(A)\ r_2(A)\ w_1(B)\ r_2(B)\ c_1\ c_2$

☒ Recoverable ☐ ACR ☐ Strict

ii) $w_1(A)\ w_1(B)\ c_1\ r_2(A)\ r_2(B)\ c_2$

☒ Recoverable ☒ ACR ☒ Strict

iii) $w_1(A)\ w_2(A)\ a_2\ c_1\ r_3(A)\ r_3(B)\ c_3$

☒ Recoverable ☒ ACR ☐ Strict

15.2 Consider the following two transactions:

*T*34: read(*A*);
read(*B*);
if *A* = 0 **then** *B* := *B* + 1;
write(*B*).
*T*35: read(*B*);
read(*A*);
if *B* = 0 **then** *A* := *A* + 1;
write(*A*).

Add lock and unlock instructions to transactions *T*31 and *T*32, so that they observe the two-phase locking protocol. Can the execution of these transactions result in a deadlock?

15.2 Answer:

a. Lock and unlock instructions:

T_{34}:
lock-S(*A*)
read(*A*)
lock-X(*B*)
read(*B*)
if $A = 0$
then $B := B + 1$
write(*B*)
unlock(*A*)
unlock(*B*)

T_{35}:

lock-S(B)
read(B)
lock-X(A)
read(A)
if $B = 0$
then $A := A + 1$
write(A)
unlock(B)
unlock(A)

b. Execution of these transactions can result in deadlock. For example, consider the following partial schedule:

T_{31}	T_{32}
lock-S (A)	
	lock-S (B)
	read(B)
read(A)	
lock-X (B)	
	lock-X (A)

The transactions are now deadlocked.

15.2 Answer:

14.9 Consider a database for a bank where the database system uses snapshot isolation. Describe a particular scenario in which a nonserializable execution occurs that would present a problem for the bank.

14.9 Answer: Suppose that the bank enforces the integrity constraint that the sum of the balances in the checking and the savings account of a customer must not be negative. Suppose the checking and savings balances for a customer are $100 and $200 respectively. Suppose that transaction T1 withdraws $200 from the checking account after verifying the integrity constraint by reading both the balances. Suppose that concurrent transaction T2 withdraws $200 from the checking account after verifying the integrity constraint by reading both the balances. Since each of the transactions checks the integrity constraints on its own snapshot, if they run concurrently each will believe that the sum of the balances after the withdrawal is $100 and therefore its withdrawal does not violate the integrity constraint. Since the two transactions update different data items, they do not have any update conflict, and under snapshot isolation both of them can commit. This is a non-serializable execution which results into a serious problem of withdrawal of more money

14.10 Consider a database for an airline where the database system uses snapshot isolation. Describe a particular scenario in which a nonserializable execution occurs, but the airline may be willing to accept it in order to gain better overall performance.

14.10 Answer: Consider a web-based airline reservation system. There could be many concurrent requests to see the list of available flights and available seats in each flight and to book tickets. Suppose, there are two users A and B concurrently accessing this web application, and only one seat is left on a flight. Suppose that both user A and user B execute transactions to book a seat on the flight, and suppose that each transaction checks the total number of seats booked on the flight, and inserts a new booking record if there are enough seats left. Let T3 and T4 be their respective booking transactions, which run concurrently. Now T3 and T4 will see from their snapshots that one ticket is available and insert new booking records. Since the two transactions do not update any common data item (tuple), snapshot isolation allows both transactions to commit. This results in an extra booking, beyond the number of seats available on the flight. However, this situation is usually not very serious since cancellations often resolve the conflict; even if the conflict is present at the time the flight is to leave, the airline can arrange a different flight for one of the passengers on the flight, giving incentives to accept the change. Using snapshot isolation improves the overall performance in this case since the booking transactions read the data from their snapshots only and do not block other concurrent transactions.

