17 CONCURRENCY CONTROL

Exercise 17.1 Answer the following questions:

1. Describe how a typical lock manager is implemented. Why must lock and unlock be atomic operations? What is the difference between a lock and a *latch*? What are *convoys* and how should a lock manager handle them?
2. Compare *lock downgrades* with upgrades. Explain why downgrades violate 2PL but are nonetheless acceptable. Discuss the use of *update* locks in conjunction with lock downgrades.
3. Contrast the timestamps assigned to restarted transactions when timestamps are used for deadlock prevention versus when timestamps are used for concurrency control.
4. State and justify the Thomas Write Rule.
5. Show that, if two schedules are conflict equivalent, then they are view equivalent.
6. Give an example of a serializable schedule that is not strict.
7. Give an example of a strict schedule that is not serializable.
8. Motivate and describe the use of locks for improved conflict resolution in Optimistic Concurrency Control.

Answer 17.1 The answer to each question is given below.

1. A typical lock manager is implemented with a hash table, also called lock table, with the data object identifier as the key. A lock table entry contains the following information: the number of transactions currently holding a lock on the object, the nature of the lock, and a pointer to a queue of lock requests.

Lock and unlock must be atomic operations because otherwise it may be possible for two transactions to obtain an exclusive lock on the same object, thereby destroying the principles of 2PL.

A lock is held over a long duration, and a latch is released immediately after the physical read or write operation is completed.

Convoy is a queue of waiting transactions. It occurs when a transaction holding a heavily used lock is suspended by the operating system, and every other transactions that needs this lock is queued.

2. A *lock upgrade* is to grant a transaction an exclusive lock of an object for which it already holds a shared lock. A *lock downgrade* happens when an exclusive lock is obtained by a transaction initially, but downgrades to a shared lock once it's clear that this is sufficient.

   Lock downgrade violates the 2PL requirement because it reduces the locking privileges held by a transaction, and the transaction may go on to acquire other locks. But the transaction did nothing but read the object that it downgraded. So it is nonetheless acceptable, provided that the transaction has not modified the object.

   The downgrade approach reduces concurrency, so we introduce a new kind of lock, called an update lock, that is compatible with shared locks but not other updates and exclusive lock. By setting an update lock initially, rather than a exclusive lock as in the case of lock downgrade, we prevent conflicts with other read operations and increase concurrency.

3. When timestamps are used for deadlock prevention, a transaction that is aborted and re-started it is given the same timestamp that it had originally. When timestamps are used for concurrency control, a transaction that is aborted and restarted is given a new, larger timestamp.

4. The Thomas Write Rule says that if an transaction T with timestamp TS(T) acts on a database object O with a write timestamp of WTS(O) such that TS(T) < WTS(O), we can safely ignore writes by T and continue.

   To understand and justify the Thomas Write Rule fully, we need to give the complete context when it arises.

   To implement timestamp-based concurrency control scheme, the following regulations are made when transaction T wants to write object O:

   (a) If $TS(T) < RTS(O)$, the write action conflicts with the most recent read action of O, and T is therefore aborted and restarted.
   (b) If $TS(T) < WTS(O)$, a naive approach would be to abort T as well because its write action conflicts with the most recent write of O, and is out of timestamp order. But it turns out that we can safely ignore such previous write and process with this new write; this is called $Thomas'WriteRule$.

   (c) Otherwise, T writes O and WTS(O) is set to TS(T).

   The justification is as follows: had $TS(T) < RTS(O)$, T would have been aborted and we would not have bothered to check the WTS(O). So to decide whether to abort T based on WTS(O), we can assume that $TS(T) >= RTS(O)$. If $TS(T) >= RTS(O)$ and $TS(T) < WTS(O)$, then $RTS(O) < WTS(O)$, which means the previous write occurred immediately before this planned-new-write of O and was never read by anyone, therefore the previous write can be safely ignored.

5. If two schedules over the same set of actions of the transactions are conflict equivalent, they must order every pair of conflicting actions of two committed transactions in the same way. Let's assume that two schedules are conflict equivalent, but they are not view equivalent, then one of the three conditions held under view equivalency must be violated. But as we can see if every pair of conflicting actions is ordered in the same way, this cannot happen. Thus we can conclude that if two schedules are conflict equivalent, they are also view equivalent.

6. The following example is a serializable schedule, but it's not strict.
   T1:R(X), T2:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit

7. The following example is a strict schedule, but it's not serializable.
   T1:R(X), T2:R(X), T1:W(X), T1:Commit, T2:W(X), T2:Commit

8. In Optimistic Concurrency Control, we have no way to tell when Ti wrote the object at the time we validate Tj, since all we have is the list of objects written by Ti and the list read by Tj. To solve such conflict, we use mechanisms very similar to locking. The basic idea is that each transaction in the Read phase tells the DBMS about items it is reading, and when a transaction Ti is committed and its writes are accepted, the DBMS checks whether any of the items written by Ti are being rad by any (yet to be validated) transaction Tj. If so, we know that Tj's validation must eventually fail. Then we can pick either the die or kill policy to resolve the conflict.

Exercise 17.2 Consider the following classes of schedules: *serializable, conflict-serializable, view-serializable, recoverable, avoids-cascading-aborts,* and *strict.* For each of the following schedules, state which of the preceding classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly.

The actions are listed in the order they are scheduled and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort or commit must follow all the listed actions.

1. T1:R(X), T2:R(X), T1:W(X), T2:W(X)

2. T1:W(X), T2:R(Y), T1:R(Y), T2:R(X)
3. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)
4. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)
5. T1:R(X), T2:W(X), T1:W(X), T2:Abort, T1:Commit
6. T1:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit
7. T1:W(X), T2:R(X), T1:W(X), T2:Abort, T1:Commit
8. T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Commit
9. T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Abort
10. T2: R(X), T3:W(X), T3:Commit, T1:W(Y), T1:Commit, T2:R(Y), T2:W(Z), T2:Commit
11. T1:R(X), T2:W(X), T2:Commit, T1:W(X), T1:Commit, T3:R(X), T3:Commit
12. T1:R(X), T2:W(X), T1:W(X), T3:R(X), T1:Commit, T2:Commit, T3:Commit

Answer 17.2 For simplicity, we assume the listed transactions are the only ones active currently in the database and if a commit or abort is not shown for a transaction, we'll assume a commit will follow all the listed actions.

1. Not serializable, not conflict-serializable, not view-serializable;
   It is recoverable and avoid cascading aborts; not strict.
2. It is serializable, conflict-serializable, and view-serializable;
   It does NOT avoid cascading aborts, is not strict;
   We can not decide whether it's recoverable or not, since the abort/commit sequence of these two transactions are not specified.
3. It is the same with number 2 above.
4. It is NOT serializable, NOT conflict-serializable, NOT view-serializable;
   It is NOT avoid cascading aborts, not strict;
   We can not decide whether it's recoverable or not, since the abort/commit sequence of these transactions are not specified.
5. It is serializable, conflict-serializable, and view-serializable;
   It is recoverable and avoid cascading aborts;
   It is not strict.
6. It is serializable and view-serializable, not conflict-serializable;
   It is recoverable and avoid cascading aborts;
   It is not strict.

