Nr.14

a) SERIALIZABLE

Die Transaktion hat in jedem Fall einen gültigen Zustand.
Jegliche Anomalien des Mehrbenutzerbetriebs werden verhindert.
Jedoch kann die Verarbeitung von Transaktionen nicht parallel ablaufen.
Daraus können lange Laufzeiten resultieren.
In diesen Fall würde das Programm verhindern, dass andere Programme auf die entsprechenden Daten zugreifen können. Da das Programm ewig läuft, würden andere Transaktionen, die auf Freigabe der entsprechenden Daten warten, “stillgelegt“. Dadurch könnte die Transaktion, eines anderen Programms keine neuen PC’s einfügen, wodurch dieses Programm jedoch auch nie neue PC’s finden kann.

b) REPEATABLE READ

Dieses Isolationslevel verwendet Sperren auf Objekte, die einem Lese- oder Schreibzugriff unterworfen sind.
Der Beispielcode liest alles aus der Tabelle PC aus. Dadurch werden auch all diese Objekte gesperrt.
Da diese Transaktion permanent läuft, hat auch hier das andere Programm nie die Möglichkeit, neue PC’s in die Tabelle einzufügen und somit kann das gezeigte Programm auch nie solche PC’s finden und läuft ewig und Sperrt die Datenobjekte.

c) READ COMMITTED

Dieses Isolationslevel verhindert das Lesen inkonsistenter Daten. Somit wird ein dirty read verhidert.
Es kann überall dort eingesetzt werden, wo nur ein Lesezugriff auf den letzten konsistenten Zustand in der Datenbank erfolgen muss.
Lesesperren werden sofort wieder freigegeben.
Dadurch kann das andere Programm z.B. in der Wartezeit von 1 Sec. einen entsprechenden PC in die Datenbank einfügen, der dann von dem hier gezeigten Programm (nach der Wartezeit von 1 Sec.) gefunden werden kann.

d) READ UNCOMMITTED

Hier gibt es kein Sperrprotokoll, wodurch das zweite Programm jeder Zeit einen PC in die Datenbank einfügen kann, der dann von dem hier gezeigten Programm gefunden werden kann.
Dieses Isolationslevel birgt jedoch die Gefahr eines dirty read, des Phantom Problems, des Unrepeatable Read und des Lost Update.

Nr.15

a)

1. Möglichkeit (T1, dann T2)

   G1: 4100
   G2: 4100

2. Möglichkeit (T2, dann T1)

   G1: 3000
   G2: 3000

b)

1. Möglichkeit (aus a)

   G1: 4100
   G2: 4100

2. Möglichkeit (aus a)

   G1: 3000
   G2: 3000

3. Möglichkeit (T2 beginnt, T1 unterbricht T2, T2 wird weitergeführt)

   G1: 3000
   G2: 4100

c)

1. Möglichkeit (aus a und b)

   G1: 4100
   G2: 4100

2. Möglichkeit (aus und b)

   G1: 3000
   G2: 3000

3. Möglichkeit (aus b)

   G1: 3000
   G2: 4100

4. Möglichkeit (T1 vor committ abgebrochen)

   G1: 4100
   G2: 3000

Nr.18

Ich würde ein READ_COMMITTED verwenden, da es sich gut für reine Leseoperationen eignet. Es ignoriert Daten, auf denen eine Sperre liegt und liest nur solche, die committed wurden.
Da zwischen dem Erstellen (schreiben) der gewünschten Daten und dem Einlesen offensichtlich eine lange Zeitspanne liegt, sollte ein READ_COMMITTED ausreichen.

Nr. 19

a) Zu dieser Transaktion gibt es keine kompensatorische Transaktion, die die erste wieder aufheben kann. Denn wenn man z.B. ein Gehalt von 100 Euro um 10% erhöht kommt man auf 110 Euro. Würde jedoch eine andere Transaktion dieses Gehalt wieder um 10% senken, käme man auf ein Gehalt von 99 Euro.

b) Auch hier gibt es aus dem gleichen Grund wie in Aufgabe a) keine kompensatorische Transaktion. Jedoch kommt noch ein weiterer Grund hinzu. Denn wenn durch die Gehaltserhöhung ein Mitarbeiter nun mehr als 3000 Euro verdient, würde dieser bei einer kompensatorischen Transaktion nicht berücksichtigt werden.

c) Hier könnte eine kompensatorische Transaktion so aussehen, dass die Note 2 wieder entfernt wird.

d) Auch in diesem Fall könnte eine kompensatorische Transaktion so gestaltet sein, dass der entsprechende Datensatz einfach entfernt wird.

e) Hier könnte eine kompensatorische Transaktion so gestaltet sein, dass sie die Wurzel des entsprechenden Wertes zieht.