1. The following schedule results in a write-read conflict:
   T2:R(X), T2:R(Y), T2:W(X), T1:R(X) ...
   T1:R(X) is a dirty read here.

2. The following schedule results in a read-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X) ...
   Now, T2 will get an unrepeatable read on X.

3. The following schedule results in a write-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:W(X) ...
   Now, T2 has overwritten uncommitted data.

4. Strict 2PL resolves these conflicts as follows:

   (a) In S2PL, T1 could not get a shared lock on X because T2 would be holding an exclusive lock on X. Thus, T1 would have to wait until T2 was finished.

   (b) Here T1 could not get an exclusive lock on X because T2 would already be holding a shared or exclusive lock on X.

   (c) Same as above.

Exercise 16.4 We call a transaction that only reads database object a **read-only** transaction, otherwise the transaction is called a **read-write** transaction. Give brief answers to the following questions:

1. What is lock thrashing and when does it occur?

2. What happens to the database system throughput if the number of read-write transactions is increased?

3. What happens to the database system throughput if the number of read-only transactions is increased?

4. Describe three ways of tuning your system to increase transaction throughput.

Answer 16.4 The answer to each question is given below.

1. *Locking thrashing* occurs when the database system reaches to a point where adding another new active transaction actually reduces throughput due to competition for locking among all active transactions. Empirically, locking thrashing is seen to occur when 30% of active transactions are blocked.

2. If the number of read-write transaction is increased, the database system throughput will increase until it reaches the thrashing point; then it will decrease since read-write transactions require exclusive locks, thus resulting in less concurrent execution.

3. If the number of read-only transaction is increased, the database system throughput will also increase since read-only transactions require only shared locks. So we are able to have more concurrency and execute more transactions in a given time.

4. Throughput can be increased in three ways:

   (a) By locking the smallest sized objects possible.

   (b) By reducing the time that transaction hold locks.

   (c) By reducing hot spots, a database object that is frequently accessed and modified.

Exercise 16.5 Suppose that a DBMS recognizes *increment*, which increments an integer-valued object by 1, and *decrement* as actions, in addition to reads and writes. A transaction that increments an object need not know the value of the object; increment and decrement are versions of blind writes. In addition to shared and exclusive locks, two special locks are supported: An object must be locked in I mode before incrementing it and locked in D mode before decrementing it. An I lock is compatible with another I or D lock on the same object, but not with S and X locks.

1. Illustrate how the use of I and D locks can increase concurrency. (Show a schedule allowed by Strict 2PL that only uses S and X locks. Explain how the use of I and D locks can allow more actions to be interleaved, while continuing to follow Strict 2PL.)

2. Informally explain how Strict 2PL guarantees serializability even in the presence of I and D locks. (Identify which pairs of actions conflict, in the sense that their relative order can affect the result, and show that the use of S, X, I, and D locks according to Strict 2PL orders all conflicting pairs of actions to be the same as the order in some serial schedule.)

Answer 16.5 The answer to each question is given below.

1. Take the following two transactions as example:

   T1: Increment A, Decrement B, Read C;
   T2: Increment B, Decrement A, Read C

   If using only strict 2PL, all actions are versions of blind writes, they have to obtain exclusive locks on objects. Following strict 2PL, T1 gets an exclusive lock on A, if T2 now gets an exclusive lock on B, there will be a deadlock. Even if T1 is fast enough to have grabbed an exclusive lock on B first, T2 will now be blocked until T1 finishes. This has little concurrency. If I and D locks are used, since I and

2. Change enrollment for a student identified by her *snum* from one class to another class.
3. Assign a new faculty member identified by his *fid* to the class with the least number of students.
4. For each class, show the number of students enrolled in the class.

Answer 16.7 The answer to each question is given below.