7. It is not serializable, not view-serializable, not conflict-serializable;
   It is not recoverable, therefore not avoid cascading aborts, not strict.

8. It is not serializable, not view-serializable, not conflict-serializable;
   It is not recoverable, therefore not avoid cascading aborts, not strict.

9. It is serializable, view-serializable, and conflict-serializable;
   It is not recoverable, therefore not avoid cascading aborts, not strict.

10. It belongs to all above classes.

11. (assume the 2nd T2:Commit is instead T1:Commit).
    It is serializable and view-serializable, not conflict-serializable;
    It is recoverable, avoid cascading aborts and strict.

12. It is serializable and view-serializable, not conflict-serializable;
    It is recoverable, but not avoid cascading aborts, not strict.

Exercise 17.3 Consider the following concurrency control protocols: 2PL, Strict 2PL, Conservative 2PL, Optimistic, Timestamp without the Thomas Write Rule, Timestamp with the Thomas Write Rule, and Multiversion. For each of the schedules in Exercise 17.2, state which of these protocols allows it, that is, allows the actions to occur in exactly the order shown.

For the timestamp-based protocols, assume that the timestamp for transaction Ti is i and that a version of the protocol that ensures recoverability is used. Further, if the Thomas Write Rule is used, show the equivalent serial schedule.

Answer 17.3 See the table 17.1.

Note the following abbreviations.
S-2PL: Strict 2PL; C-2PL: Conservative 2PL; Opt cc: Optimistic; TS W/O THR: Timestamp without Thomas Write Rule; TS With THR: Timestamp without Thomas Write Rule.

Thomas Write Rule is used in the following schedules, and the equivalent serial schedules are shown below:
5. T1:R(X), T1:W(X), T2:Abort, T1:Commit

6. T1:R(X), T1:W(X), T2:Commit, T1:Commit
11. T1:R(X), T2:Commit, T1:W(X), T2:Commit, T3:R(X), T3:Commit

Exercise 17.4 Consider the following sequences of actions, listed in the order they are submitted to the DBMS:

	2PL	S-2PL	C-2PL	Opt CC	TS w/o TWR	TS w/ TWR	Multiv.
1	N	N	N	N	N	N	N
2	Y	N	N	Y	Y	Y	Y
3	N	N	N	Y	N	N	Y
4	N	N	N	Y	N	N	Y
5	N	N	N	Y	N	Y	Y
6	N	N	N	N	N	Y	Y
7	N	N	N	Y	N	N	N
8	N	N	N	N	N	N	N
9	N	N	N	Y	N	N	N
10	N	N	N	N	Y	Y	Y
11	N	N	N	N	N	Y	N
12	N	N	N	N	N	Y	Y

Table 17.1

- **Sequence S1:** T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit
- **Sequence S2:** T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each sequence and for each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.

Assume that the timestamp of transaction Ti is i. For lock-based concurrency control mechanisms, add lock and unlock requests to the previous sequence of actions as per the locking protocol. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

1. Strict 2PL with timestamps used for deadlock prevention.
2. Strict 2PL with deadlock detection. (Show the waits-for graph in case of deadlock.)
3. Conservative (and Strict, i.e., with locks held until end-of-transaction) 2PL.
4. Optimistic concurrency control.
5. Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.
6. Multiversion concurrency control.

Answer 17.4 The answer to each question is given below.

1. Assume we use Wait-Die policy.
   Sequence S1: T1 acquires shared-lock on X;
   When T2 asks for an exclusive lock on X, since T2 has a lower priority, it will be aborted;
   T3 now gets exclusive-lock on Y;
   When T1 also asks for an exclusive-lock on Y which is still held by T3, since T1 has higher priority, T1 will be blocked waiting;
   T3 now finishes write, commits and releases all the lock;
   T1 wakes up, acquires the lock, proceeds and finishes;
   T2 now can be restarted successfully.

   Sequence S2: The sequence and consequence are the same with Sequence S1, except T2 was able to advance a little more before it gets aborted.

2. In deadlock detection, transactions are allowed to wait, they are not aborted until a deadlock has been detected. (Compared to prevention schema, some transactions may have been aborted prematurely.)

   Sequence S1: T1 gets a shared-lock on X;
   T2 blocks waiting for an exclusive-lock on X;
   T3 gets an exclusive-lock on Y;
   T1 blocks waiting for an exclusive-lock on Y;
   T3 finishes, commits and releases locks;
   T1 wakes up, gets an exclusive-lock on Y, finishes up and releases lock on X and Y;
   T2 now gets both an exclusive-lock on X and Y, and proceeds to finish.
   No deadlock.

   Sequence S2: There is a deadlock. T1 waits for T2, while T2 waits for T1.

3. **Sequence S1:** With conservative and strict 2PL, the sequence is easy. T1 acquires lock on both X and Y, commits, releases locks; then T2; then T3.

   Sequence S2: Same as Sequence S1.

4. Optimistic concurrency control:
   For both S1 and S2: each transaction will execute, read values from the database and write to a private workspace; they then acquire a timestamp to enter the validation phase. The timestamp of transaction Ti is i.

   Sequence S1: Since T1 gets the earliest timestamp, it will commit without problem; but when validating T2 against T1, none of the three conditions hold, so T2 will be aborted and restarted later; so is T3 (same as T2).

   Sequence S2: The fate is the same as in Sequence S1.

	Serializable	Conflict-serializable	Recoverable	Avoid cascading aborts
1	No	No	No	No
2	No	No	Yes	Yes
3	Yes	Yes	Yes	Yes
4	Yes	Yes	Yes	Yes

Table 17.2

5. Timestamp concurrency control with buffering of reads and writes and TWR.

   Sequence S1: This sequence will be allowed the way it is.

   Sequence S2: Same as above.

6. Multiversion concurrency control

   Sequence S1: T1 reads X, so RTS(X) = 1;
   T2 is able to write X, since TS(T2) ¿ RTS(X); and RTS(X) and WTS(X) are set to 2;
   T2 writes Y, RTS(Y) and WTS(Y) are set to 2;
   T3 is able to write Y as well, so RTS(Y) and WTS(Y) are set to 3;
   Now when T1 tries to write Y, since TS(T1) ¡ RTS(Y), T1 needs to be aborted and restarted later.

   Sequence S2: The fate is similar to the one in Sequence S1.

Exercise 17.5 For each of the following locking protocols, assuming that every transaction follows that locking protocol, state which of these desirable properties are ensured: serializability, conflict-serializability, recoverability, avoidance of cascading aborts.

1. Always obtain an exclusive lock before writing; hold exclusive locks until end-of-transaction. No shared locks are ever obtained.
2. In addition to (1), obtain a shared lock before reading; shared locks can be released at any time.
3. As in (2), and in addition, locking is two-phase.
4. As in (2), and in addition, all locks held until end-of-transaction.

