

**Soluzione**

1. (A)  (B)  (C)  (D)  (E)  (F)
2. (A)  (B)  (C)  (D)
3. (A)  (B)  (C)  (D)  (E)  (F)
4. (A)  (B)  (C)  (D)
5. (A)  (B)  (C)  (D)
6. (A)  (B)  (C)  (D)
7. (A)  (B)  (C)  (D)
8. (A)  (B)  (C)  (D)

9. Per prevenire SQL Injection è necessario utilizzare *prepared statement* con *query parametriche* oppure *stored procedures*. Nel caso sia inevitabile utilizzare concatenazione di stringhe con valori inseriti da utenti per costruire statement o query SQL da eseguire, è necessario sanitizzare gli input provenienti dagli utenti per assicurare che non contengano comandi SQL dannosi (e.g.: filtrando caratteri o keyword non contentititi).
10. Un approccio *multi-threaded request handling* è generalmente meno efficiente nello scenario indicato. Ciascun thread che gestisce una richiesta spenderà buona parte del tempo in uno stato inattivo, in attesa che gli accessi asincroni al database siano effettuati. Inoltre, un approccio di questo tipo, tipicamente limita a-priori il numero di richieste che possono essere gestite in contemporanea.
11. Nell'ambito del Testing Web End-to-End, la fragilità si riferisce al fatto che il codice per l'esecuzione di test automatici tende a smettere di funzionare in presenza di cambiamenti minori nel layout oppure nell'aspetto delle pagine web dell'applicazione web sotto test. Questo tipicamente avviene perché, in seguito ai cambiamenti evolutivi dell'applicazione web, i selettori utilizzati dal test automatico non riescono più ad identificare correttamente gli elementi con cui interagire.