1. Because we are inserting a new row in the table *Enrolled*, we do not need any lock on the existing rows. So we would use READ UNCOMMITTED.
2. Because we are updating one existing row in the table *Enrolled*, we need an exclusive lock on the row which we are updating. So we would use READ COMMITTED.
3. To prevent other transactions from inserting or updating the table *Enrolled* while we are reading from it (known as the phantom problem), we would need to use SERIALIZABLE.
4. same as above.

Exercise 16.8 Consider the following schema:

Suppliers(*sid:* `integer`, *sname:* `string`, *address:* `string`)
Parts(*pid:* `integer`, *pname:* `string`, *color:* `string`)
Catalog(*sid:* `integer`, *pid:* `integer`, *cost:* `real`)

The Catalog relation lists the prices charged for parts by Suppliers.

For each of the following transactions, state the SQL isolation level that you would use and explain why you chose it.

1. A transaction that adds a new part to a supplier's catalog.
2. A transaction that increases the price that a supplier charges for a part.
3. A transaction that determines the total number of items for a given supplier.
4. A transaction that shows, for each part, the supplier that supplies the part at the lowest price.

Answer 16.8 The answer to each question is given below.

1. Because we are inserting a new row in the table *Catalog*, we do not need any lock on the existing rows. So we would use READ UNCOMMITTED.

2. Because we are updating one existing row in the table *Catalog*, we need an exclusive lock on the row which we are updating. So we would use READ COMMITTED.

3. To prevent other transactions from inserting or updating the table *Catalog* while we are reading from it (known as the phantom problem), we would need to use SERIALIZABLE.

4. same as above.

Exercise 16.9 Consider a database with the following schema:

Suppliers(*sid:* `integer`, *sname:* `string`, *address:* `string`)
Parts(*pid:* `integer`, *pname:* `string`, *color:* `string`)
Catalog(*sid:* `integer`, *pid:* `integer`, *cost:* `real`)

The Catalog relation lists the prices charged for parts by Suppliers.

Consider the transactions $T1$ and $T2$. $T1$ always has SQL isolation level `SERIALIZABLE`. We first run $T1$ concurrently with $T2$ and then we run $T1$ concurrently with $T2$ but we change the isolation level of $T2$ as specified below. Give a database instance and SQL statements for $T1$ and $T2$ such that result of running $T2$ with the first SQL isolation level is different from running $T2$ with the second SQL isolation level. Also specify the common schedule of $T1$ and $T2$ and explain why the results are different.

1. `SERIALIZABLE` versus `REPEATABLE READ`.

2. `REPEATABLE READ` versus `READ COMMITTED`.

3. `READ COMMITTED` versus `READ UNCOMMITTED`.

Answer 16.9 The answer to each question is given below.

1. Suppose a database instance of table Catalog and SQL statements shown below:

sid	pid	cost
18	45	$7.05
22	98	$89.35
31	52	$357.65
31	53	$26.22
58	15	$37.50
58	94	$26.22

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. 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. The only way R could be in BCNF is if B includes a key, *i.e.* B is a key for R.
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 19.2 Consider a relation R with five attributes $ABCDE$. You are given the following dependencies: $A \rightarrow B$, $BC \rightarrow E$, and $ED \rightarrow A$.

1. List all keys for R.
2. Is R in 3NF?
3. Is R in BCNF?

Answer 19.2

1. CDE, ACD, BCD
2. R is in 3NF because B, E and A are all parts of keys.
3. R is not in BCNF because none of A, BC and ED contain a key.

Exercise 19.3 Consider the relation shown in Figure 19.1.

1. List all the functional dependencies that this relation instance satisfies.

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_2	y_1	z_1
x_2	y_1	z_3

Figure 19.1 Relation for Exercise 19.3.

2. Assume that the value of attribute Z of the last record in the relation is changed from z_3 to z_2. Now list all the functional dependencies that this relation instance satisfies.

Answer 19.3

1. The following functional dependencies hold over R: $Z \rightarrow Y$, $X \rightarrow Y$, and $XZ \rightarrow Y$
2. Same as part 1. Functional dependency set is unchanged.

