Exercise 16.3 Consider a database with objects *X* and *Y* and assume that there are two transactions *T*1 and *T*2. Transaction *T*1 reads objects *X* and *Y* and then writes object *X*. Transaction *T*2 reads objects *X* and *Y* and then writes objects *X* and *Y*.

1. Give an example schedule with actions of transactions *T*1 and *T*2 on objects *X* and *Y* that 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**.
2. Give an example schedule with actions of transactions *T*1 and *T*2 on objects *X* and *Y* that results in a read-write conflict.
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X) …
   T2 will get an **unrepeatable read** on X.
3. Give an example schedule with actions of transactions *T*1 and *T*2 on objects *X* and *Y* that results in a write-write conflict.
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:W(X) …
   T2 has **overwritten uncommitted data**.
4. For each of the three schedules, show that Strict 2PL disallows the schedule.
   Strict 2PL resolves these conflict 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) T1 could not get an exclusive on X because T2 would already be holding a shared or exclusive lock on X.
   (c) Same as above.

Exercise 16.7 Consider the university enrollment database schema:

Student(*snum:* `integer`, *sname*: `string`, *major*: `string`, *level*: `string`, *age*: `integer`)
Class(*name:* `string`, *meets_at*: `time`, *room*: `string`, *fid*: `integer`)
Enrolled(*snum:* `string`, *cname*: `string`)
Faculty(*fid*: `integer`, *fname*: `string`, *deptid*: `integer`)

The meaning of these relations is straightforward; for example, Enrolled has one record per student–class pair such that the student is enrolled in the class.

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

1. Enroll a student identified by her *snum* into the class named "Introduction to Database Systems".
   Ans: Because we are inserting a new row in Enrolled, we do not need any lock on the existing rows. So we would use **read uncommitted**.
2. Change enrollment for a student identified by her *snum* from one class to another class.
   Ans: Because we are updating one existing row in Enrolled, we need an exclusive lock on the row which we are updating. So we would use **read committed**.
3. Assign a new faculty member identified by his *fid* to the class with the least number of students.
   Ans: To prevent other transactions from inserting or updating Enrolled while we are reading from it (known as the phantom problem), we would need to use **serializable**.
4. For each class, show the number of students enrolled in the class.
   Ans: Same as above, **serializable**.