Ze świata USOS. Część 4 – Ratunku! Moje dane!

Krzysztof STENCEL[*]

Jak każdy system informatyczny, USOS wymaga implementacji rozmaitych algorytmów, których zadaniem jest realizacja procesów gospodarczych (w ramach uczelni). Projektowanie algorytmów to pasjonująca dziedzina aktywności ludzkiej. Wszyscy lubią programować. Reszta informatyki praktycznej wydaje się nudna. Na tę resztę składają się interfejs użytkownika i magazyn danych. Każdy człowiek, który miał styczność z komputerem, wie, jak irytujący bywa marny interfejs użytkownika. Mimo iż zaprojektowanie dobrego interfejsu użytkownika jest nie lada sztuką, sama czynność uchodzi za mało kreatywną (kto lubi wyrównywać pola na formatkach, niech odeśle do Redakcji ten numer *Delty* z dowodami w postaci zeznań co najmniej dwóch świadków; autor tej notki zwróci mu cenę zeszytu). Dziś chcę się zająć czymś pozornie jeszcze nudniejszym – bazą danych. To właśnie tam są gromadzone wszelkie dane o studentach, pracownikach, ocenach, przedmiotach, protokołach itd. Zapytasz, Czytelniku, co może być ciekawego w worku na dane? Wszystko!

Czy chciałbyś, by z tego worka mogło coś zniknąć? Albo nawet, by cały worek licho wzięło? Chociaż taka pokusa może dotyczyć twoich negatywnych ocen, to jednak możliwość przypadkowej lub celowej utraty albo zniekształcenia danych oznaczałaby bezzasadność użytkowania systemu komputerowego przy takiej skali instytucji, jaką jest wyższa uczelnia. Dane należy więc tak zabezpieczyć, by klęski żywiołowe, ludzka głupota, nieuwaga, a także występność nie mogły doprowadzić do utraty danych. Mechanizmy stosowane, by ten cel osiągnąć, są stosunkowo proste. Wielu z Czytelników straciło kiedyś dane i dopiero po tym zdarzeniu zaczęło stosować kopie zapasowe (kto był odpowiednio rozsądny bez wcześniejszej utraty danych, może wysłać pod adresem redakcji kamień, którym autor tej notki pierwszy rzuci w siebie *per procura*). W przypadku bazy danych kopia zapasowa to jednak nie wszystko. Dane przechowane w bazie podlegają ciągłym modyfikacjom. Sytuacja, w której dane wpisane do systemu są tracone, jest wielce niepożądana. Aby tego uniknąć, baza danych prowadzi **dziennik**, tj. zapis wszystkich wykonywanych modyfikacji danych. Zanim użytkownik otrzyma potwierdzenie utrwalenia danych, informacja o tej operacji **musi** być zapisana na trwałe w dzienniku. Zaraz! Przecież zapis na dysku jest kosztowny! Jakże więc zapisywać każdą czynność użytkownika? Racja. Dlatego modyfikacje danych są grupowane w większe jednostki zwane **transakcjami**. Użytkownik dokonuje zmian w danych i gdy zdecyduje się je zapisać, kończy transakcję, żądając zapisu jej efektów do bazy danych.

Tak zwane **zatwierdzenie transakcji** oznacza utrwalenie odpowiednich wpisów dziennika, więc także ostateczne utrwalenie zmienianych danych. Warto pamiętać, że dziennik także może ulec awarii. Podobnie jak kopie zapasowe, należy utrwalać go w kilku oddalonych od siebie miejscach. Można też skorzystać z usług składowania danych w chmurach obliczeniowych. Gdy dojdzie do awarii, roztropny administrator ma kopie zapasowe i dziennik. Odtwarza więc bazę danych z kopii zapasowej. Potem zleca wykonanie na niej wszystkich późniejszych operacji z dziennika. Dzięki temu efekty żadnej zatwierdzonej transakcji nie będą utracone. A nieroztropni administratorzy mogą usłyszeć jedynie *Zaprawdę, powiadam wam, nie znam was. Czuwajcie więc, bo nie znacie dnia ani godziny* (Mt 25, 12–13).