Exercise 19.4 Assume that you are given a relation with attributes $ABCD$.

1. Assume that no record has NULL values. Write an SQL query that checks whether the functional dependency $A \rightarrow B$ holds.
2. Assume again that no record has NULL values. Write an SQL assertion that enforces the functional dependency $A \rightarrow B$.
3. Let us now assume that records could have NULL values. Repeat the previous two questions under this assumption.

Answer 19.4 Assuming...

1. The following statement returns 0 iff no statement violates the FD $A \rightarrow B$.

```
SELECT COUNT (*)
FROM   R AS R1, R AS R2
WHERE  (R1.B != R2.B) AND (R1.A = R2.A)
```

2.
```
CREATE ASSERTION ADeterminesB
CHECK  ((SELECT COUNT (*)
        FROM   R AS R1, R AS R2
        WHERE  (R1.B != R2.B) AND (R1.A = R2.A))
        =0)
```

3. Note that the following queries can be written with the NULL and NOT NULL interchanged. Since we are doing a full join of a table and itself, we are creating tuples in sets of two therefore the order is not important.

```
SELECT COUNT (*)
FROM   R AS R1, R AS R2
WHERE  ((R1.B != R2.B) AND (R1.A = R2.A))
       OR ((R1.B is NULL) AND (R2.B is NOT NULL)
              AND (R1.A = R2.A))

CREATE ASSERTION ADeterminesBNull
CHECK  ((SELECT COUNT (*)
       FROM   R AS R1, R AS R2
       WHERE  ((R1.B != R2.B) AND (R1.A = R2.A)))
              OR ((R1.B is NULL) AND (R2.B is NOT NULL)
                     AND (R1.A = R2.A))
       =0)
```

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 19.6 Suppose that we have the following three tuples in a legal instance of a relation schema S with three attributes ABC (listed in order): (1,2,3), (4,2,3), and (5,3,3).

1. Which of the following dependencies can you infer does *not* hold over schema S?

   (a) $A \rightarrow B$, (b) $BC \rightarrow A$, (c) $B \rightarrow C$

2. Can you identify any dependencies that hold over S?

Answer 19.6

1. $BC\rightarrow A$ does not hold over S (look at the tuples (1,2,3) and (4,2,3)). The other tuples hold over S.

2. No. Given just an instance of S, we can say that certain dependencies (e.g., $A \rightarrow B$ and B $\rightarrow C$) are not violated by this instance, but we cannot say that these dependencies hold with respect to S. To say that an FD holds w.r.t. a relation is to make a statement about *all* allowable instances of that relation!

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.

1. $C \rightarrow D$, $C \rightarrow A$, $B \rightarrow C$
2. $B \rightarrow C$, $D \rightarrow A$
3. $ABC \rightarrow D$, $D \rightarrow A$
4. $A \rightarrow B$, $BC \rightarrow D$, $A \rightarrow C$
5. $AB \rightarrow C$, $AB \rightarrow D$, $C \rightarrow A$, $D \rightarrow B$

Answer 19.7

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. (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. (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) 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. (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.

Exercise 19.8 Consider the attribute set $R = ABCDEGH$ and the FD set $F = \{AB \rightarrow C,\ AC \rightarrow B,\ AD \rightarrow E,\ B \rightarrow D,\ BC \rightarrow A,\ E \rightarrow G\}$.

1. For each of the following attribute sets, do the following: (i) Compute the set of dependencies that hold over the set and write down a minimal cover. (ii) Name the strongest normal form that is not violated by the relation containing these attributes. (iii) Decompose it into a collection of BCNF relations if it is not in BCNF.

   (a) ABC, (b) $ABCD$, (c) $ABCEG$, (d) $DCEGH$, (e) $ACEH$

2. Which of the following decompositions of $R = ABCDEG$, with the same set of dependencies F, is (a) dependency-preserving? (b) lossless-join?

