Statistiky v dotazech

T(R) - počet záznamů v relaci R
S(R) - průměrná velikost záznamů v relaci R v bajtech
S(R,A) - průměrná velikost atributu A v relaci R v bajtech
B(R) - počet obsazených bloků relací R
V(R, A) - počet unikátních hodnot atributu A v relaci R

Odhady velikostí

Kartézký součin

$W = R_1 \times R_2$
$T(W) = T(R_1) \cdot T(R_2)$
$S(W) = S(R_1) + S(R_2)$

Selekce

$W = \sigma_{A=`wat`}R$
$T(W) = \frac{T(R)}{V(R,A)}$ ..... Pouze při rovnoměrném rozložení hodnot v R.
$S(W) = S(R)$

$W = \sigma_{A\neq`wat`}R$
$T(W) = T(R) - \frac{T(R)}{V(R,A)}$ ..... Pouze při rovnoměrném rozložení hodnot v R.
$S(W) = S(R)$

U operací ≤, ≥, <, > záleží na velikosti rozsahu.

$W = \sigma_{A=`wat`\vee B=`wat`}R$
$$T(W) = T(R) \cdot \left(1 - \left(\frac{1 - \frac{T(R)}{V(R,A)}}{T(R)}\right) \cdot \left(\frac{1 - \frac{T(R)}{V(R,B)}}{T(R)}\right)\right)$$
$S(W) = S(R)$

Spojení

$W = R_1 \bowtie R_2$
$$T(W) = \frac{T(R_1) \cdot T(R_2)}{max\{V(R_1, A_1), V(R_2, A_1)\} \cdot \cdots \cdot max\{V(R_1, A_n), V(R_2, A_n)\}}$$
$S(W) = S(R_1) + S(R_2) - S(R_1, Z), kde\ Z\ jsou\ společné\ atributy\ R_1\ a\ R_2$

Sjednocení

$T(R \cup S) = avg\{max\{T(R), T(S)\}, T(R) + T(S)\}$

Rozdíl

$T(R - S) = T(R) - {}^1/_2\, T(S)$

Průnik

$T(R \cap S) = avg\{0, min\{T(R), T(S)\}\}$

Raidová pole

Diskové pole - více fyzických disků tvořících jeden logický. Zvětšuje kapacitu, dovoluje paralelizaci R/W.

Mirroring - Zvyšuje spolehlivost. Zápis se provádí na všechny disky, číst lze z libovolného. Výpadek jednoho disku je druhý dostupný (u poruchy řadiče vypadne vše).

Data Stipping - zvýšení propustnosti, často paralelizace čtení. Dvě formy: Bit a Block. Bit rozděluje každý bajt, přístupová doba je ale horší. Block ukládá bloky postupně na různé disky, čtení přes více bloků lze paralelizovat.

RAID

RAID0

Block stiping - vysoký výkon, kapacita a spolehlivost beze změn.

RAID1

Mirroring - kapacita snížena 1/n, rychlé čtení, zápis stejný.

RAID2

Bit stiping, samoopravné Hammingovi kódy, zotaví se po výpadku jednoho disku, nutno synchronizovat disky. NEPOUŽÍVANÉ

RAID3

Byte striping, jeden paritní disk, oprava dat z vypadlého disku je XOR bajtů z ostatních disků, nutno synchronizovat disky. Minimálně 3 disky. NEPOUŽÍVANÉ

RAID4

Block striping, jeden paritní disk, větší rychlost než RAID3, není nutná synchronizace. Velká zátěž na paritní disk. Minimálně 3 disky. NEPOUŽÍVANÉ

RAID5

Rozděluje paritu rovnoměrně mezi disky. Řeší problém velké zátěže na paritní disk u RAID4. Vyšší výkon než RAID4 - jeho náhrada. POUŽÍVANÉ

RAID6

Podobné RAID5, ale obsahuje „dvojitou paritu“. Používá samoopravné Hammingovi kódy, řešící výpadek až dvou disků. Minimálně 4 disky. Kapacita snížena o dva paritní disky. POUŽÍVANÉ pro velkokapacitní disky.