Bardzo wiele osób jest zainteresowanych zawartością naszego worka z danymi. Dla przykładu policzmy, jak to jest na Uniwersytecie Warszawskim. Jest tu około 50 000 studentów i 7000 pracowników. Niemal wszyscy są użytkownikami USOS. Czy worek z otworem na kilkadziesiąt tysięcy rąk lub z kilkudziesięcioma tysiącami otworów wydaje się rzeczą banalną? Warto jeszcze wspomnieć, że ten worek musi być na tyle zmyślny, żeby nie pozwolić na zniszczenie elementu swojej zawartości uchwyconego przez więcej niż jedną rękę. W naszej analogii rozrywanie takiego elementu obrazuje sytuację, gdy dwóch użytkowników USOS jednocześnie chce zmodyfikować jeden rekord z danymi. Gdyby nie było odpowiednich zabezpieczeń, dane równocześnie modyfikowane przez dwie osoby mogłyby łatwo ulec uszkodzeniu. Zajmijmy się najpierw problemem bezpieczeństwa modyfikacji. I tym razem przyda się nam pojęcie transakcji. Przyjmujemy silne założenie, że jeśli zapewnimy bezpieczne przetwarzanie transakcji jako całości, to modyfikacje danych będą bezpieczne.

[*]Instytut Informatyki, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

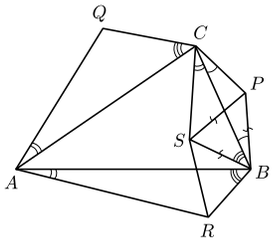
W ramach każdej transakcji poszczególne odczytywane i modyfikowane elementy danych są zabezpieczane **zamkami**. Gdy transakcja odczyta jakąś daną, zakłada na nią zamek, który do jej zakończenia nie pozwala nikomu innemu modyfikować tej danej. Gdy transakcja zmodyfikuje jakąś daną, nie wolno nikomu innemu nawet odczytywać tej danej. Dziecinne proste? Nie do końca. Może się zdarzyć, że jedna transakcja zablokuje jedną daną, a druga transakcja drugą daną, a następnie pierwsza zapragnie drugiej danej, a druga pierwszej. W tym stanie mogą na siebie czekać w nieskończoność. Taką sytuację nazywamy **zakleszczeniem** i, oczywiście, baza danych nie powinna do niej dopuścić. Należy zatem sprawdzić, czy w grafie oczekiwania (krawędź takiego grafu biegnie od transakcji oczekującej do mającej zamek) nie ma cyklu. Gdy jest, wystarczy przerwać jedną z uczestniczących w nim transakcji. Jak często poszukiwać cyklu w tym grafie, gdy może być on naprawdę duży? Czy są inne metody radzenia sobie z zakleszczeniami w bazach danych? To pytania, które sprawiają, że nasz pozornie banalny worek z danymi staje się skomplikowanym i ciekawym oprogramowaniem.