Answer 17.5 See the table 17.2.

Exercise 17.6 The Venn diagram (from [76]) in Figure 17.1 shows the inclusions between several classes of schedules. Give one example schedule for each of the regions $S1$ through $S12$ in the diagram.

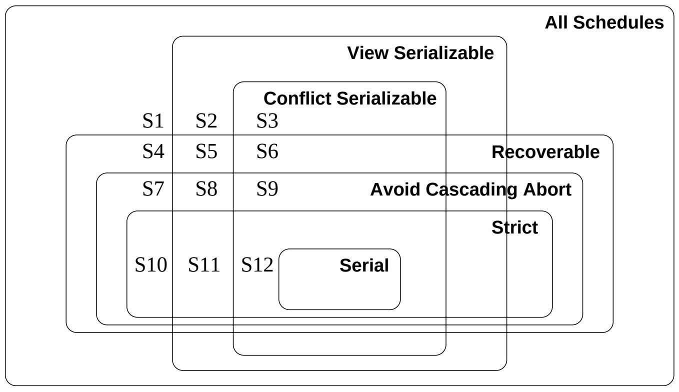

Figure 17.1 Venn Diagram for Classes of Schedules

Answer 17.6 Each section is described below.

- S1
  T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Commit
- S2
  T1:R(X), T2:W(X), T1:W(X), T3:R(X), T3:Commit, T1:Commit, T2:Commit
- S3
  T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Abort
- S4
  T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y), T3:Commit, T2:Commit, T1:Commit
- S5
  T1:R(X), T2:W(X), T1:W(X), T3:R(X), T1:Commit, T2:Commit, T3:Commit
- S6
  T1:W(X), T2:R(Y), T1:R(Y), T2:R(X), T1:Commit, T2:Commit
- S7
  T1:R(X), T2:R(X), T1:W(X), T2:W(X), T1:Commit, T2:Commit
- S8
  T1:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit
- S9
  T1:R(X), T2:W(X), T1:W(X), T2:Abort, T1:Commit
- S10
  T1:R(X), T2:R(X), T1:W(X), T1:Commit, T2:W(X), T2:Commit

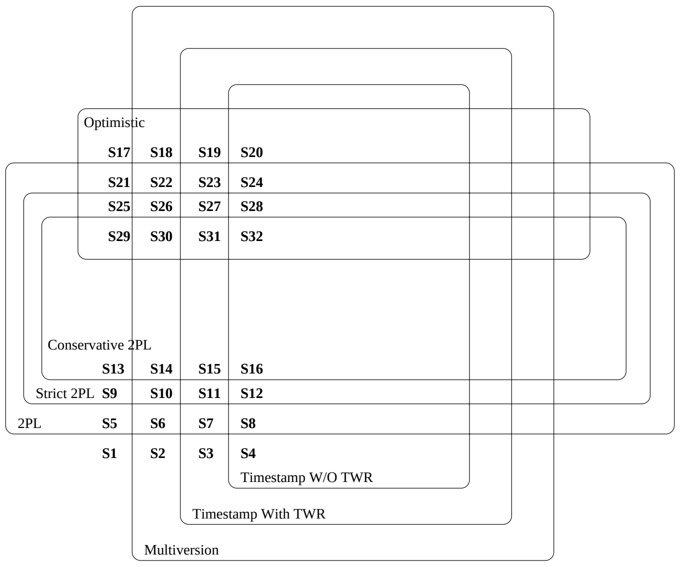

Figure 17.2

- S11
  T1:R(X), T2:W(X), T2:Commit, T1:W(X), T1:Commit, T3:R(X), T3:Commit
- S12
  T1:R(X), T2:R(X), T1:Commit, T2:W(X), T2:Commit

Exercise 17.7 Briefly answer the following questions:

1. Draw a Venn diagram that shows the inclusions between the classes of schedules permitted by the following concurrency control protocols: *2PL, Strict 2PL, Conservative 2PL, Optimistic, Timestamp without the Thomas Write Rule, Timestamp with the Thomas Write Rule,* and *Multiversion*.
2. Give one example schedule for each region in the diagram.
3. Extend the Venn diagram to include serializable and conflict-serializable schedules.

Answer 17.7 The answer to each question is given below.

1. See figure 17.2.
2. (a) Here we define the following schedule first:
   i. C1: T0:R(O),T0:Commit.
   ii. C2: T1:Begin,T2:Begin,T1:W(A),T1:Commit,T2:R(A),T2:Commit.

iii. C3: T4:Begin,T3:Begin,T3:W(B),T3:Commit,T4:W(B),T4:Abort.
iv. C4: T4:Begin,T3:Begin,T3:W(B),T3:Commit,T4:R(B),T4:Abort.
v. C5: T3:Begin,T4:Begin,T4:R(B),T4:Commit,T3:W(B),T3:Commit.
vi. C6: T5:Begin,T6:Begin,T6:R(D),T5:R(C),T5:Commit, T6:W(C),T6:Commit.
vii. C7: T5:Begin,T6:Begin,T6:R(D),T5:R(C),T6:W(C), T5:Commit,T6:Commit.
viii. C8: T5:Begin,T6:Begin,T5:R(C),T6:W(C),T5:R(D), T5:Commit,T6:Commit.

Then we have the following schedule for each region in the diagram.(Please note, S1: C2,C5,C8 means that S1 is the combination of schedule C2,C5,C8.)

i. S1: C2,C5,C8
ii. S2: C2,C4,C8
iii. S3: C2,C3,C8
iv. S4: C2,C8
v. S5: C2,C5,C7
vi. S6: C2,C4,C7
vii. S7: C2,C3,C7
viii. S8: C2,C7
ix. S9: C2,C5,C6
x. S10: C2,C4,C6
xi. S11: C2,C3,C6
xii. S12: C2,C6
xiii. S13: C2,C5
xiv. S14: C2,C4
xv. S15: C2,C3
xvi. S16: C2,C1

And for the rest of 16 schedules, just remove the C2 from the corresponding schedule.(eg, S17: C5,C8, which is made by removing C2 from S1.)