Problem 4: Cost-Based Optimization (18 points)

In this problem, we will be optimizing the query X ⋈ Y ⋈ Z on relations X(A, B), Y(B, C) and Z(C, D). *(Recall that the notation X(A, B) means that relation X has attributes A and B.)* We have collected the following statistics about our relations, where T(R) is the number of tuples in a relation and V(R, A) is the number of distinct values of attribute A in a relation:

T(X) = 200	T(Y) = 1000	T(Z) = 100
V(X, A) = 100	V(Y, B) = 10	V(Z, C) = 100
V(X, B) = 20	V(Y, C) = 1000	V(Z, D) = 20

a) (8 points) As a first step to cost-based optimization, we must estimate the statistics of some potential intermediate and final tables in our query. Use the estimation rules based on T() and V() that we discussed in class to estimate the T() and V() statistics for the following tables:

i. X ⋈ Y

T(X ⋈ Y) = T(X) T(Y) / max(V(X, B), V(Y, B)) = 200 * 1000 / 20 = 10,000
V(X ⋈ Y, A) = V(X, A) = 100
V(X ⋈ Y, B) = min(V(X, B), V(Y, B)) = 10
V(X ⋈ Y, C) = V(Y, C) = 1000

ii. Y ⋈ Z

T(Y ⋈ Z) = T(Y) T(Z) / max(V(Y, C), V(Z, C)) = 1000 * 100 / 1000 = 100
V(Y ⋈ Z, B) = V(Y, B) = 10
V(Y ⋈ Z, C) = min(V(Y, C), V(Z, C)) = 100
V(Y ⋈ Z, D) = V(Z, D) = 20

iii. X ⋈ Y ⋈ Z
We can write this as (X ⋈ Y) ⋈ Z and use the results from above for X ⋈ Y:

T(X ⋈ Y ⋈ Z) = T(X ⋈ Y) T(Z) / max(V(X ⋈ Y, C), V(Z, C)) = 10,000 * 100 / 1000 = 1000
V(X ⋈ Y ⋈ Z, A) = V(X ⋈ Y, A) = 100
V(X ⋈ Y ⋈ Z, B) = V(X ⋈ Y, B) = 10
V(X ⋈ Y ⋈ Z, C) = min(V(X ⋈ Y, C), V(Z, C)) = 100
V(X ⋈ Y ⋈ Z, D) = V(Z, D) = 20

Problem 4 (10 points)

Consider a query optimizer that uses histograms. In particular, the following information is known about the attribute A of relation R where R has the attributes R(A, B, C). Attribute A is of type integer. • There are 100 tuples with A values between 1 and 10.

• There are 300 tuples with A values between 11 and 20.

• There are 200 tuples with A values between 21 and 30.

(a) Suppose that for each range, the possible A values of tuples are uniformly distributed. For example, a tuple with an A value within the range of 21 and 30 has one of the 10 possible values with equal probability. How many tuples are expected in the result of the query σA=7(R)?

Expected number of tuples: 100/10 = 10

(b) Now suppose for the same R we have the following information.

• There are 3 unique A values (uniformly distributed) within the A values between 1 and 10.

• There are 5 unique A values (uniformly distributed) within the A values between 11 and 20.

• There are 10 unique A values (uniformly distributed) within the A values between 21 and 30. How many tuples are expected in the result of the query σA=17(R) when 17 is one of the 5 unique values for the second range?

Expected number of tuples: 300/5 = 60

(c) Using the same R from (b), now consider the query R ⧏⧐A=D S, where S has attributes S(D, E, F). Assume that the following information is known about the attribute D of relation S. Attribute D is of type integer.

• There are 50 tuples with D values between 1 and 10. In this range, there are 5 unique D values uniformly distributed.

• There are 100 tuples with D values between 11 and 20. In this range, there are 5 unique D values uniformly distributed.

• There are 200 tuples with D values between 21 and 30. In this range, there are 5 unique D values uniformly distributed. Assuming that the containment of value sets assumption holds when joining R and S, how many tuples are expected in the answer?

Expected number of tuples: 100*50/5 + 300*100/5 + 200*200/10 = 11000

Problem 1:

Recovery (10 Points) Consider a database with two elements, X and Y. The initial values of X and Y are both 0. We consider three transactions U, V and W that modify these elements concurrently:

- T1: X := 42
- T2: Y := 20, X := 10
- T3: X := 100, Z := 101

While the transactions execute, the database system crashes. We consider three recovery mechanisms in turn below; your task is to first complete the missing parts of the log and then answer the following questions:

(a) Using **pure undo logging:**

1. < ST ART T2 >
2. < ST ART T3 >
3. < T2, X, 0 >
4. < T2, Y, 0 >
5. < COMMIT T2 >
6. < ST ART CKP T(T3) >
7. < T3, X, ??? >
8. < ST ART T1 >
9. < T3, Z, 0 >
10. < T1, X, ??? >

(i) Please complete the undo log by providing in the appropriate values for the log below:

Line 6: 10
Line 10: 100

(ii) If the database crashes immediately after writing the above log entries, then subsequently performs recovery, which of the following statements is true:

A. After a successful recovery, the state of Y is 0. 2
B. At the time of crash, the state of X (on disk) must be 10.
C. At the time of crash, the state of Z (on disk) must be 101.
D. After a successful recovery, the state of X is 10.

