/Kniha databázové systémy: <http://dbedu.cs.vsb.cz/SubPages/OpenFile.aspx?file=book/dbcb.pdf>

Teorie

Logický přístup - přístup ke stránce datové struktury

Fyzický přístup - přístup ke stránce na disku

Sekvenční průchod tabulkou typu halda

- tabulka se prochází sekvenčně, špatná složitost, ale dobrý výkon čtení

Oracle - TABLE ACCESS (FULL)

SQL Server - TABLE SCAN

Přístup k záznamu pomocí ROWID

- načtení záznamu z tabulky pomocí ID řádku (a bloku)

Oracle - TABLE ACCESS (BY INDEX ROW ID)

SQL Server - RID Lookup (Heap)

Bodový a rozsahový dotaz v indexu

- bodový dotaz projde rychle stromem (indexem) a najde danou hodnotu

- rozsahový dotaz (WHERE x > from AND x < to) projde indexem a najde nejmenší hodnotu (from), poté pokračuje pomocí zřetězených ukazatelů na další hodnoty, dokud nenajde hodnotu vyšší hranice (to)

- používá náhodné čtení, může být pomalejší než sekvenční průchod

Oracle - INDEX (UNIQUE SCAN), INDEX (RANGE SCAN)

SQL Server - INDEX SEEK

IOT/clustered table

- indexově organizovaná tabulka

- bloky uspořádané podle primárního klíče

- implementace obvykle pomocí stromu

Zotavení

Deferred update (NO-UNDO/REDO)

- zapisuje se do memory bufferu, při checkpointu se data zapíšou nejprve do logu a poté do databáze

- jelikož nepotvrzená data nejsou v DB, tak se neprovádí UNDO

- vyžaduje velkou velikost memory bufferu

- do logu se ukládají nové hodnoty

Immediate update (UNDO/[NO-]REDO)

- zapisuje se nejprve do logu a poté okamžitě do databáze

- při zotavení se musí provést UNDO změn, které jsou zapsané v DB a nebyly potvrzené

- do logu se ukládají staré hodnoty6

Savepoint - záchranný bod v transakci, ke kterému se dá navrátit pomocí operace rollback

Souběh

- situace, kdy k DB přistupuje více uživatelů současně, obecněji kdy k jednomu prostředku přistupuje více procesů/vláken současně

Lost update (ztráta aktualizace)

- dvě transakce načtou stejná data a zapíšou je, podle toho, která se provede později, tak její zápis bude platit, druhý zápis je ztracen

- WW konflikt (dirty write)

Uncommitted dependency (nepotvrzená závislost)

- jedna transakce načte nebo zapíše data, která byla zapsána jinou transakcí, ale nejsou potvrzená, později můžou být zrušena rollbackem

- WR konflikt (dirty read)

Inconsistent analysis (nekonzistentní analýza)

- transakce si přečte potvrzená data jiné transakce

- RW konflikt (non-repeatable read)

Uzamykání

Shared lock - sdílený zámek pro čtení, jde ho získat, pokud jiné transakce mají pouze S lock nad daným záznamem

Exclusive lock - exkluzivní zámek pro zápis, jde ho získat, pokud nad daným záznamem nejsou žádné jiné zámky

Tabulkové zámky

Plánované zamykání (intent locking protocol) - vyjadřuje záměr zamknout data nižších vrstev

- pokud chceme zamknout záznam, musíme mít zámek na tabulce

Přísné dvoufázové uzamykání

1) Pro čtení je potřeba S lock

2) Pro zápis je potřeba X lock, pokud už transakce má S lock, transformuje ho na X, pokud má S lock sama

3) Pokud není zámek dostupný, zařadí se transakce do fronty (nesmí čekat nekonečně dlouho - starvation)

4) Teprve po ukončení transakce se uvolní všechny zámky

- řeší ztrátu aktualizace, ale zase zavádí deadlock

- nepotvrzená závislost je tímto zamykáním vyřešena, musí se počkat na COMMIT/ROLLBACK druhé transakce