3. See figure 17.3.

Exercise 17.8 Answer each of the following questions briefly. The questions are based on the following relational schema:

Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`, *did:* `integer`)
Dept(<u>*did:* `integer`</u>, *dname:* `string`, *floor:* `integer`)

and on the following update command:

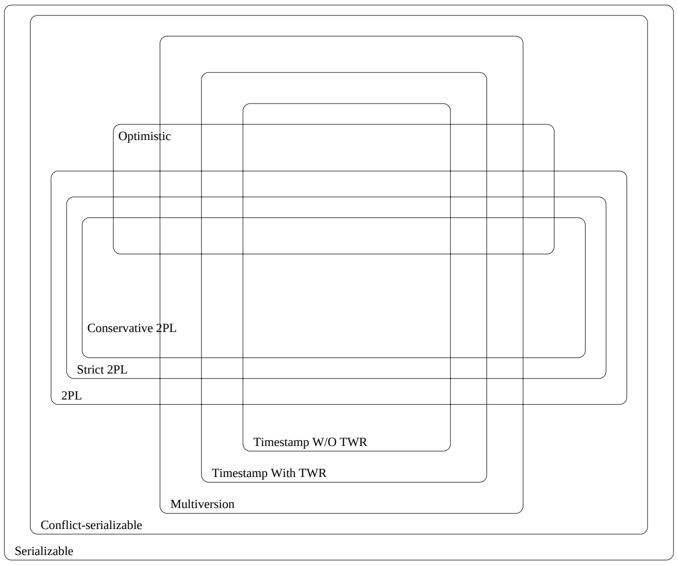

Figure 17.3

replace (salary = 1.1 * EMP.salary) where EMP.ename = 'Santa'

1. Give an example of a query that would conflict with this command (in a concurrency control sense) if both were run at the same time. Explain what could go wrong, and how locking tuples would solve the problem.
2. Give an example of a query or a command that would conflict with this command, such that the conflict could not be resolved by just locking individual tuples or pages but requires index locking.
3. Explain what index locking is and how it resolves the preceding conflict.

Answer 17.8 The answer to each question is given below.

1. One example that would conflict with the above query is:

   Select eid From Emp Where salary = 10,000

   Suppose there are two employees who both have salary of 10,000. If the update command is carried out first, neither of them will be selected; if the select command is done first, both of them should have been selected. However, if these two were run at the same time, one tuple might be updated first causing one to be selected while the other is not. By locking tuples, it is ensured that either all the tuples are updated first or all the tuples go through the selection query first.

2. One example that would conflict with the above query is:

   Insert (EID, ”Santa”, AGE, SALARY, DID) Into Emp

3. If there is an index on a particular field of a relation, a transaction can obtain a lock on one or more index page(s). This will effectively lock all the existing records with the field having a certain range of values, it will also prevent insertion of new records with the field value in that particular range. This will prevent the so-called phantom problem. This technique is called **index locking.**

   It is clear to see that index locking could solve the problem mentioned in 2.

Exercise 17.9 SQL supports four isolation-levels and two access-modes, for a total of eight combinations of isolation-level and access-mode. Each combination implicitly defines a class of transactions; the following questions refer to these eight classes:

1. For each of the eight classes, describe a locking protocol that allows only transactions in this class. Does the locking protocol for a given class make any assumptions about the locking protocols used for other classes? Explain briefly.
2. Consider a schedule generated by the execution of several SQL transactions. Is it guaranteed to be conflict-serializable? to be serializable? to be recoverable?
3. Consider a schedule generated by the execution of several SQL transactions, each of which has `READ ONLY` access-mode. Is it guaranteed to be conflict-serializable? to be serializable? to be recoverable?
4. Consider a schedule generated by the execution of several SQL transactions, each of which has `SERIALIZABLE` isolation-level. Is it guaranteed to be conflict-serializable? to be serializable? to be recoverable?
5. Can you think of a timestamp-based concurrency control scheme that can support the eight classes of SQL transactions?

Answer 17.9 1. The classes `SERIALIZABLE`, `REPEATABLE READ` and `READ COMMITTED` rely on the assumption that other classes obtain exclusive locks before writing objects and hold exclusive locks until the end of the transaction.

   (a) `SERIALIZABLE` + `READ ONLY`: Strict 2PL including locks on a set of objects that it requires to be unchanged. No exclusive locks are granted.
   (b) `SERIALIZABLE` + `READ WRITE`: Strict 2PL including locks on a set of objects that it requires to be unchanged.
   (c) `REPEATABLE READ` + `READ ONLY`: Strict 2PL, only locks individual objects, not sets of objects. No exclusive locks are granted.
   (d) `REPEATABLE READ` + `READ WRITE`: Strict 2PL, only locks individual objects, not sets of objects.

(e) `READ COMMITTED` + `READ ONLY`: Obtains shared locks before reading objects, but these locks are released immediately.

(f) `READ COMMITTED` + `READ WRITE`: Obtains exclusive locks before writing objects, and hold these locks until the end. Obtains shared locks before reading objects, but these locks are released immediately.

(g) `READ UNCOMMITTED` + `READ ONLY`: Do not obtain shared locks before reading objects.

(h) `READ UNCOMMITTED` + `READ WRITE`: Obtains exclusive locks before writing objects, and hold these locks until the end. Does not obtain shared locks before reading objects.

2. Suppose we do not have any requirements for the access-mode and isolation-level of the transaction, then they are not guaranteed to be conflict-serializable, serializable, or recoverable.

3. A schedule generated by the execution of several SQL transactions, each of which having `READ ONLY` access-mode, would be guaranteed to be conflict-serializable, serializable, and recoverable. This is because the only actions are reads so there are no WW, RW, or WR conflicts.

4. A schedule generated by the execution of several SQL transactions, each of which having `SERIALIZABLE` isolation-level, would be guaranteed to be conflict-serializable, serializable, and recoverable. This is because `SERIALIZABLE` isolation level follows strict 2PL.

5. Timestamp locking with wait-die or would wait would be suitable for any `SERIALIZABLE` or `REPEATABLE READ` transaction because these follow strict 2PL. This could be modified to allow `READ COMMITTED` by allowing other transactions with a higher priority to read values changed by this transaction, as long as they didn't need to overwrite the changes. `READ UNCOMMITTED` transactions can only be in the `READ ONLY` access mode, so they can read from any timestamp.

Exercise 17.10 Consider the tree shown in Figure 19.5. Describe the steps involved in executing each of the following operations according to the tree-index concurrency control algorithm discussed in Section 19.3.2, in terms of the order in which nodes are locked, unlocked, read, and written. Be specific about the kind of lock obtained and answer each part independently of the others, always starting with the tree shown in Figure 19.5.

1. Search for data entry 40*.

2. Search for all data entries $k*$ with $k \leq 40$.

3. Insert data entry 62*.

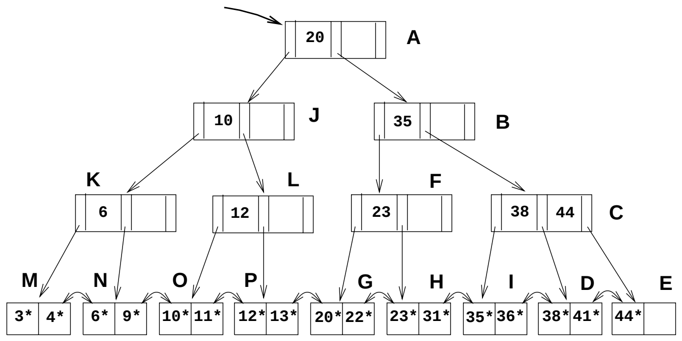

Figure 17.4

4. Insert data entry 40*.
5. Insert data entries 62* and 75*.

Answer 17.10 Please refer to Figure 17.4, and note the abbreviation used:
S(A): Obtains shared lock on A.
X(A): Obtains exclusive lock on A.
RS(A): Release shared lock on A.
RX(A): Release exclusive lock on A.
R(A): Read node A.
W(A): Write node A.