   (a) $\{AB,\ BC,\ ABDE,\ EG\ \}$

   (b) $\{ABC,\ ACDE,\ ADG\ \}$

Answer 19.8

1. (a) i. $R1 = ABC$: The FD's are $AB \rightarrow C$, $AC \rightarrow B$, $BC \rightarrow A$.

ii. This is already a minimal cover.

iii. This is in BCNF since AB, AC and BC are candidate keys for *R1*. (In fact, these are all the candidate keys for *R1*).

(b) i. *R2* = $ABCD$: The FD's are $AB \rightarrow C$, $AC \rightarrow B$, $B \rightarrow D$, $BC \rightarrow A$.

ii. This is a minimal cover already.

iii. The keys are: AB, AC, BC. *R2* is not in BCNF or even 2NF because of the FD, $B \rightarrow D$ (B is a proper subset of a key!) However, it is in 1NF. Decompose as in: ABC, BD. This is a BCNF decomposition.

(c) i. *R3* = $ABCEG$; The FDs are $AB \rightarrow C$, $AC \rightarrow B$, $BC \rightarrow A$, $E \rightarrow G$.

ii. This is in minimal cover already.

iii. The keys are: ABE, ACE, BCE. It is not even in 2NF since E is a proper subset of the keys and there is a FD $E \rightarrow G$. It is in 1NF . Decompose as in: ABE, ABC, EG. This is a BCNF decompostion.

(d) i. *R4* = $DCEGH$; The FD is $E \rightarrow G$.

ii. This is in minimal cover already.

iii. The key is $DCEH$; It is not in BCNF since in the FD $E \rightarrow G$, E is a subset of the key and is not in 2NF either. It is in 1 NF Decompose as in: $DCEH$, EG

(e) i. *R5* = $ACEH$; No FDs exist.

ii. This is a minimal cover.

iii. Key is $ACEH$ itself.

iv. It is in BCNF form.

2. (a) The decomposition. { AB, BC, ABDE, EG } is *not* lossless. To prove this consider the following instance of R:

$$\{(a_1, b, c_1, d_1, e_1, g_1),\ (a_2, b, c_2, d_2, e_2, g_2)\}$$

Because of the functional dependencies $BC \rightarrow A$ and $AB \rightarrow C$, $a_1 \neq a_2$ if and only if $c_1 \neq c_2$. It is easy to that the join AB $\bowtie$ BC contains 4 tuples:

$$\{(a_1, b, c_1),\ (a_1, b, c_2),\ (a_2, b, c_1),\ (a_2, b, c_2)\}$$

So the join of AB, BC, $ABDE$ and EG will contain at least 4 tuples, (actually it contains 8 tuples) so we have a lossy decomposition here.

This decomposition does not preserve the FD, $AB \rightarrow C$ (or $AC \rightarrow B$)

(b) The decomposition {ABC, ACDE, ADG } is lossless. Intuitively, this is because the join of ABC, $ACDE$ and ADG can be constructed in two steps; first construct the join of ABC and ACDE: this is lossless because their (attribute) intersection is AC which is a key for $ABCDE$ (in fact $ABCDEG$) so this is lossless. Now join this intermediate join with ADG. This is also lossless because the attribute intersection is is AD and $AD \rightarrow ADG$. So by the test mentioned in the text this step is also a lossless decomposition.

The projection of the FD's of R onto ABC gives us: $AB \rightarrow C$, $AC \rightarrow B$ and $BC \rightarrow A$. The projection of the FD's of R onto $ACDE$ gives us: $AD \rightarrow E$ and The projection of the FD's of R onto ADG gives us: $AD \rightarrow G$ (by transitivity) The closure of this set of dependencies does not contain $E \rightarrow G$ nor does it contain $B \rightarrow D$. So this decomposition is not dependency preserving.

Exercise 19.9 Let R be decomposed into R_1, R_2, ..., R_n. Let F be a set of FDs on R.