- řeší nekonzistentní analýzu, ale opět zavádí deadlock

Řešení deadlocku

Detekce - nastavení časových limitů, detekce v grafu

Předcházení (Wait-Die, Wound-Wait)

1) Každé transakci je přiděleno časové razítko

2) Pokud A požaduje zámek na záznam, na který drží zámek B, pak:

a) Wait-Die - pokud je A starší než B, A čeká, pokud ne, tak A je restartována

b) Wound-Wait - Pokud je A mladší než B, A čeká, pokud ne, tak B je restartována

3) Při restartování si transakce nechává své původní časové razítko

Sériový plán - transakce jsou provedeny za sebou

Serializovatelný plán

- výsledek tohoto plánu odpovídá výsledku libovolného sériového plánu (jinak řečeno, výsledek po spuštění transakcí paralelně odpovídá výsledku, který bychom dostali, kdybychom je spustili všechny po sobě, v libovolném pořadí)

- přísné dvoufázové zamykání zaručuje, že plán bude vždy serializovatelný

Úrovně izolace

Read uncommitted - dovoluje číst nepotvrzení změny ostatních transakcí, nebezpečné (nejrychlejší)

Read committed - dovoluje číst potvrzené změny ostatních transakcí (zámky se mohou uvolnit před koncem transakce)

Repeatable read - zajišťuje opakovatelnost čtení, ale můžou se zde vyskytovat fantomové (nově přidané záznamy)

Serializable - zajišťuje kompletní izolovanost transakce od ostatních transakcí, jako jediná úroveň plně zajišťuje vlastnost ACID (nejpomalejší)

Dirty read - čtení nepotvrzených záznamů

Non-repeatable read - čtení potvrzených záznamů (po COMMITu jiné transakce)

Phantom - čtení nově přidaných záznamů

Fenomén DIRTY READ: Klient A provede změnu dat a prozatím neukončí transakci. Klient B přečte tato změněná data. Poté klient A odvolá svou transakci. Klient B tedy přečetl data, která nikdy nebyla potvrzena.

Fenomén NON-REPEATABLE READ: Klient A přečte data a prozatím neukončí transakci. Klient B změní nebo zruší tato data a ukončí svou transakci. Klient A ve své transakci znovu čte stejná data a nenajde je.

Fenomén PHANTOM: Klient A položí dotaz, přečte odpověď na něj a prozatím neukončí transakci. Klient B vloží do databáze další řádky vyhovující podmínkám v dotazu klienta A a ukončí svou transakci. Klient A ve své transakci znovu položí stejný dotaz a obdrží jinou odpověď.

|  |  |  |  |
| --- | --- | --- | --- |
| Stupeň izolace / fenomén | DIRTY READ | NON-REPEATABLE READ | PHANTOM |
| READ UNCOMMITTED | může nastat | může nastat | může nastat |
| READ COMMITTED | nemůže nastat | může nastat | může nastat |
| REPEATABLE READ | nemůže nastat | nemůže nastat | může nastat |
| SERIALIZABLE | nemůže nastat | nemůže nastat | nemůže nastat |

Triggery

- procedury, které se automaticky spouští po úpravě záznamů v tabulce

Balíky

- jmenné prostory umožňující zapouzdření objektů (datových typů, uložených procedur, funkcí, triggerů, výjimek atd.), dovolují označení privátních (pomocných) a veřejných procedur

- rozdělený na deklaraci (veřejná část) a tělo (privátní část, implementace procedur)

Přístupová práva

1) stejné jako má tvůrce procedury - AUTHID DEFINER

2) stejné jako má ten, kdo proceduru spouští - AUTHID CURRENT\_USER

COMMIT WAIT - počká, než se data zapíšou do REDO logu (je zaručeno korektní potvrzení aktualizace)

Otázky

1. Popište odloženou aktualizaci, zda se jedná o UNDO nebo REDO.

- data se zapisují do paměťového bufferu, při kontrolním bodu jsou zapsána do logu a poté na disk