1. S(A),R(A),S(B),RS(A),R(B),S(C),RS(B),R(C),S(D),RS(C),R(D),...(obtain other locks needed for further operation), then release lock on D when the transaction is going to commit(Strict 2PL).
2. S(A), R(A), S(J), S(B), R(J), R(B), RS(A), S(K), S(L), RS(J), S(F), S(C), RS(B), R(K), R(L),
R(F), R(C), S(M), S(N), S(O), S(P), S(G), S(H), S(I), S(D), R(M), R(N), R(O), R(P), R(G),
R(H), R(I), R(D), ..., then release locks on M, N, O, P, G, H, I, D when the transaction is going to commit.
3. X(A), R(A), X(B), R(B), RX(A), X(C), R(C), X(E), R(E), W(E), ..., then release lock on D when the transaction is going to commit.
4. X(A), R(A), X(B), R(B), RX(A), X(F), R(F), RX(B), X(H), R(H), New Node Z, X(Z), X(G), R(G), W(G), W(Z), W(H), W(F), RX(F), ..., then release locks on G, Z, H when the transaction is going to commit.

5. X(A), R(A), X(B), R(B), RX(A), X(C), R(C), X(E), R(E), New Node Z, X(Z), W(E), W(Z), New Node Y, X(Y), W(C), W(Y), W(B), RX(B), RX(C), RX(Y), ..., then release locks on E, Z when the transaction is going to commit.

Exercise 17.11 Consider a database organized in terms of the following hierarchy of objects: The database itself is an object (D), and it contains two files ($F1$ and $F2$), each of which contains 1000 pages ($P1 \ldots P1000$ and $P1001 \ldots P2000$, respectively). Each page contains 100 records, and records are identified as $p : i$, where p is the page identifier and i is the slot of the record on that page.

Multiple-granularity locking is used, with S, X, IS, IX and SIX locks, and database-level, file-level, page-level and record-level locking. For each of the following operations, indicate the sequence of lock requests that must be generated by a transaction that wants to carry out (just) these operations:

1. Read record $P1200 : 5$.
2. Read records $P1200 : 98$ through $P1205 : 2$.
3. Read all (records on all) pages in file $F1$.
4. Read pages $P500$ through $P520$.
5. Read pages $P10$ through $P980$.
6. Read all pages in $F1$ and (based on the values read) modify 10 pages.
7. Delete record $P1200 : 98$. (This is a blind write.)
8. Delete the first record from each page. (Again, these are blind writes.)
9. Delete all records.

Answer 17.11 The answer to each question is given below.

1. IS on D; IS on F2; IS on P1200; S on P1200:5.
2. IS on D; IS on F2; IS on P1200, S on 1201 through 1204, IS on P1205; S on P1200:98/99/100, S on P1205:1/2.
3. IS on D; S on F1
4. IS on D; IS on F1; S on P500 through P520.
5. IS on D; S on F1 (performance hit of locking 970 pages is likely to be higher than other blocked transactions).

6. IS and IX on D; SIX on F1.
7. IX on D; IX on F2; X on P1200.
   (Locking the whole page is not necessary, but it would require some reorganization or compaction.)
8. IX on D; X on F1 and F2.
   (There are many ways to do this, there is a tradeoff between overhead and concurrency.)
9. IX on D; X on F1 and F2.

Exercise 17.12 Suppose that we have only two types of transactions, $T1$ and $T2$. Transactions preserve database consistency when run individually. We have defined several *integrity constraints* such that the DBMS never executes any SQL statement that brings the database into an inconsistent state. Assume that the DBMS does not perform *any* concurrency control. Give an example schedule of two transactions $T1$ and $T2$ that satisfies all these conditions, yet produces a database instance that is not the result of any serial execution of $T1$ and $T2$.

Answer 17.12 We cannot guarantee the correctness of the database just by defining integrity constraints. Consider the case where transaction $T1$ increments all employee's salaries by \$500 and transaction $T2$ decrements all employee's salaries by \$500. If there's no concurrency control when executing these two transactions together, it could possibly overwrite some uncommitted data and increment some of employess's salaries by \$500, decrement some by \$500, and leave no changes on the rest. Conflicts caused by such as interleaving transactions cannot be simply resolved by defining integrity constraints on the database. Thus we need concurrency control.

21 SECURITY

Exercise 21.1 Briefly answer the following questions:

1. Explain the intuition behind the two rules in the Bell-LaPadula model for mandatory access control.
2. Give an example of how covert channels can be used to defeat the Bell-LaPadula model.
3. Give an example of polyinstantiation.
4. Describe a scenario in which mandatory access controls prevent a breach of security that cannot be prevented through discretionary controls.
5. Describe a scenario in which discretionary access controls are required to enforce a security policy that cannot be enforced using only mandatory controls.
6. If a DBMS already supports discretionary and mandatory access controls, is there a need for encryption?
7. Explain the need for each of the following limits in a statistical database system:
   (a) A maximum on the number of queries a user can pose.
   (b) A minimum on the number of tuples involved in answering a query.
   (c) A maximum on the intersection of two queries (i.e., on the number of tuples that both queries examine).
8. Explain the use of an audit trail, with special reference to a statistical database system.
9. What is the role of the DBA with respect to security?
10. Describe AES and its relationship to DES.

11. What is public-key encryption? How does it differ from the encryption approach taken in the Data Encryption Standard (DES), and in what ways is it better than DES?

12. Explain how a company offering services on the Internet could use encryption-based techniques to make its order-entry process secure. Discuss the role of DES, AES, SSL, SET, and digital signatures. Search the Web to find out more about related techniques such as *electronic cash*.

Answer 21.1 The answer to each question is given below.

1. The *Simple Security Property* states that subjects can only interact with objects with a lesser or equal security class. This ensures subjects with low security classes from accessing high security objects. The **-Property* states that subjects can only create objects with a greater or equal security class. This prevents a high security subject from mistakenly creating an object with a low security class (which low security subjects could then access!).

2. One example of a covert channel is in statistical databases. If a malicious subject wants to find the salary of a new employee, and can issue queries to find the average salary in a department, and the total number of current employees in the depatment, then the malicious subject can calculate the new employees salary based on the increase in average salary and number of employees.

3. Say relation R contains the following values:

cid	*carname*	Security Class
1	Honda	U
1	Porsche	C
2	Toyota	C
3	Mazda	C
3	Ferrari	TS

Then subjects with security class U will see R as:

cid	*carname*	Security Class
1	Honda	U

Subjects with security class C will see R as:

cid	*carname*	Security Class
1	Honda	U
1	Porsche	C
2	Toyota	C
3	Mazda	C

Subjects with security class TS will see R as:

cid	*carname*	Security Class
1	Honda	U
1	Porsche	C
2	Toyota	C
3	Mazda	C
3	Ferrari	TS

4. Trojan horse tables are an example where discretionary access controls are not sufficient. If a malicious user creates a table and has access to the source code of some other user with privileges to other tables, then the malicious user can modify the source code to copy tuples from privileged tables to his or her non-privileged table.