Innym problemem związanym z równoległym dostępem do danych wielu użytkowników jest wydajność. Gdy rozpoczyna się rejestracja na zajęcia, USOS jest równocześnie atakowany przez kilkadziesiąt tysięcy studentów. Jak zapewnić im godziwy czas reakcji systemu? Odpowiedzią jest **replikacja**, czyli powielenie systemu łącznie z jego bazą danych. Baza danych jest zwykle *wąskim gardłem* systemu informatycznego. Powielenie serwera WWW czy serwera aplikacyjnego niewiele może pomóc, jeśli dane też nie będą powielone. Pojawią się więc kopie worka z tymi samymi danymi, a wraz z nimi problem ich **synchronizacji**. Jeśli dane są w wielu kopiach potencjalnie różnych, to która jest ważna? Rozwiązania problemu replikacji i synchronizacji danych są zwykle oparte na podziale uczestniczących baz danych na główne (*master*) i podrzędne (*slave*). Przy czym często zbiór tych pierwszych jest jednoelementowy. W USOS zastosowano hybrydowe podejście replikacji z jedną bazą główną i wieloma podrzędnymi. Teoretycznie baza główna zawsze zawiera dane aktualne, a bazy podrzędne muszą się do niej dostosować. Takie rozwiązanie nie pozwoliłoby jednak na sprawne przeprowadzenie rejestracji na zajęcia, które wymaga bardzo licznych modyfikacji (zapisy studentów) danych w bazach podrzędnych. Główna baza okresowo przekazuje podrzędnej uprawnienia do modyfikacji pewnego podzbioru danych na zasadzie wyłączności. Tak dzieje się w USOS np. przy rejestracji. Na czas rejestracji na zajęcia replika bazy danych dla Wydziału Fizyki otrzymuje prawa do modyfikacji zapisów studentów fizyki. Na czas wystawiania ocen baza danych dla Wydziału Chemii otrzymuje prawa do modyfikacji ocen studentów chemii itd. Szczęśliwie w USOS możliwe jest rozsądne rozwiązanie problemu replikacji danych, które w ogólności może nie istnieć. Zgodnie z tzw. *twierdzeniem CAP*, sformułowanym przez Erica Brewera, nie może istnieć system baz danych spełniający jednocześnie trzy warunki: spójność danych (*Consistency*), dostępność danych (*Availability*) i poprawne obsługiwanie replikacji (*Partition tolerance*). Dowolny system może realizować co najwyżej dwa z tych postulatów. W USOS, oczywiście, zrezygnowaliśmy z C – okresowo każda z replik może mieć inne dane. Widać, że zarówno oprogramowanie, jak i wdrożenie wielu worków z danymi, które pozwalają na wygodne i szybkie przetwarzanie danych przez tysiące użytkowników, nie jest zadaniem banalnym.

Interesujący jest sam format danych w bazach stosowanych w praktyce. Zgodnie z pomysłem Edgara Codda z 1970 roku poszczególne elementy danych przechowywane są w tabelach, z których każda ma skończoną liczbę nazwanych kolumn. W pojedynczych klatkach znajdują się dane podstawowych typów danych, takich jak napisy, liczby, daty. Dane przechowywane w bazach danych są jednak bardzo skomplikowane. Do ich przetwarzania potrzeba więc złożonych algorytmów. Ponad 40 lat temu zdecydowano się na rozwiązanie z prostym modelem danych i ze skomplikowanymi metodami przetwarzania. Taka koncepcja przetrwała próbę czasu.

Projektując bazę danych, musimy więc dokonać zanurzenia złożonego modelu danych instytucji w prosty model danych. Dokonujemy tego zgodnie z postulatem **normalizacji**: każdy rodzaj danej umieszczamy w oddzielnej tabeli. I tak powstają osobne tabele na podstawowe dane osoby, na informacje o immatrykulacji, protokołach, ocenach, pracownikach itd. Jest tych tabel w USOS kilkaset. Gdy przychodzi do użycia tych danych, trzeba połączyć fakty z poszczególnych tabel. Operacja złączenia tabel jest łączna i przemienna, więc kolejność złączania

Rozwiązanie zadania M 1409.
Niech S będzie obrazem punktu C przy obrocie $\mathcal{R}$ wokół P o 90°, jak na rysunku. Wówczas $\sphericalangle SCB = 45^\circ - \sphericalangle BCP = 30^\circ$. Ponadto, skoro $PS = PC = PB$ i $\sphericalangle SPB = 150^\circ - 90^\circ = 60^\circ$, to trójkąt SBP jest równoboczny. Zatem $\sphericalangle SBC = 60^\circ - \sphericalangle PBC = 45^\circ$.