RAID1+0

Zmirrorivané stripy. Odolnost proti výpadkům, vypadnout může jeden disk v libovolném RAID1 poli. POUŽÍVANÉ, používá se i RAID5+0

RAID0+1

Stripované mirrory. Vypadne-li cokoliv v jednom RAID0, celá jeho větev vypadne. NEPOUŽÍVANÉ

Materializované pohledy v databázích

Definujte, co je to materializovaný pohled

Pohled je databázový objekt, který poskytuje data, ale nejsou v něm přímo uložena. Je to jen předpis pro jejich získání. Slouží pro získávání dat (jde i rozšířit na aktualizace). U komplikovaných pohledů může být dotaz zpomalen, protože data musí být získána při každém použití. Toto lze (krom indexů) řešit i, typicky u agregačních pohledů, materializovaným pohledem. Výsledek dotazu získaného přes tento pohled je uložen v tabulce. Jedná se o formu optimalizace, protože si výsledek dotazu můžeme přepočítat. Obvykle se materializované pohledy aktualizují. Optimalizátor ho může použít k přepsání dotazu.

Určete výhodu a nevýhodu používání materializovaných pohledů

Výhodou je nemuset počítat drahý, a/nebo často používaný dotaz. Problémem je že MV zabírá místo na disku, takže by se nemělo jednat o velké a složité relace. Z toho důvodu je často použit pro výsledky agregačních funkcí. Dalším problémem je nutnost MV aktualizovat, o to se ale často postará databázový systém sám.

Popište techniku vytvoření a aktualizace materializovaného pohledu, pokud to databáze přímo nepodporuje

Vytváření - vytvoření nové tabulky
Aktualizace - trigger

Uvažujte relaci *hodnocení(učo, předmět, kredity, známka1, známka2, známka3)*, T(*hodnocení*)=250. Předpokládejte, že relace je uložena v sekvenčním souboru uspořádaném podle primárního klíče a že povolené známky jsou A-F a NULL.

Pro atribut *známka1* zkonstruujte bitmapový index a popište jeho princip

Bitmapový index je druh indexu, kde každé unikátní hodnotě přiřadíme bitové pole. Index je tedy kolekce h bitových polí, kde h je počet unikátních hodnot atributu. Délka poje je rovna počtu záznamů (v našem případě 250), jedná se o invertovaný index. Tam, kde se vyskytuje daná hodnota, je v příslušném poli „1“, jinak „0“. Index se hodí typicky pro atributy s malým počtem hodnot, použitelné i pro rozsahové indexy. Indexy se dobře kombinují, bitové operace jsou rychlé. Index je ale paměťově náročný, a aktualizace záznamů bývá taky problém.

S informacemi ze zadání neumím sestavit bitmapový index přesně.

Představte techniku RLE pro ukládání bitového vektoru a následující sekvenci dekódujte. 1110 1010 1011 00

Bitmapový index je problematický v tom, že je velmi dlouhý a často obsahuje velmi mnoho po sobě jdoucích nul. Řešením je komprese RLE. Počet nul mezi jedničkami (i nula nul) zakóduji do binárního čísla. Před toto binární číslo přidám vyjádření počtu bitů, které toto binární číslo zabírá v unární soustavě (jedničky zakončeny nulou). Nuly na konci se ignorují. Výsledný RLE kód je uspořádaná posloupnost těchto dvojic.

1110 1010 -> 0000 0000 001
10 11 -> 0001
0 0 -> 1
Výsledek -> 0000 0000 0010 0011

Uvažujte aplikaci, která bude shromažďovat meteorologické údaje ze stanic umístěných na libovolném místě zeměkoule. Aplikace bude přijímat záznamy, které budou obsahovat: čas měření (s přesností na vteřiny), aktuální teplotu (s přesností na jedné desetiny stupně), množství srážek od posledního měření (v desetinách mm) a textovou identifikaci meteostanice.

Navrhněte vhodnou relaci, která bude dané záznamy ukládat (názvy atributů a datové typy).

mereni(id, cas_mereni, teplota, rozdil_srazek, meteostanice)
cas_mereni <- bigint
teplota <- double
rozdil_srazek <- double
meteostanice <- varchar(25)

Pro uvedené atributy popište vhodný způsob uložení

Protože v relačním schématu mám řetězec s proměnlivou délkou, navrhuji použít proměnlivou délku schématu, a šetřit tím místo. Protože ale zbytek atribut má pevnou délku, stojí za to uvažovat i o kompromisu mezi fixní a proměnlivou délkou.