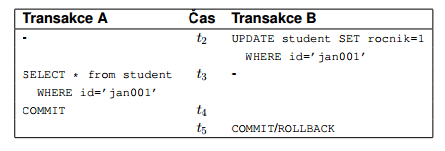
- není třeba provádět UNDO, protože nepotvrzená data nejsou v databázi

- pokud jsou data v logu a ne v databázi, provede se REDO

2. Co může nastat u read uncommited a nemůže nastat u read commited - naznačte na plánu dvou transakci.

DIRTY READ

Plán z přednášky:



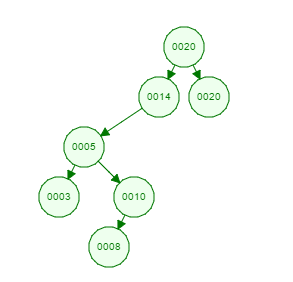
|  |  |  |  |
| --- | --- | --- | --- |
| Stupeň izolace / fenomén | DIRTY READ | NON-REPEATABLE READ | PHANTOM |
| READ UNCOMMITTED | může nastat | může nastat | může nastat |
| READ COMMITTED | nemůže nastat | může nastat | může nastat |
| REPEATABLE READ | nemůže nastat | nemůže nastat | může nastat |
| SERIALIZABLE | nemůže nastat | nemůže nastat | nemůže nastat |

3. Popište plán dotazu SELECT \* FROM tabulka WHERE login = abcde, kde login je index a tabulka je heap table.

1) V indexu se nalezne prvek s loginem abcde (průchod B+ stromem, rychlé, ale náhodný přístup k datům na disku/v bufferu) - UNIQUE SCAN

2) Podle ROWID z indexu se načte záznam z tabulky a vyberou se všechny atributy - TABLE ACCESS BY ROWID / RID Lookup

4. Nakreslete binární vyhledávací strom po vložení čísel 20, 14, 5, 10, 8, 20, 3.



5. Popište, co je SQL injection a jak mu zabranit.

SQL injection je spuštění SQL dotazu obsahujícího neošetřený kód od klienta, který může vést ke kompromitaci databáze.

Příklad:  
Dotaz = "SELECT \* FROM [user] WHERE login = '" + login + "' AND pass = '" + pass + "'"

Execute(Dotaz)

- pokud uživatel (např. ve formuláři) vyplní do loginu např. řetězec

asd' OR 1=1; --

tak se vrátí první uživatel z databáze (a klient se na něho přihlásí), i když nezná jeho přihlašovací údaje

Řešení:

1) Sanizace vstupních řetězců

- nebezpečné, pokud to programujeme sami, téměř jistě na něco zapomeneme, ale i pokud používáme knihovny, může se stát, že velmi speciálně zkonstruovaný vstup (např. v nějaké čínském znakové sadě) bude moct injektovat škodlivý kód

- musíme použít, pokud nelze využít parametrizované dotazy (např. DDL dotazy)

2) Parametrizované dotazy (implicitní, pokud se používá statické SQL)

- správné řešení, které nejprve na server odešle dotaz s "placeholdery" pro vstupní data, a až poté samotná data

- zajistí se tak, že server bere vstup uživatele opravdu jako data, a ne jako direktivy dotazu

- zároveň se tak dá optimalizovat spouštění více stejných dotazů s různými daty, protože server už má předpočtený plán dotazu a pouze ho spouští znovu nad různými daty (pokud je ale dat opravdu hodně, mělo by se spíše použít bulk vkládání)

3) Omezit přístupová práva

- skript my měl mít co nejméně přístupových práv, tak, aby mohl provádět jen to, co má, a nemohl například zasahovat do ostatních částí databáze

6. Jakým způsobem lze u ORM minimalizovat počet přístupů k databázi, případně omezit zatížení sítě?

Měli bychom využít znalosti konkrétní databáze a navrhnout ORM tak, abychom stahovali data s co nejmenším počtem dotazů (je např. lepší použít jeden SELECT dotaz, který vrátí milion výsledků, než 1000 SELECT dotazů, kde každý vrátí 1000 výsledků). Místo více dotazů tedy můžeme použít spojování tabulek. Měli bychom taky stahovat pouze ta data, které potřebujeme. Pokud navrhujeme funkci, která generuje velký počet mezivýsledků, měla by být naimplementovaná na straně databáze (jako uložená procedura/funkce), aby se mezivýsledky nemusely zbytečně přenášet přes síť.

