Seit-Amet Ailin

Stefan Anca-Mihaela

1240 F

**Pros & cons of using SQLite vs Firebase**

* FireBase este un serviciu cloud si este o bază de date NoSQL. Dacă dorim să salvăm în cloud din dispozitiv, Firebase este util. Pentru a accesa datele, dispozitivul are nevoie de conexiune la internet (cand utilizatorul se deconectează, SDK-urile de baze de date în timp real utilizează memoria cache locală). Cu SQLite, datele sunt stocate pe dispozitiv.
* SQLite este doar datastore-ul în care vom stoca și de unde vom recupera datele. De asemenea, nu este o bază de date distribuită. Dacă dorim să partajăm date între mai mulți utilizatori, nu o putem face cu SQLite.
* Firebase este o bază de date distribuită și oferă realtime streaming support. Dar depinde de utilizare: dacă dorim o aplicație simplă offline, putem utiliza SQLite, dar dacă dorim să construim o aplicație de date distribuite, utilizăm Firebase.
* Firebase este realtime database, în timp ce SQLite este in-process database. Prelucrarea datelor în timp real înseamnă o introducere, procesare și ieșire continuă a datelor.
* SQLite este independent și absolut gratuit. FireBase e bun pentru o dezvoltare rapidă, dar este un serviciu third-party (Firebase Pricing). În orice moment în viitor (Google decide când) poate fi închis. Deci, este o dependență instabilă.
* Firebase este deținută de Google. Firebase este gratuit, cu excepția cazului în care aplicația are o mulțime de utilizatori care apelează API-urile lor. In cazul acesta, se plătește.
* SQLite este practic Firebase direct pe dispozitivul meu (mai puțin lucrurile pe care Firebase le oferă (serviciul lor). Problema e ca, dacă imi sparg telefonul, informațiile care sunt stocate în baza de date SQLite se duc odată cu telefonul.
* Firebase ajută la securizarea operațiunii de mesagerie (notificări push), la raportarea blocărilor și la sincronizarea datelor în timp real (milisecunde). O bază de date SQLite este un fișier simplu stocat local pe dispozitiv. Firebase îl poate găzdui pe un server pentru a-l face global, accesibil de oricine.
* SQLite este un motor de baze de date SQL încorporat. Spre deosebire de majoritatea bazelor de date SQL, SQLite nu are un proces de server separat. SQLite citește și scrie direct pe fișierele obișnuite pe disc. O bază de date SQL completă cu mai multe tabele, indici, declanșatori și vizualizări este conținută într-un singur fișier disc.

**Pros & cons using SQL vs NoSQL**

* SQL:
* Cu o bună structură relațională a bazei de date există redundanță de date foarte mică. Dacă avem o anumită repetare în date, putem doar să o facem referință din multe locuri, fără a duplica datele.
* Putem prelua doar o parte din datele necesare.
* Putem impune reguli privind validarea și integritatea datelor.
* Dacă avem nevoie de tranzacții, adică schimbarea a două lucruri care trebuie făcute împreună sau niciuna dintre ele, este ușor de făcut.
* Dacă se schimbă doar o mică parte din structura de date, trebuie doar să schimbăm acele mici date.
* Dezvoltat în anii 1970 pentru a face față primului val de aplicații de stocare a datelor
* Structura și tipurile de date sunt fixate în avans. Pentru a stoca informații despre un nou element de date, întreaga bază de date trebuie modificată, timp în care baza de date trebuie luată offline.
* Trebuie scalat pe verticală, ceea ce înseamnă că un singur server trebuie să fie din ce în ce mai puternic pentru a face față cererii crescute.
* Bazele de date SQL nu sunt potrivite pentru stocarea ierarhică a datelor.
* Un mix de open-source precum Postgres și MySQL, și comercial precum Oracle Database.
* NoSQL:
* Nu este nevoie de schemă (de asemenea, de proiectarea bazei de date), datele sunt generate din clase și readuse la clase
* Nu este nevoie de SQL JOINs (care sunt adesea încorporate în adnotările ORM)
* Mai ușor de înțeles - datele nu sunt fragmentate
* Mai ușor de întreținut atunci când formatul datelor se schimbă des
* Nu poluează clasele de modele - NoSQL ar putea să nu necesite adnotări sau să necesite cel puțin mai puține dintre ele, aproape că nu necesită extinderea clasei
* Dezvoltat în anii 2000 pentru a face față limitărilor bazelor de date SQL, în special în ceea ce privește replicarea și stocarea datelor nestructurate
* De obicei dinamic. Înregistrările pot adăuga noi informații din mers și, spre deosebire de rândurile tabelelor SQL, datele diferite pot fi stocate împreună, după cum este necesar. Pentru unele baze de date (de exemplu, magazine cu coloane largi), este ceva mai dificil să adăugăm câmpuri noi în mod dinamic.
* Trebuie scalat pe orizontală, ceea ce înseamnă că pentru a adăuga capacitate, un administrator al bazei de date poate adăuga pur și simplu mai multe servere de mărfuri sau instanțe cloud. Baza de date răspândește automat datele pe servere, după cum este necesar.
* Este complet Open-source, dar soluțiile NoSQL gata pentru întreprindere sunt furnizate doar de companii start-up relativ noi și, desigur, la un preț.
* NoSQL oferă câteva facilități pentru interogare și analiză ad-hoc. Este mult mai ușor să codăm o interogare SQL, dar în NoSQL chiar și o interogare simplă necesită o expertiză semnificativă în programare, iar instrumentele BI utilizate în mod obișnuit nu oferă conectivitate la NoSQL.
* Mai potrivit pentru stocarea de date ierarhice, deoarece acceptă key-value pair method.

Dacă avem o mulțime de relații de tip parent-child, atunci ne putem orienta către NoSQL, ceea ce le poate reflecta frumos. Dar dacă există o mulțime de relații ManyToMany, poate că baza de date relațională (SQL) ar putea fi o abordare mai bună - poate economisi mult spațiu.  
NoSQL este excelent atunci când nu avem încă stocarea clar definită. Nu știm ce și cum ar trebui să fie stocate sau schema curentă se poate schimba rapid. Bazele de date cu relații uzuale sunt mai rigide în această privință. Deci, pentru depozitare rapidă și citire NoSQL este perfect.