1. Define what it means for F to *be preserved* in the set of decomposed relations.
2. Describe a polynomial-time algorithm to test dependency-preservation.
3. Projecting the FDs stated over a set of attributes X onto a subset of attributes Y requires that we consider the closure of the FDs. Give an example where considering the closure is important in testing dependency-preservation, that is, considering just the given FDs gives incorrect results.

Answer 19.9

1. Let F_i denote the projection of F on R_i. F is *preserved* if the closure of the (union of) the F_i's equals F (note that F is always a superset of this closure.)
2. We shall describe an algorithm for testing dependency preservation which is polynomial in the cardinality of F. For each dependency X $\rightarrow$ Y $\in$ F check if it is in F as follows: start with the set S (of attributes in) X. For each relation R_i, compute the closure of $S \cap R_i$ relative to F and project this closure to the attributes of R_i. If this results in additional attributes, add them to S. Do this repeatedly until there is no change to S.
3. There is an example in the text in Section 19.5.2.

Exercise 19.10 Suppose you are given a relation *R(A,B,C,D)*. For each of the following sets of FDs, assuming they are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) State whether or not the proposed decomposition of R into smaller relations is a good decomposition and briefly explain why or why not.

1. $B \rightarrow C$, $D \rightarrow A$; decompose into BC and AD.
2. $AB \rightarrow C$, $C \rightarrow A$, $C \rightarrow D$; decompose into ACD and BC.
3. $A \rightarrow BC$, $C \rightarrow AD$; decompose into ABC and AD.
4. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decompose into AB and ACD.

5. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decompose into AB, AD and CD.

Answer 19.10

1. Candidate key(s): BD. The decomposition into BC and AD is unsatisfactory because it is lossy (the join of BC and AD is the cartesian product which could be much bigger than $ABCD$)

2. Candidate key(s): AB, BC. The decomposition into ACD and BC is lossless since $ACD \cap BC$ (which is C) $\rightarrow ACD$. The projection of the FD's on ACD include C $\rightarrow$ D, C $\rightarrow$ A (so C is a key for ACD) and the projection of FD on BC produces no nontrivial dependencies. In particular this is a BCNF decomposition (check that R is not!). However, it is not dependency preserving since the dependency AB $\rightarrow$ C is not preserved. So to enforce preservation of this dependency (if we do not want to use a join) we need to add ABC which introduces redundancy. So implicitly there is some redundancy across relations (although none inside ACD and BC).

3. Candidate key(s): A, C. Since A and C are both candidate keys for R, it is already in BCNF. So from a normalization standpoint it makes no sense to decompose R. Further more, the decompose is not dependency-preserving since $C \rightarrow AD$ can no longer be enforced.

4. Candidate key(s): A. The projection of the dependencies on AB are: $A \rightarrow B$ and those on ACD are: $A \rightarrow C$ and $C \rightarrow D$ (rest follow from these). The scheme ACD is not even in 3NF, since C is not a superkey, and D is not part of a key. This is a lossless-join decomposition (since A is a key), but not dependency preserving, since $B \rightarrow C$ is not preserved.

5. Candidate key(s): A (just as before) This is a lossless BCNF decomposition (easy to check!) This is, however, not dependency preserving (B *consider* $\rightarrow C$). So it is not free of (implied) redundancy. This is not the best decomposition (the decomposition AB, BC, CD is better.)

Exercise 19.11 Consider a relation R that has three attributes ABC. It is decomposed into relations R_1 with attributes AB and R_2 with attributes BC.

1. State the definition of a lossless-join decomposition with respect to this example. Answer this question concisely by writing a relational algebra equation involving R, R_1, and R_2.

2. Suppose that $B \rightarrow C$. Is the decomposition of R into R_1 and R_2 lossless-join? Reconcile your answer with the observation that neither of the FDs $R_1 \cap R_2 \rightarrow R_1$ nor $R_1 \cap R_2 \rightarrow R_2$ hold, in light of the simple test offering a necessary and sufficient condition for lossless-join decomposition into two relations in Section 15.6.1.