Review Questions

2. a) { EventID -> EventJournal (all attributes) }

b) { JourISBN -> JournalName, JournalName -> JourISBN, JourISBN -> Publisher }

c) The table is not in 3NF. The functional dependency JourISBN -> JournalName is a non-trivial functional dependency where JourISBN is not a superkey of EventJournal, and JournalName - JourISBN = JournalName is not contained in a candidate key of EventJournal. We can use the Compute3NF algorithm to create a 3NF decomposition of EventJournal.

We compute a minimal cover:
F = {EventID -> EventName, EventWebLink, JournalName, Publisher, JourISBN;
    JourISBN -> JournalName;
    JournalName -> JourISBN;
    JourISBN -> Publisher }

Fc = {EventID -> EventName, EventWebLink, Publisher, JourISBN;
    JourISBN ->JournalName;
    JournalName -> JourISBN }

The candidate keys are EventID. We decompose the table into

{(EventID, EventName, EventWebLink, Publisher, JourISBN), (JourISBN, JournalName)}

3. This is not a valid approach, since every non-primary key needs to be fully dependent on the primary key, and O and X are not fully dependent on PK, Y.

4. a) ContractID -> CustomerID, ServiceAddress, ServiceType, StartDate
CustomerID -> ServiceAddress

b) We use the primary key ContractID. This is a superkey of R, and every non-key is fully dependent on it.

c) I didn't list these in part (a) bc I only listed the non-redundant ones so here ya go:
ContractID -> CustomerID
ContractID -> CustomerID, ServiceAddress
ContractID -> CustomerID, ServiceAddress, ServiceType
ContractID -> ServiceAddress
...
I'm sure you get the idea

5. a) Each record takes 15+2+10+10 = 37 bytes. (What does fields starting at a byte that is a multiple of 8 have to do with this?)

b) floor(512/37) = 13. Each block can store 13 records. We need floor(5000/13) = 385 blocks.

c) In the worst-case, we search all the blocks. This would be 385 block accesses.

d) In the worst-case, we search all the blocks. This would be 385 block accesses.

8. Conflict-serializable schedule:
r4[b], r3[a], r2[b], r1[a], w2[b]
r1[d], w1[d],
r2[d], w2[d],
w3[a], r3[c], w3[c],
w4[b], r4[a], w4[a]

Equivalent serial schedule:
r1[a], r1[d], w1[d],
r2[b], w2[b], r2[d], w2[d],
r3[a], w3[a], r3[c], w3[c],
r4[b], w4[b], r4[a], w4[a]

6. Result (same as Temp):

Rating	AvgAge
1	33
3	38.17
7	40
8	40.5
9	35
10	25.5

8. a) T2read(Z), T3write(Z)
T2read(Y), T3write(Y)

T2write(Y), T3read(Y)
T2write(Y), T3write(Y)
T2write(Y), T1read(Y)
T2write(Y), T1write(Y)

T2read(Y), T1write(Y)
T3read(Y), T1write(Y)

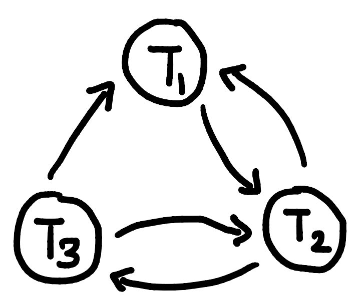
T1read(X), T2write(X)

T1write(X), T2read(X)
T1write(X), T2write(X)

T3write(Y), T1read(Y)
T3write(Y), T1write(Y)

T3write(Z), T2read(Z)

b) This schedule is not conflict-serializable, since there are cycles in its serialization graph.



d) T3 reads item Y after it is written by T2 before T2 has committed (nevermind the fact that there are no commit operations at all), so it is not a recoverable schedule.

1. b)

Email -> Address
CardNumber, ExpiryDate -> Email, Address
CardNumber -> ExpiryDate
Name -> Price, Duration
Email, Name -> RentDuration, Discount, StartTime

c)

A primary key could be Email, CardNumber, Name. It is a candidate key for this relation, is unique, and cannot be reduced.

d)

This is not compliant with 2NF. The non-prime attribute ExpiryDate is dependent on just CardNumber, which is a subset of the primary key. We can decompose the relation as follows:

CardNumber, ExpiryDate
Email, CardNumber
Email, Address
Name, Price, Duration
Email, Name, RentDuration, Discount, StartTime

2. a)

3NF ???

b)

False. It is possible to have every non-prime attribute fully dependent on each candidate key. For example, we can have the relation (A, B, C) where A and B are candidate keys. So we have A -> BC, B -> AC. C is the only non-prime attribute. It is fully dependent on A, and fully dependent on B. Hence this is in 2NF.

c)

We have P -> CX, C -> PX. There are no transitive dependencies.

3. a)

Yes. We can have X -> C1. C1 is in a candidate key, so this does not violate 3NF. But X is not in {P, C1, C2}.

8. a)

```
select Event.EventName, count(TopicID)
from Event
join Covers on Event.EventID = Covers.EventID
group by EventID
```

c)

```
select Event.EventName, count(TopicID)
from Event
join Covers on Event.EventID = Covers.EventID
where substr(Event.EventName, 0, 1) == 'A'
group by EventID
```

d)

```
select Event.EventName, count(TopicID)
from Event
join Covers on Event.EventID = Covers.EventID
group by EventID
having count(TopicID) < 5
```

g)

```
select EventName
from Event
where EventID not in (
	select EventID
	from covers
)
```

1. a)

(slide 17-18)

```
select SalaryGrades.HighestSal - Employees.salary
from SalaryGrades
join Employees on Employees.salary >= SalaryGrades.LowestSal and Employees.salary <= SalaryGrades.HighestSal
where Employees.EmpName = 'Smith'
```

b)

```
select EmpName
from Employees
where salary >= (
	select LowestSal
	from SalaryGrades
	where grade = 'G1'
)
and salary <= (
	select HighestSal
	from SalaryGrades
	where grade = 'G1'
)
```

7.

(slide 19)

i. Superkey

ii. Both

8.

someOtherColumn could have a NULL value (I am pretty sure this does not throw an error though, might just have unexpected behavior since NULL > 5 will eventually evaluate to false?)

9.

Yes.

4. b)

```
select model
from (
	(select model, price from PC)
	union
	(select model, price from Printer)
)
where price >= all (
	(select model, price from PC)
	union
	(select model, price from Printer)

)
```