W takim razie trójkąty ABR, CBS są podobne (kkk). Stąd

$$\frac{BA}{BR} = \frac{BC}{BS}.$$

W połączeniu z równością $\sphericalangle ABC = \sphericalangle RBS$ oznacza to, że trójkąty RBS i ABC są podobne (bkb). Zatem $\sphericalangle RSB = \sphericalangle ACB$, więc również $\sphericalangle RSP = \sphericalangle QCP$. Ponadto $PS = PC$ oraz

$$\frac{SR}{CA} = \frac{BR}{BA} = \frac{QC}{CA},$$

czyli $SR = QC$. Tym samym trójkąty PCQ i PSR są przystające (bkb). Z definicji punktu S trójkąt PSR jest obrazem trójkąta PCQ przy obrocie $\mathcal{R}$, co kończy dowód.

Rozwiązanie zadania F 847.
Niech natężenie światła po przejściu przez pierwszy polaryzator wynosi I_1. Zgodnie z prawem Malusa natężenie światła po przejściu przez drugi polaryzator wynosi $I_2 = I_1 \cos^2 \varphi$. Kąt między osiami polaryzacji drugiego i trzeciego polaryzatora wynosi $\frac{\pi}{2} - \varphi$. Natężenie światła za trzecim z polaryzatorów wynosi więc

$$I_3 = I_2 \cos^2\left(\frac{\pi}{2} - \varphi\right) =$$

$$= I_1 \cos^2\varphi \cos^2\left(\frac{\pi}{2} - \varphi\right) =$$

$$= \frac{I_1}{4} \sin^2(2\varphi).$$

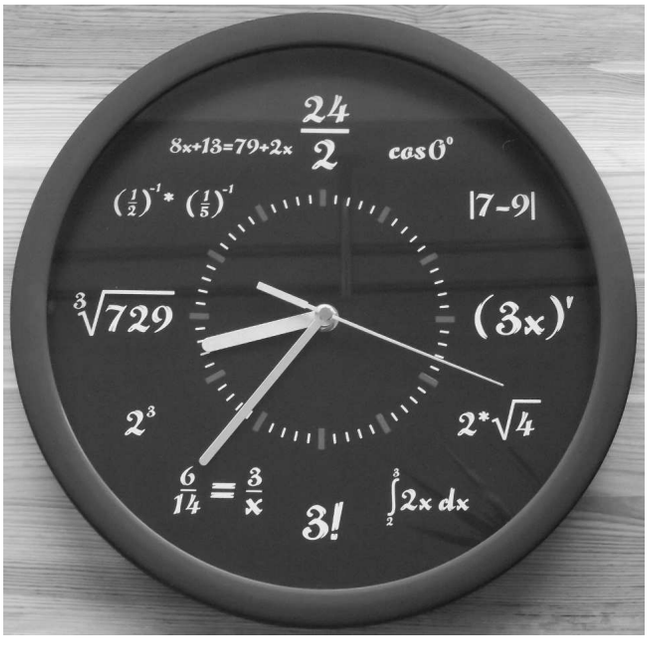
Natężenie światła będzie największe dla $\varphi = 45^\circ$, a najmniejsze (wygaszenie) dla $\varphi = 0$ i $\varphi = \pi/2$, to znaczy dla dodatkowego polaryzatora ustawionego równolegle do osi polaryzacji pierwszego albo ostatniego z polaryzatorów.