1. Co je to ACID?

Každá transakce musí splňovat vlastnost ACID: atomičnost (angl. atomicity), korektnost (angl. correctness), izolovanost (angl. isolation) a trvalost (angl. durability):

A - Atomičnost – transakce musí být atomická: jsou provedeny všechny operace transakce nebo žádná.

C - Korektnost – transakce převádí korektní stav databáze do jiného korektního stavu databáze, mezi začátkem a koncem transakce nemusí být databáze v korektním stavu.

I - Izolovanost – transakce jsou navzájem izolovány: změny provedené jednou transakcí jsou pro ostatní transakce viditelné až po provedení COMMIT.

D - Trvalost – jakmile je transakce potvrzena, změny v databázi se stávají trvalými i po případném pádu systému.

2) Co nemůže nastat u REPEATABLE READ oproti READ COMMITED?

NON-REPEATABLE READ

|  |  |  |  |
| --- | --- | --- | --- |
| Stupeň izolace / fenomén | DIRTY READ | NON-REPEATABLE READ | PHANTOM |
| READ UNCOMMITTED | může nastat | může nastat | může nastat |
| READ COMMITTED | nemůže nastat | může nastat | může nastat |
| REPEATABLE READ | nemůže nastat | nemůže nastat | může nastat |
| SERIALIZABLE | nemůže nastat | nemůže nastat | nemůže nastat |

3) Halda, vkládání, vyhledávání a jejich složitost.

Tabulka typu halda je stránkované perzistentní pole, vkládání proběhne tak, že se záznam vloží na první prázdné místo, složitost tedy má O(1). Vyhledávání je relativně složitá operace se složitostí O(n), musí se sekvenčně projít všechny záznamy.

4) Rozdíl mezi statickým a dynamickým SQL.

Statické SQL je kompilováno a používá implicitně parametrizované dotazy. Používá se například v procedurálních rozšířeních DBŘS jako PL/SQL a T-SQL nebo v SQL rozšířeních jiných jazyků (vložené SQL v C). Dynamické SQL se oproti tomu zasílá na DBŘS až v době jeho běhu, nelze tedy předvídat, jak dotazy budou vypadat a výkonnost je nižší, dotazy lze ale tvořit libovolně (jako řetězec), což sice může přinášet bezpečnostní problémy, ale dovoluje tvořit libovolné dotazy (JDD - definice dat).

5) Napište 4-5 vlastností objektově-relačního modelu (ve srovnání s relačním modelem).

Vlastní (složené) strukturované datové typy.

Funkce, procedury, triggery a další kód na straně serveru.

Kolekce dat (vnořené tabulky, pole).

Dědičnost datových typů.

Datové typy ukazatelů na data.

Můžeme datovým typům přiřazovat chování (metody).

6) Co je to doménový objekt a DAO v ORM? (Uvedtě příklad).

Doménový objekt reprezentuje objekt z domény aplikace (např. Auto v IS autopůjčovny). DAO (data access object) poskytuje rozhraní k práci s DTO (data transfer object), které slouží jako přepravky dat a jsou mapovány na konkrétní záznam v tabulce. DAO představuje obvykle tabulku samotnou a její funkce (vyhledání konkrétního záznamu, smazání konkrétního záznamu, načtení všech záznamů), DTO představuje jeden záznam (jeho data).

1. Seriový plán, serializovatelný plán.

Sériový plán je vykonání transakcí za sebou sériově. Serializovatelný plán je ekvivalentní s libovolným sériovým plánem (viz teorie nahoře).