True statement (one of A, B, C, or D): D

(b) Using **pure redo logging:**

1. < ST ART T2 >
2. < ST ART T3 >
3. < T2, X, 10 >
4. < T2, Y, 20 >
5. < COMMIT T2 >
6. < ST ART CKP T(T3) >
7. < T3, X, ??? >
8. < ST ART T1 >
9. < T3, Z, ??? >
10. < T1, X, 42 >
11. < END CKP T >
12. < COMMIT T3 >
13. < ST ART CKP T(T1) >
14. < COMMIT T1 >

(i) Line 7: 100
Line 9: 101

(ii) If the database crashes immediately after writing the above log entries, then subsequently performs recovery, which of the following statements is true:

A. After a successful recovery, the state of X is 100.
B. At the time of crash, the state of X (on disk) must be 42.
C. At the time of crash, the state of Y (on disk) must be 20.
D. After a successful recovery, the state of Z is 0.

True statement (one of A, B, C, or D): C

(c) **Using undo-redo logging**, with log format (txn, item, before-val, after-val):

1. < ST ART T1 >
2. < ST ART T2 >
3. < ST ART T3 >
4. < T3, X, 0, 100 >
5. < T3, Z, 0, 101 >
6. < COMMIT T3 >
7. < T2, X, 100, 10 >
8. < ST ART CKP T(T1, T2) >
9. < T2, Y, ???, ??? >
10. < END CKP T >
11. < ST ART CKP T(T1, T2) >
12. < COMMIT T2 >
13. < T1, X, ???, ??? >

(i) Please complete the undo-redo log by providing in the appropriate values for the log below:

Line 9: 0, 20
Line 13: 10, 42

(ii) If the database crashes immediately after writing the above log entries, which of the following statements is true:

A. During recovery, the log entry in line 4 will be redone.
B. During recovery, the log entry in line 7 will be ignored.
C. During recovery, the log entry in line 9 will be ignored.
D. During recovery, the log entry in line 13 will be undone.

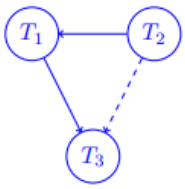
True statement (one of A, B, C, or D): B

Problem 2: Serializability and Precedence Graphs (10 Points) For each of the following schedules, draw the precedence graph and determine whether the schedule is conflict serializable. If the schedule is conflict serializable, write the conflict equivalent serial schedule of transactions (e.g., T5; T7; T2). If the schedule is not conflict serializable, identify a set of conflicting actions that prevents it from being conflict serializable (e.g., W5(C), W7(D), W2(D)). To reiterate: for each, (a) draw the precedence graph, and, (b) if conflict serializable, give the conflict equivalent serial ordering of transactions or, if not, identify the conflicting operations.

- R1(A)W2(B)R2(A)W1(B)W3(B)W1(C)R3(B)W1(A)
- W2(C)R1(A)W3(B)R1(C)R3(B)R3(A)W1(B)
- R3(B)R2(A)R1(A)R2(C)W3(A)W2(B)W1(C)
- W1(A)W2(B)W1(B)W2(A)

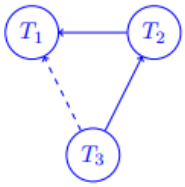
Solutions:

(a) $R_1(A)W_2(B)R_2(A)W_1(B)W_3(B)W_1(C)R_3(B)W_1(A)$



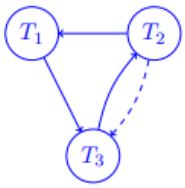
Conflict serializable, equivalent to $T_2T_1T_3$

(b) $W_2(C)R_1(A)W_3(B)R_1(C)R_3(B)R_3(A)W_2(B)W_1(B)$



Conflict serializable, equivalent to $T_3T_2T_1$

(c) $R_3(B)R_2(A)R_1(A)R_2(C)W_3(A)W_2(B)W_1(C)$



Not conflict serializable due to $R_3(B)$ before $W_2(B)$ and $R_2(A)$ before $W_3(A)$.

