Bazy Danych 2017
Ostatnia lista zadań

Use [nulls] properly and they work for you,
but abuse them, and they can ruin everything

1. (1 pkt.) Mamy bazę z dwiema tabelami:
`Bombiarze(id int primary key, stopien int)`,
`Agenci(id int primary key, bombiarz int)`.

Bombiarze są podejrzewani o podkładanie bomb w autobusach, agenci ich śledzą, monitorują stopień zagrożenia (atrybut `stopien`) i w przypadku zebrania odpowiednich dowodów zatrzymują do wyjaśnienia. Można założyć, że jesli bombiarz ma przydzielonego agenta to nie będzie w stanie podłożyć bomby. Agenci są dobrze zakonspirowani i czasem ich atrybut `bombiarz` ma wartość NULL - nie wiemy wtedy, którego bombiarza śledzi agent (a nawet czy w ogóle jakiegoś śledzi).

Czy poniższe zapytanie poprawnie wypisuje listę bombiarzy, którzy nie mają przydzielonego agenta?

```
SELECT id FROM Bombiarze
WHERE id NOT IN (SELECT bombiarz FROM Agenci)
```

Odpowiedź uzasadnij. Jeśli uważasz zapytanie za niepoprawne to je popraw.

2. (1 pkt.) Jak odpowiedź na zapytanie `SELECT <expr> FROM Bombiarze` zależy od wyrażenia `<expr>` oraz od tego czy kolumna `stopien` zawiera wartości NULL?

Jako `<expr>` wypróbuj: `stopien` (ile krotek jest w odpowiedzi?), `count(*)`, `count(stopien)`, `count(distinct stopien)`, `max(stopien)`, `sum(stopien)`.

3. (1 pkt.) Pewna firma poprosiła cię o konsultacje dotyczące bezpieczeństwa aplikacji napisanej dla niej przez studentów ii. Poniższy fragment kodu wzbudził twoje wątpliwości. Dlaczego? Spróbuj wykorzystać tę lukę w celu poznania danych tajnych agentów z tabeli `TajniAgenci`?

```
// txtSzukaj - dane wczytane z formularza HTML, operator + to
   konkatenacja napisow
   zapytanie = RunSqlCmd("SELECT imie, nazwisko FROM
      RzecznicyPrasowi
                WHERE nazwisko LIKE '%"+txtSzukaj+"%';", conn);
// wykonanie zapytania i wyswietlenie wszystkich odpowiedzi
```

4. (3 pkt.) Rozważmy następujący stan tabeli `rozl`

```
   typ |  kwota
-------+-------
     1 |    10
     1 |    20
     2 |   100
     2 |   200
```

Przypuśćmy, że w ramach transakcji T1 został wykonany kod PL/pgSQL:
```
SELECT SUM(kwota) INTO suma FROM rozl WHERE typ = 1;
INSERT INTO rozl(typ, kwota) VALUES (2, suma);
```
Równolegle, transakcja T2 wykonała kod:
```
SELECT SUM(kwota) INTO suma FROM rozl WHERE typ = 2;
INSERT INTO rozl(typ, kwota) VALUES (1, suma);
```

Spodziewamy się, że w bazie postgresql w wersji >=9.1 przy ustawieniu ISOLATION LEVEL na REPEATABLE READS obie transakcje zostaną zatwierdzone, a przy ustawieniu ISOLATION LEVEL na SERIALIZABLE jedna z transakcji zostanie wycofana. Dlaczego? Jaki poziom izolacji Postgresql ustawia domyślnie? Omów jak zaimplementowane są poszczególne poziomy izolacji w Postgresql. Wskazówka: odpowiedź znajdziesz w dokumentacji: `https://www.postgresql.org/docs/current/static/transaction-iso.html`.

5. (2 pkt.) Przeczytaj opis implementacji transakcji w grafowej bazie danych Neo4j: *Each transaction is represented as an in-memory object whose state represents writes to the database. This object is supported by a lock manager, which applies write locks to nodes and relationships as they are created, updated, and deleted. On transaction rollback, the transaction object is discarded and the write locks released, whereas on successful completion the transaction is committed to disk.*

Committing data to disk in Neo4j uses a Write Ahead Log, whereby changes are appended as actionable entries in the active transaction log. On transaction commit (...) a commit entry will be written to the log. This causes the log to be flushed to disk, thereby making the changes durable. Once the disk flush has occurred, the changes are applied to the graph itself. After all the changes have been applied to the graph, any write locks associated with the transaction are released.

Załóżmy na potrzeby naszej analizy, że w przypadku operacji czytania transakcja najpierw próbuje odczytać dane ze swojego stanu (czyli widzi własne modyfikacje danych), a jeśli dane nie były modyfikowane to czyta je bezpośrednio z bazy respektując write-locki innych transakcji (tj. oczekując na zakończenie transakcji modyfikującej dane, które zamierza przeczytać).

Co można powiedzieć o poziomie izolacji osiąganym dzięki powyższej implementacji? Przedyskuj możliwość wystąpienia następujących problemów: *dirty read, nonrepeatable read, phantom read, serialization anomaly* —definicje tych pojęć znajdziesz w linku do dokumentacji postgresql z poprzedniego zadania.

6. (2 pkt.) Rozważmy tabele `student(id INT PRIMARY KEY, name TEXT, gender TEXT, birthday DATE)` oraz `zapisy(id INT PRIMARY KEY, sid INT, course TEXT)`. W tabelach przechowujemy dane o wszystkich studentach UWr i ich aktualnych zapisach na zajęcia.

Z pewnych powodów możesz utworzyć tylko jeden indeks (poza tymi stworzonymi dla kluczy głównych). Jaki indeks najlepiej utworzyć w Postgresql aby przyspieszyć wykonywanie poniższego zapytania (`$param` jest parametrem typu `int` uzupełnianym przez aplikację)? Przedstaw kilka propozycji i wybierz najlepszą.

```
SELECT * FROM person p JOIN zapisy z ON (p.id=z.sid)
WHERE p.gender='m' AND EXTRACT(year from age(p.birthday))=$param AND
      z.course='Bazy danych';
```