5. Manditory access controls do not distinguish between people in the same clearance level so it is not possible to limit permissions to certain users within the same clearance level. Also, it is not possible to give only insert or select privileges to different users in the same level: all users in the same clearance level have select, insert, delete and update privileges.

6. Yes, especially if the data is transmitted over a network in a distributed environment. In these cases it is important to encrypt the data so people 'listening' on the wire cannot directly access the information.

7. (a) If a user can issue an unlimited number of queries, he or she can repeatedly decompose statistical information by gathering the statistics at each level (for example, at age ¿ 20, age ¿ 21, etc.).
   (b) If a malicious subject can query a database and retrieve single rows of statistical information, he or she may be able to isolate sensitive information such as maximum and minimum values.
   (c) Often the information from two queries can be combined to deduce or infer specific values. This is often the case with average and total aggregates. This can be prevented by restricting the tuple overlap between queries.

8. The *audit trail* is a log of updates with the authorization id of the user who issued the update. Since it is possible to infer information from statistical databases using repeated queries, or queries that target a common set of tuples, the DBA can use an audit trail to see which people issued these security-breaking queries.

9. The DBA creates new accounts, ensures that passwords are safe and changed often, assigns mandatory access control levels, and can analyze the audit trail to look for security breaches. They can also assist users with their discretionary permissions.

10. Public-key encryption is an encryption scheme that uses a public encryption key and a private decryption key. These keys are part of one-way functions whose inverse is very difficult to determine (which is why large prime numbers are involved in encryption algorithms...factoring is difficult!). The public key and private key are inverses which allow a user to encrypt any information, but only the person with the private key can decode the messages. DES has only one key and a specific decrypting algorithm. DES decoding can be more difficult and relies on only one key so both the sender and the receiver must know it.

11. A one-way function is a mathematical function whose inversese is very difficult to determine. These are used to determine the public and private keys, and to do the actual decoding: a message is encoding using the function and is decoded using the inverse of the function. Since the inverse is difficult to find, the code can not be broken easily.

12. An internet server could issue each user a public key with which to encrypt his or her data and send it back to the server (which holds all of the private keys). This way users cannot decode other users' messages, and even knowledge of the public key is not sufficient to decode the message. With DES, the encryption key is used both in encryption and decryption so sending keys to users is risky (anyone who intercepts the key can potentially decode the message).

Exercise 21.2 You are the DBA for the VeryFine Toy Company and create a relation called Employees with fields *ename*, *dept*, and *salary*. For authorization reasons, you also define views EmployeeNames (with *ename* as the only attribute) and DeptInfo with fields *dept* and *avgsalary*. The latter lists the average salary for each department.

1. Show the view definition statements for EmployeeNames and DeptInfo.

2. What privileges should be granted to a user who needs to know only average department salaries for the Toy and CS departments?

3. You want to authorize your secretary to fire people (you will probably tell him whom to fire, but you want to be able to delegate this task), to check on who is an employee, and to check on average department salaries. What privileges should you grant?

4. Continuing with the preceding scenario, you do not want your secretary to be able to look at the salaries of individuals. Does your answer to the previous question ensure this? Be specific: Can your secretary possibly find out salaries of *some* individuals (depending on the actual set of tuples), or can your secretary always find out the salary of any individual he wants to?

5. You want to give your secretary the authority to allow other people to read the EmployeeNames view. Show the appropriate command.

6. Your secretary defines two new views using the EmployeeNames view. The first is called AtoRNames and simply selects names that begin with a letter in the range A to R. The second is called HowManyNames and counts the number of names. You are so pleased with this achievement that you decide to give your secretary the right to insert tuples into the EmployeeNames view. Show the appropriate command and describe what privileges your secretary has after this command is executed.

7. Your secretary allows Todd to read the EmployeeNames relation and later quits. You then revoke the secretary's privileges. What happens to Todd's privileges?

8. Give an example of a view update on the preceding schema that cannot be implemented through updates to Employees.

9. You decide to go on an extended vacation, and to make sure that emergencies can be handled, you want to authorize your boss Joe to read and modify the Employees relation and the EmployeeNames relation (and Joe must be able to delegate authority, of course, since he is too far up the management hierarchy to actually do any work). Show the appropriate SQL statements. Can Joe read the DeptInfo view?

10. After returning from your (wonderful) vacation, you see a note from Joe, indicating that he authorized his secretary Mike to read the Employees relation. You want to revoke Mike's `SELECT` privilege on Employees, but you do not want to revoke the rights you gave to Joe, even temporarily. Can you do this in SQL?

11. Later you realize that Joe has been quite busy. He has defined a view called AllNames using the view EmployeeNames, defined another relation called StaffNames that he has access to (but you cannot access), and given his secretary Mike the right to read from the AllNames view. Mike has passed this right on to his friend Susan. You decide that, even at the cost of annoying Joe by revoking some of his privileges, you simply have to take away Mike and Susan's rights to see your data. What `REVOKE` statement would you execute? What rights does Joe have on Employees after this statement is executed? What views are dropped as a consequence?

Answer 21.2

1. EmployeeNames and DeptInfo are defined below:

```
CREATE VIEW EmployeeNames (ename)
       AS SELECT E.ename
          FROM   Employees E
```

```
CREATE VIEW DeptInfo (dept, avgsalary)
       AS SELECT   DISTINCT E.dept, AVG (E.salary) AS avgsalary
          FROM     Employees E
          GROUP BY E.dept
```

2. `SELECT` privilege on the `VIEW` DeptInfo.

   Note that it is impossible to allow the user to access only the average salaries of 'Toy' and 'CS' departments but not those of the other departments. If we really want the average salaries of other departments to be hidden from this user, we have no choice but to create another view.

3. a) `DELETE` on Employees
   b) `SELECT` on EmployeeNames
   c) `SELECT` on DeptInfo

4. No it does not ensure that. It is not possible for the secretary to find out the salary of any employee using just the relations alone.
   If the tuples are such that there is just one employee in a department and the secretary knows this information along with the name of the employee who works there, then he can possibly find out the salary. However, the relations themselves do not allow the secretary to deduce such a fact.

5. `GRANT SELECT ON` Employees `TO` Secretary `WITH GRANT OPTION`

6. `GRANT INSERT ON` Employees `TO` Secretary
   The secretary can now also insert tuples into AtoRNames, which is an updatable view created by the secretary. However, the secretary still cannot insert tuples into HowManyNames because this view is not updatable.

7. Todd's privileges are also revoked.

8. One example of a view update that cannot be implemented through updates to Employees is changing the average salary for a department since one doesn't know which salaries to change.

9. `GRANT SELECT, INSERT, UPDATE ON` Employees `TO` Joe `WITH GRANT OPTION`
   `GRANT SELECT, INSERT, UPDATE ON` EmployeeNames `TO` Joe `WITH GRANT OPTION`
   Joe cannot read the DeptInfo view, but could an identical view.