(d) $W_1(A)W_2(B)W_1(B)W_2(A)$

Not conflict serializable due to $W_1(A)$ before $W_2(A)$ and $W_2(B)$ before $W_1(B)$.

Problem 5: Deadlock Prevention (10 points)

Consider three transactions, T_1, T_2 and T_3 that exclusive lock (L) and unlock (U) database items A, B and C in the following order:

- $T_1 : L(B)L(C)U(C)U(B)$
- $T_2 : L(A)L(C)U(A)U(C)$
- $T_3 : L(C)L(B)U(C)U(B)$

For this problem we want to simulate how these transactions execute under either the wait-die or wound-wait deadlock prevention strategies.

For the simulation, assume that these transactions execute in a round-robin fashion (T_1 then T_2 then T_3). When it is a transaction's turn, it executes its next lock or unlock step if it can, and otherwise dies or waits or wounds the holder of the lock, as appropriate. If a transaction is waiting when it is its turn, it skips the step and continues waiting. When a waiting transaction is restarted (due to the action of some other transaction) it becomes eligible to run at its very next turn.

Assume that the transactions have timestamps in the order of $T_1 < T_2 < T_3$ (T_1 is the oldest).

Fill in the following tables to show the sequence of steps that the three transactions make. Use $L(X)$ to denote locking an object X, $U(X)$ to denote unlocking X, "Die" to denote the transaction dying or being wounded, and "Wait" to denote the transaction waiting. If a transaction A is wounded by some other transaction B, transaction B's next action is 'Die'.

In the grid, the cell for $[i, j]$ (row i column T_j) represents the i^{th} action of T_j. We have filled out a few cells to illustrate how you should fill out the rest of the table. Simulate actions until all three transactions complete (not all cells in the grid may be necessary). If a transaction is already complete, its turn is skipped.

(b) Simulation when **wound-wait** deadlock prevention is used:

Step	T_1	T_2	T_3
1	L(B)	L(A)	L(C)
2	L(C)	Wait	Die
3	U(C)	L(C)	Wait
4	U(B)	U(A)	Wait
5		U(C)	L(C)
6			L(B)
7			U(C)
8			U(B)

(a) Simulation when **wait-die** deadlock prevention is used:

Step	T_1	T_2	T_3
1	L(B)	L(A)	L(C)
2	Wait	Wait	Die
3	L(C)	Die	Die
4	U(C)	L(A)	L(C)
5	U(B)	Wait	Die
6		L(C)	Die
7		U(A)	Die
8		U(C)	Die
9			L(C)
10			L(B)
11			U(C)
12			U(B)
13			

(c) Circle each of the following statements that are true:

(i) Wound-wait always outperforms wait-die.

(ii) If we know all records that each transaction will access, we can prevent deadlocks by imposing a total order on lock acquisitions.

(iii) Wait-die always outperforms wound-wait.

(iv) In general, locking uses fewer resources than optimistic methods that rely on validation.

ii and iv are true.

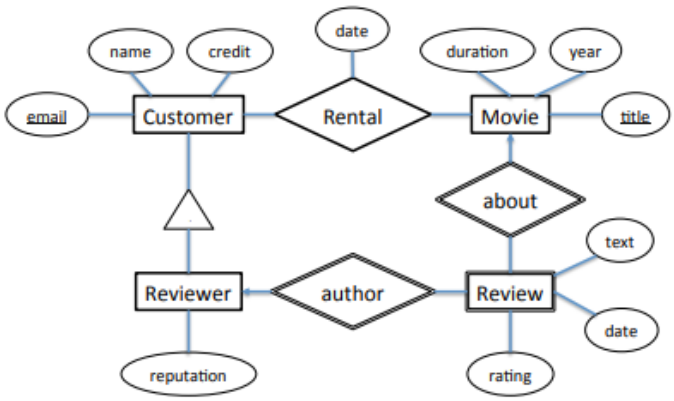
4. (14 points)

(a) (14 points) Design an E/R Diagram for an online video rental company:

- The company has data about movies, customers, rentals, reviewers, reviews.
- A Movie has a Title (key), Year, and Duration.
- A Customer has Name, Email (key), and Credit.
- Customers rent movies; customers may rent many movies, and a movie may be rented by many customers; each Rental has a Date.
- A Reviewer is a Customer, and has a Reputation attribute.
- A Review has a Rating, a Date, and Text (content). Each review is uniquely identified by the movie it is reviewing, and by the reviewer who wrote it.