2. Co je LOG, k čemu slouží a kde se používá?

Záznam na disku, do kterého se zapisují veškeré operace změn v databázi. Zápis je dopředný (nejprve do logu, až pak do fyzického úložiště na disku), abychom mohli korektně provádět obnovu databáze. Zápis do něho se provádí sekvenčně, což je mnohem rychlejší než náhodný zápis, proto si může DBŘS dovolit zapisovat do něho záznamy o všech změnách.

V Případě operace ROLLBACK na základě záznamů uložených v logu je systém schopen vrátit příslušné hodnoty záznamu na původní hodnoty. V případě odložené aktualizace se díky logu provádí operace REDO. U okamžité aktualizace můžeme vrátit původní hodnoty operací UNDO.

3. Repeatable read

Úroveň serializace, která nedovoluje špinavá čtení ani problém nekonzistenční analýzy. Stále se ale v ní může objevovat problém fantomů, tj. nově přidaných dat do databáze, který může ovlivnit výsledky dotazů.

4. Složený index

Složený index se skládá z více než jednoho klíče, je vhodný použít tehdy, pokud často používáme selekci dle více atributů. Pokud ale máme ve složeném indexu klíč např. z atributů (A, B) a používáme selekci pouze podle B, bude se provádět sekvenční průchod, tj. index nebude využit. Je to z toho důvodu, že je složený index uložen (a hlavně seřazen) v daném pořadí, takže ho v tomto případě můžeme využít pouze pro selekci nad A nebo A, B dohromady.

5. Výhoda procedur na serveru

Kód používá statické SQL, je tedy rychlejší, než pokud bychom dotazy posílali dynamicky za běhu. Pokud procedura používá mezivýpočty, tak ty nejsou zbytečně odesílány přes síť ke klientovi a průběh je opět rychlejší. Pokud databázi používá více různých klientů, můžeme tak zaručit jednotnost chování databázových funkcí a kontrolu korektnosti databáze máme na jednom místě, nemusíme ji pokaždé implementovat znovu a implementace tedy není závislá na platformě (i když je závislá na DBMS, to ale obvykle není problém). Nevýhodou je nutnost psát v procedurálním rozšíření DBŘS, které často nedosahuje kvalit vyšších programovacích jazyků, a přenesení zátěže výpočetní logiky na jedno místo (databáze).

6. B-tree

Strom, který má c - 2 \* c prvků (klíčů) v uzlu, každý uzel má maximálně (2 \* c) + 1 potomků, je tedy ((2 \* c) + 1)-ární. c je kladné přirozené číslo. Kořen jako jediný uzel smí mít méně než c prvků. Složitost operací vyhledání, mazání a vkládání je logc(n). Zaručuje faktor využití (počet položek uzlu / c v procentech) alespoň 50 %. V každém uzlu je c - 2\*c klíčů (kromě kořene, kde může být 1 - 2\*c klíčů), které jsou seřazeny, mezi i.tým a i+1.tým klíčem je ukazatel na potomky, ve kterém jsou klíče s hodnotami >= i.tého klíče a <= (i + 1).tého klíče. Každý uzel je buď listový (nemá potomky), anebo má m klíčů a m+1 potomků. Všechny listové stránky jsou na stejné úrovni.

<http://i.imgur.com/E9nl4Xr.png>

7. Halda

Tabulka typu halda je stránkované persistentní pole. Záznamy nejsou fyzicky mazány, jsou pouze označeny jako smazané (pro skutečné smazání musí být proveden shrinking). Při vkládání je záznam umístěn na první volnou pozici nebo na konec pole.

Neefektivní vyhledávání (O(n)) – nelze se spoléhat na uspořádání záznamů v tabulce).

Efektivní z pohledu využití místa a operace INSERT, která má konstantní složitost (O(1)).

8. Aplikační objekty v ORM

DTO (data transfer object) - objekt přenášející data. Představuje jeden záznam (řádek) z tabulky.

9. Rozdíl mezi Repeatable read a serializable

· U urovně izolace Serializable jsou všechny řádky uzamčeny po dobu trvání transakce.