10. There is no way to do this in SQL: even though you granted privileges to Joe and Joe granted privileges to Mike, you cannot revoke Joe's privileges without also revoking Mike's.

11. Since you don't own AllNames, you can only prevent Mike and Susan from accessing it by revoking Joe's right to read EmployeeNames:

`REVOKE SELECT ON` EmployeeNames `FROM` Joe
The view AllNames is dropped as a consequence. Joe can still modify EmployeeNames without reading it.

Exercise 21.3 You are a painter and have an Internet store where you sell your paintings directly to the public. You would like customers to pay for their purchases with credit cards, and wish to ensure that these electronic transactions are secure.

Assume that Mary wants to purchase your recent painting of the Cornell Uris Library. Answer the following questions.

1. How can you ensure that the user who is purchasing the painting is really Mary?
2. Explain how SSL ensures that the communication of the credit card number is secure. What is the role of a certification authority in this case?
3. Assume that you would like Mary to be able to verify that all your email messages are really sent from you. How can you authenticate your messages without encrypting the actual text?
4. Assume that your customers can also negotiate the price of certain paintings and assume that Mary wants to negotiate the price of your painting of the Madison Terrace. You would like the text of this communication to be private between you and Mary. Explain the advantages and disadvantages of different methods of encrypting your communication with Mary.

Answer 21.3 The answer to each question is given below.

1. In order to determine whether the user who is purchasing the painting is really Mary, we need some level of verification when Mary first registers with the system. On the lowest level, we can simply ask the user to confirm things like Mary's address or social security number. To increase the level of security, we could also ask the user to verify Mary's credit card number. Since these numbers are deemed difficult to obtain, most merchant websites consider this sufficient evidence for proof of identity.

   For an even higher level of security, we can take external steps to verify Mary's information such as calling her up with the phone number provided, sending a letter to Mary's mailing address, or sending her an e-mail with instructions to reply back. In each instance, we attempt to validate the information the user provided so that the element of uncertainty in the provided information is decreased.

2. SSL Encryption is a form of public-key encryption where a third party certification authority acts to validate public keys between two clients. In a general public-key

encryption system, data is sent to a user encrypted with a publicly known key for that user, such that only the user's private key, known only to that user, can be used to decrypt the data. Attempts to decrypt the information using other keys will produce garbage data, and the ability to decipher the private key is considered computationally expensive even for the most modern computing systems.

In SSL Encryption, the client transmitting the data asks the certification authority for a certificate containing public key information about the other client. The first client then validates this information by decrypting the certificate to get the second client's public key. If the decrypted certificate matches up with the certification authority's information, the first client then uses this public key to encrypt a randomly generated session key and sends it to the second client. The first client and second client now have a randomly generated public-key system that they can use to communicate privately without fear that anyone else can decode their information.

Once complete, SSL encryption ensures that data such as credit card information transmitted between the two clients cannot be easily decrypted by others intercepting packets because the certification authority helped to generate a randomly created public-key that only the two clients involved can understand.

3. A message to Mary can be sent unencrypted with a message signature attached to the message. A signature is obtained by applying a one-way function to the message and is considerably smaller than the message itself. Mary can then apply the one-way function and if the results of it match the signature, she'll know it was authentic.

4. One method of sending Mary a message is to create a digital signature for the message by encrypting it twice. First, we encrypt the message using our private key, then we encrypt the results using Mary's public key. The first encryption ensures that the message did indeed come from us, since only we know our private key while the second encryption ensures that only Mary can read the message, since only Mary knows her private key.

   This system is very safe from tampering since it is hard to send a message pretending to be someone else as well as difficult to properly decode an intercepted message. The only disadvantages are that it requires that we have copies of each person's public key as well as spend the time to encrypt/decrypt the messages. For example, if Mary receives the message on her laptop or PDA while traveling, she may not have the resources or public keys to decrypt it and respond, and might need to wait until she returns to the office.

   Another method of communicating with Mary, discussed in the previous question, is to use message signatures. This allows Mary to be able to read the message from almost any system since it is not encrypted and ensure that the message is authentic. The only disadvantage is that it does not safely prevent someone else from reading the message as well.

Exercise 21.4 Consider Exercises 6.6 to 6.9 from Chapter 6. For each exercise, identify what data should be accessible to different groups of users, and write the SQL statements to enforce these access control policies.

Answer 21.4 Please note that there is an impedance mismatch between the level of generality provided by SQL security and the level of granularity provided by most applications. For example, in SQL if we grant a user 'Bob' access to his records in the Customer table, we also give him access to everyone else's records in the Customer table. SQL cannot grant role level security without creating a View for *every user in the database* which is a *very* imcorrect model to use.

Therefore in most current applications, the role level security is reimplemented at the application server level. The following solutions reflect reasonable choices a DBA would make in order to limit some of the database information to entire classes of users.

- *Exercise 6.6*
  Using the Schema defined in the Solution Guide to Exercise 6.6 we define the follow security modifications to the database. In addition, the class of users for the website is denoted GeneralUser.
  - In order to search the database for a record, web users need read access to the record information tables.
    ```
    GRANT SELECT ON Album TO GeneralUser
    GRANT SELECT ON Songs TO GeneralUser
    GRANT SELECT ON Musicians TO GeneralUser
    GRANT SELECT ON Performs TO GeneralUser
    ```
  - To register and login, web users need the ability to insert records into the Users table, as well as read select data from it.
    ```
    CREATE VIEW UserLogin(userid,password)
           AS SELECT U.userid, U.password
              FROM Users U

    GRANT SELECT ON UserLogin TO GeneralUser
    GRANT INSERT ON Users TO GeneralUser
    ```
  - For the shopping basket, no database security is neccessary since this information is usually handled by the application server as temporary state data, i.e., changes are never saved to the database.

- To checkout, web users need the ability to create new orders.
  `GRANT INSERT ON` Orders `TO` GeneralUser

- *Exercise 6.7*

  Using the Schema defined in the Solution Guide to Exercise 6.7 we define the follow security modifications to the database. In addition, the class of users for the doctors is denoted DoctorUser and the class of users for the patients is denoted PatientUser. Both classes are members of the AllUsers Class and inherit all rights from it.