Answer (draw an E/R Diagram):

Solution:



Also OK: define RENTAL to be an entity set, with two many-one relationships, to CUSTOMER and MOVIE.

2. (1) $AB \to C$ (2) $C \to A$ (3) $BC \to D$ (4) $ACD \to B$
(5) $BE \to C$ (6) $CE \to FA$ (7) $CF \to BD$ (8) $D \to EF$

Solution:

- Reduction of the left side: $ACD \to B$ can be reduced to $CD \to B$ because $\{B\} \subseteq Closure(F, CD) = (CD), (CDEF), (CDEFB)$
- Reduction of the right side: $CD \to B$ can be reduced to $CD \to \oslash$ because $\{B\} \subseteq Closure(F \backslash \{CD \to B\}, CD) = (CD), (CDEF), (CDEFB)$
- Reduction of the right side: $CE \to FA$ can be reduced to $CE \to F$ because $\{A\} \subseteq Closure(F \backslash \{CE \to FA\} \cup \{CE \to F\}, CE) = (CE), (CEF), (CEFA)$
- Reduction of the right side: $CF \to BD$ can be reduced to $CF \to B$ because $\{D\} \subseteq Closure(F \backslash \{CF \to BD\} \cup \{CF \to B\}, CF) = (CF), (CFB), (CFBD)$
- As minimal basis remains:

  $AB \to C$, $C \to A$, $BC \to D$, $BE \to C$,
  $CE \to F$, $CF \to B$, $D \to EF$

- An alternative solution would be to right-reduce $CF \to BD$ instead of $CD \to B$

7. For each of the sets of dependencies in question 4:

i. if R is not already in 3NF, decompose it into a set of 3NF relations

ii. if R is not already in BCNF, decompose it into a set of BCNF relations

a. $C \rightarrow D, \quad C \rightarrow A, \quad B \rightarrow C$

[hide answer]

i. application of the 3NF algorithm decomposes R into three relations based on the dependencies ($R_1(CD)$, $R_2(CA)$, $R_3(BC)$); this decomposition leaves enough "connectivity" between the relations that no extra "candidate key" relation is needed to make them join-preserving (from now on, we denote a relation like $R_1(CD)$ simply by CD)

ii. the above 3NF decomposition is also in BCNF

b. $B \rightarrow C, \quad D \rightarrow A$

[hide answer]

i. application of the 3NF algorithm decomposes R into BC, AD (using the dependencies), but also requires the addition of a "linking" relation BD containing the candidate key

ii. this is the same decomposition that would be reached by following the BCNF algorithm, although it would proceed by first producing AD, BCD and then decomposing BCD further into BC, BD

c. $ABC \rightarrow D, \quad D \rightarrow A$

[hide answer]

i. R is already in 3NF

ii. applying the BCNF algorithm means decomposition to "fix" the violation caused by $D \rightarrow A$ giving AD, BCD (note that this does not preserve the dependency $ABC \rightarrow D$, and, in fact, it is not possible to find a BCNF decomposition which does preserve it)

d. $A \rightarrow B, \quad BC \rightarrow D, \quad A \rightarrow C$

[hide answer]

i. the standard algorithm produces AB, BCD, AC, which contains enough connectivity to not require a "linking" relation

ii. $BC \rightarrow D$ violates BCNF because BC does not contain a key, so we split R into BCD, ABC, and this is now BCNF

e. $AB \rightarrow C, \quad AB \rightarrow D, \quad C \rightarrow A, \quad D \rightarrow B$

[hide answer]

i. R is already in 3NF

ii. applying the standard BCNF decomposition algorithm requires us to "fix" the BCNF violations caused by $C \rightarrow A$ and $D \rightarrow B$; this could be achieved by decomposing R into AC and BCD, but this does not preserve the other dependencies; to achieve dependency preservation requires the following decomposition: AC, BD, CD, ABC, ABD (but the methods for doing this were not covered in lectures)

f. $A \rightarrow BCD$

[hide answer]

i. R is already in 3NF

ii. R is already in BCNF