jest dowolna. Możemy je łączyć parami według dowolnego planu w postaci drzewa binarnego. Wykonując operacje w odpowiedniej kolejności, możemy uniknąć generowania dużych wyników pośrednich obliczeń. Niestety, okazuje się, że wybór optymalnego kształtu drzewa złączenia jest problemem NP-trudnym. Nie znamy deterministycznego algorytmu o wielomianowej złożoności czasowej. Trzeba więc przeszukać całą przestrzeń możliwych rozwiązań lub zastosować algorytmy sztucznej inteligencji. Jest to cena, jaką płacimy za prostotę modelu danych. Samych kształtów pełnych drzew binarnych o n liściach jest $\binom{2n}{n}/(n+1)$, a trzeba jeszcze przypisać poszczególnym liściom konkretne tabele. W systemie R firmy IBM zastosowano przeszukujący wszystkie plany algorytm o złożoności $O(n2^n)$. Niestety, nawet dla małych n (rzędu kilkunastu) jest to liczba zbyt duża, by taki algorytm mógł być zastosowany w praktyce. Dlatego współczesne systemy zarządzania bazami danych przeszukują tę przestrzeń heurystycznie, najczęściej za pomocą symulowanego wyżarzania lub algorytmów genetycznych. A przecież obliczenie odpowiedzi na zapytanie wymaga nie tylko złączania tabel...

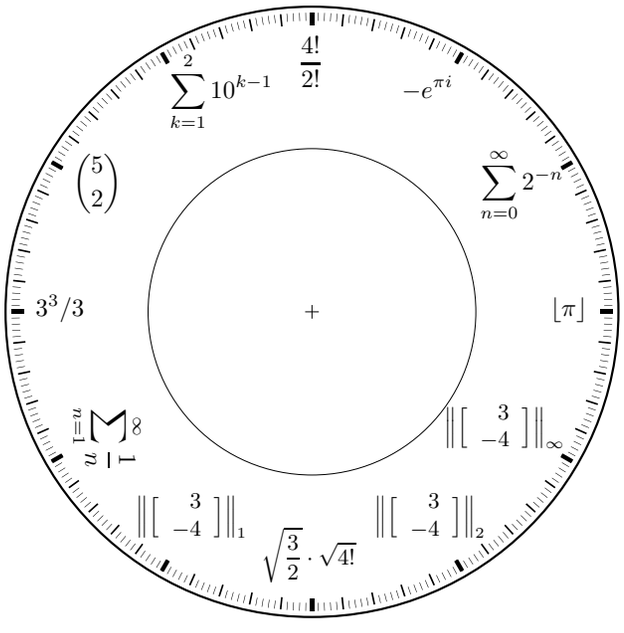
Mam nadzieję, że udało mi się wykazać, iż worek z danymi, taki, jaki zastosowano w USOS, jest niezwykle skomplikowanym tworem. Nie poruszyłem przy tym wielu zagadnień, takich jak indeksy (drzewiaste i haszowane struktury danych służące do wyszukiwania konkretnych wierszy w tabelach), (pół)automatyczne strojenie baz danych (w tym doboru indeksów), kwestia uprawnień użytkowników, zarządzanie fizycznymi nośnikami pamięci, archiwizacja i retencja danych itd. Konstrukcja bazy danych ma wiele wspólnego z konstrukcją cepa. Jest przysłowiowo – ale tylko pozornie – prosta. Każdy może zbudować sobie swój cep, a nawet bazę danych. Ale czy Twoja baza danych będzie działała wydajnie i bez utraty danych przez pół minuty czy kilka lat? Czy bijak Twojego cepa odłączy się od dzierżaka po trzech czy po milionie uderzeń?

Zegar

W prezencie od znajomych dostałem zegar (rys. 1). Powiesiłem go sobie na ścianie i zacząłem wymyślać... inne zegary. Moja propozycja jest na rysunku 2.



Rys. 1



Rys. 2

Czytelników zachęcam do projektowania własnych tarcz zegarowych. Oczekujemy na nie do 1 marca 2014 roku pod adresem `delta@mimuw.edu.pl`. Najlepszy projekt (liczą się wybrane wyrażenia, mogą być też wzięte z fizyki, astronomii lub informatyki) zrealizuję (będzie działał) i wyślę autorowi.

Przemysław KICIAK