  - In order to access the website and lookup information about drugs, all the users need read access to the pharmacy/drug information tables.

```
GRANT SELECT ON Drug TO AllUsers
GRANT SELECT ON Pharmacy TO AllUsers
GRANT SELECT ON Pham_co TO AllUsers
GRANT SELECT ON Contract TO AllUsers
GRANT SELECT ON Sell TO AllUsers
GRANT SELECT ON Drug TO AllUsers
```

  - To create, modify, and view prescriptions, doctors need special access to the prescriptions table as well as the ability to lookup their patient's names so as to assign prescriptions properly.

```
CREATE VIEW PatientName(ssn,name)
       AS SELECT P.ssn, P.name
          FROM Patient P

GRANT SELECT ON PatientName TO DoctorUser
GRANT INSERT, UPDATE, SELECT ON Prescription TO DoctorUser
```

  - Patients need the ability to change their primary physician, and in order to do so they need to be able to select from a list of doctor names.

```
CREATE VIEW DoctorName(ssn,name)
       AS SELECT D.ssn, D.name
          FROM Patient D

GRANT SELECT ON DoctorName TO PatientUser
GRANT UPDATE (phy_snn) ON Pri_Phy_Patient TO PatientUser
```

  - To view the status of a prescription, patients need partial access to the Prescription table.

```
CREATE VIEW PrescriptionStatus(phy_ssn,date,quatity,tradename,pharmacyname)
       AS SELECT P.phy_ssn, P.date, P.quantity, P.tradename, C.pharmacyname
          FROM Prescription P, Pharmacy_Co C
          WHERE (pssn=P.ssn)

GRANT SELECT ON PrescriptionStatus TO PatientUser
```

  - To checkout, patients need the ability to create new orders.

```
CREATE VIEW PatientAddress(ssn,address)
       AS SELECT U.ssn,U.address
          FROM Users U

GRANT SELECT ON PatientAddress TO PatientUser
GRANT INSERT ON Orders TO PatientUser
```

- *Exercise 6.8*
  Using the Schema defined in the Solution Guide to Exercise 6.8 we define the follow security modifications to the database. In addition, the class of users for the faculty is denoted FacultyUser and the class of users for the students is denoted StudentUser.

  – Faculty needs to be able can view/create/delete classes.
  `GRANT SELECT, INSERT, DELETE ON Class TO FacultyUser`

  – To enroll in classes, students need to be able to view class information and insert information into the enrollment table. The view is neccessary because you do not want students to have access to faculty id numbers of not neccessary.

```
CREATE VIEW ClassInfo(name,meets_at,room,fname)
       AS SELECT C.name, C.meets_at, C.room, F.fname
          FROM Class C, Faculty F
          WHERE (C.fid = F.fid)

GRANT SELECT ON ClassInfo TO StudentUser
GRANT INSERT ON Enrolled TO StudentUser
```

- *Exercise 6.9*
  Using the Schema defined in the Solution Guide to Exercise 6.9 we define the follow security modifications to the database. In addition, the class of users for the employees is denoted EmployeeUser and the class of users for the Passengers is denoted PassengerUser. Both classes are members of the AllUsers Class and inherit all rights from it.

  – In order to access the website and lookup information about flights, all the users need read access to the flight and aircraft information tables.
```
GRANT SELECT ON Flights TO AllUsers
GRANT SELECT ON Aircraft TO AllUsers
GRANT SELECT ON Certified TO AllUsers
```

  – Employees need to be able to add new flights and delete old ones.
  `GRANT INSERT, DELETE ON Flights TO EmployeeUser`

  – Passengers need to be able to make reservations on flights.
  `GRANT INSERT ON Reservation TO PassangerUser`

Exercise 21.5 Consider Exercises 7.7 to 7.9 from Chapter 7. For each exercise, discuss where encryption, SSL, and digital signatures are appropriate.

Answer 21.5 The answer to each question is given below.

- *Exercise 7.7*
  For the Notown Records website, encryption plays an important part in ensuring that customers are able to safely interact and order records over the Internet. Before discussing what should be encrypted, it is important to also note what should not be encrypted. Many of operations including searching the database and browsing record catalogs do not require any encryption. These operations are performed often and encrypting every communication from the client to the website would severely drain the server's resources. As a result, it is better for the server to focus on encrypting only information that is of a more serious nature.

  There are three places where we can apply an encryption scheme to this system: user registration, user login, and user checkout. The purpose of encrypting data at registration and checkout is obvious, the user is transmitting sensitive personal information like address and credit card numbers, and it is of utmost importance to protect this information. We also encrypt the password transmitted during user login since that ensures that future communications with the user are safe after login.

  In practice, we would use SSL encryption via a certification authority, e.g., Verisign. Upon an encryption request, the client's web browser requests the Verisign certificate, validates it by decrypting it, and uses the public key from the decrypted certificate to encrypt a radomly generated session key that it then sends to the server. Using this session key, the certification authority is no longer needed, and the server/client then transmit information just as they would in a normal public key system.

  The Notown website could also use digital signatures to verify a user's registration information via e-mail, as well as send an order confirmation e-mail after an order has been placed. By checking the digital signature of e-mails sent, the website and user can help to double check each other's identities after a new registration has been processed or a transaction has been placed.

- *Exercise 7.8*
  For this question, security is *much* more important than in the previous question, since medical records are considered very serious. As such, we would want to encrypt most of the information transmitted during a doctor or patient's visit to the website. The only information we do not need to transit encrypted would be pages with read-only data, e.g., company information or help pages. Although

this puts a higher strain on the system, the demand for security from users in the medical field is much higher than that of users in the record sales industry.

As before, we would use an SSL encryption via a certification authorization to handle the actual encryption. When a new user registers, it would especially important to verify their identify as to prevent someone else from ordering prescriptions in their name. To this end, digital signatures would be much more important to verify the validity of the e-mail used to sign up with. In addition, external authorization including faxes, phone calls, and written mail should also be used to verify registration information when a user signs up.

- *Exercise 7.9*
  For the course enrollment system, security is important for many of the processes, but the system does not need to encrypt as much as it did in the online pharmacy system.

  For faculty members, their passwords should be encrypted when they login as well as their registration information when they sign up for the first time. When they create/delete existing courses, it is probably not necessary to encrypt their data so long as the system is sure the user is properly logged in. To this end, the system could ask the faculty to re-enter their passwords when they are about to submit a change to the system.

  For students, their login and registration should also be encrypted. Like the faculty system, it is probably not important to encrypt any of the information about what classes a student signs up for as long as the login information is accurate. Students would then be able to freely modify their schedule after logging in to the website and only be asked their password again when they were ready to submit their changes to the system.

  In both cases, SSL encryption would again be the method of choice for the actual encryption process. Digital signatures could be used in e-mails sent to the students confirming their course registration information.

- *Exercise 7.10*
  For the airline reservation website, encryption would again be important in user login and user registration information for both customers and employees. Other information like searching for flights should be left unencrypted as a customer may search for dozens of flights. When a customer purchases a flight, their order, including their flight information and credit card number, should be encrypted to prevent others from learning which flight they have selected, as well as protecting the customer's money. For airline employees, it is preferable to encrypt all of the transmitted data so as not to give hackers the opportunity to see any backend part of the system. Since it is likely there will be few employees per number of

clients, the drain on the system from the extra level of encryption for employees should be negligible.

Note that before when we considered the prescription website, we recommended encrypting all transmitted data, whereas when we considered the course enrollment system we recommended a looser form of encryption on both classes of users. By looser form of encryption, we refer to the fact that some of the transmitted data is encrypted while some data is not. Contrast this with the airline reservation system, where we suggest loose encryption for customers and total encryption for employees. For each, keep in mind that the true level of encryption can change depending on the available resources and sensitivity of the data.

As before, SSL encryption is the preferred method of data encryption. For the airline reservation system, digital signatures can be applied to confirm when customers have places orders via e-mail.