· U RR je možné vkládání nových řádku do datového souboru.

· RR:

· K uvolnění zámku S a X dochází až po ukončení transakce

· Problémy: výskyt fantomů = pokud provedeme stejný dotaz ( select) v transakci vícekrát získáme jiný počet záznamů

· Serializable:

· K uvolnění zámku S a X dochází až po ukončení transakce

· Nejvyšší stupeň izolace

· Snižuje propustnost databaze

· Nemá problémy

10. Kandidát na index ( výhody X nevýhody )

Primární nebo cizí klíč nebo atribut často se vyskytující v klauzuli where.

Výhody: rychlejší prohledávání tabulek.

Nevýhody: každý index znamená zvýšení počtu operací při změnách v databázi

11. Kdy je lepší použít uloženou proceduru na serveru než psát v klientské aplikaci?

· Pokud chceme přenést funkcionalitu na server, což nám zaručí jednotnou logiku. Pokud funkce využívá hodně mezivýpočtů, tak provedení na serveru šetří data zasílaná po síti.

12. ROW ID + klíč v indexu

· Index umožnuje rychlejší vyhledávání pomocí klíče, ke klíči je přiřazeno číslo rowid které slouží jako odkaz do tabulky typu halda.

· ROWID = Jednoznačně identifikuje řádek v rámci jedné tabulky ( jedinečné číslo označující záznam tabulky, popř. jeho blok a offset)

13. Plán vykonávání dotazu

1. převod do interní formy (parsování sql, vytvoření stromu dotazu)

2. převod do kanonické formy (hledání optimální formy, přeskládání příkazů - projekce jako první, selekce před spojením, nahrazení příkazů efektivnějšími příkazy)

3. vygenerování optimalních plánů (ocenění podle IO a CPU cost, a podle vstupních dat, indexů, apod.) a výběr nejlepšího plánu (používá se heuristika, plán není optimální, protože čas hledání plánu nesmí přesáhnout ušetřený čas)

14. TRIGGER

· Blok kodu spuštěný v závislosti na příkazu insert update nebo delete.

· V triggeru nemůžeme upravovat stejnou tabulku na kterou je trigger nastaven

· PL – SQL :

o CREATE OR REPLACE TRIGGER název

o [after,before] [insert update delete] on tabulka for each row

o Begin

o …

§ Přístup k novým hodnotám pomocí :new …

§ Přístup ke starým hodnotám pomocí :old …

o End

· T-SQL :

o CREATE TRIGGER

o ON TABULKA [FOR INSERT UPDATE DELETE]

o BEGIN

§ Přístup k novým hodnotám pomocí

§ Select id from inserted

o End

15. MVC

· Je návrhový vzor, jenž systém rozděluje do tří částí tak, aby každá část spravovala svou část funkcionality a nebyly na sobě závislé.

· Složení:

o Model = obsahuje logiku aplikace a aplikační data

o View = grafické rozhraní, slouží k zobrazení dat z modelu a k interakci s uživatelem

o Controller ~~=~~ přebírá data z view a předává je modelu ke zpracování, poté se stará o překreslení view

· Cíl = model je nezávislý na view a controlleru

· Výhody:

o Snadná změna uživatelského rozhraní

o Rozdělení práce

o Možnost použít logiku i pro více rozhraní

16. Nekonzistentní analýza

· Transakce A načítá hodnoty entic, ale než je stihne načíst všechny, tak transakce B aktualizuje některé hodnoty

· Řešení pomocí zámků

- non-repeatable read

17. Popsat plán vykonání dotazu "select\* from student where login=@login", kdy login je indexem.

V indexu (obvykle B+ strom) se najde klíč login, pokud je unikátní, jedná se o UNIQUE SCAN, pokud ne, tak se najde nejnižší hodnota a poté se jde dále pomocí zřetězených odkazů v listech (RANGE SCAN). Takto se najde rowid záznamu v tabulce typu halda a k tomu se přistoupí a načtou se všechna data.