Uveďte formát pro uložení záznamů relace, který podporuje NULL hodnoty

V tomto případě je určitě vhodné schéma s proměnlivou délkou, protože hodnota NULL je vlastně typ položky, jejíž hodnotu již dále nepotřebujeme uložit.

Databázové systémy a vysoká dostupnost

Definujte pojem dostupnosti, co znamená pojem pět devítek

Dostupnost je vyjádření v procentech, jak často je služba dostupná za jednotku času. Definováno jako $Av = (totaltime - downtime) / totaltime$. Downtime máme dva druhy - plánované a neplánované, při vysoké dostupnosti ale tento rozdíl nebereme v úvahu.

Pojem devítek je rozdělení do tříd dostupnosti, čím vyšší třída, tím vyšší dostupnost. Vzorec pro určení třídy je $c = \lfloor -log_{10}(1 - Av) \rfloor$. Konkrétně třída pěti devítek (třída 5) odpovídá dostupnosti 99.999% což se jistě dá považovat za vysokou dostupnost.

Popište princip master-slave replikace, včetně uvedení DML příkazů, které lze/nelze na jednotlivých uzlech vykonat.

Jedná se o distribuovaný model - replikační, který používá load-balancing na čtecí dotazy. Hodí se tedy při velmi častém čtení. Master je manažer dat a distribuuje změny slaveům. Slave ukládá data, provádí dotazy nad nimi, ale nemodifikuje.

Představte techniku log-shipping realizující master-slave replikaci.

Známo také jako „warm standby“. Primární uzel obhospodařuje všechny dotazy, je v permanentním archivačním módu - posílá WAL recordy do standby uzlů. Standby uzly nezpracovávají žádné dotazy, jsou v permanentním recorvery módu - zpracovávají WAL recordy z primárního uzlu. Při failoveru přebere standby úlohu primárního.

Implemetace operací v databázích pomocí proudového zpracování

Vysvětlete princip proudového zpracování

Operaci můžeme řešit postupně a zpracovávat záznam po záznamu. Výsledek se generuje postupně a nikoliv naráz. Není tak paměťově náročný, protože nemusí ukládat.

Představte rozhraní iterátoru, které je vhodné pro proudové zpracování

Podpůrný koncept pro proudové zpracování. Skládá se ze tří částí. *Open* - inicializuje operaci, připraví kontext na postupné vracení výsledků. *GetNext* - vrácení dalšího řádku výsledku, který se dále zpracovává. *Close* - ukončuje operaci, popřípadě uvolní buffer.

Zformulujte algoritmus pro vyhodnocení operace DISTINCT pro relaci R libovolné velikosti

Základní myšlenkou je postupně záznam po záznamu zjišťovat, zdali je záznam už obsažen ve výstupu. Není-li, přidá se do něj. Problémem je, že sekvenční testování je pomalé (kvadratická složitost). Řešením je použít interní strukturu, například hashovací tabulku, které nám zajistí rychlé vyhledání.

Určete náklady tohoto algoritmu na vyhodnocení (čtení/zápis bloků) pro relaci o velikosti B (R)=1000 a jeho nároky na operační paměť (max. M=50).

U jednoprůchodového zpracování narážíme na problém. Neplatí totiž podmínka $B(R) < M$. Je tedy nutné využít dvouprůchodového přístupu. Zde je podmínka $B(R) \leq M^2$, kterou již splňujeme, dokonce dosahujeme podmínky optimality $M \geq \sqrt{B(R)}$. Náklady na tento algoritmus tedy jsou $3B(R)$.

Zhodnoťte vlastnosti tohoto algoritmu pro proudové zpracování.

V přípravné fázi je nutno vykonat celý mergeSort, což je bohužel ta nejdražší část. Nicméně připravené dávky již nemusíme spojovat (ušetříme), stačí nám postupně brát vždy nejmenší prvek z připravených dávek. K tomu může posloužit upravený iterátor, který bere do úvahu více zdrojů a bere aktuální nejmenší prvek. Duplicitní záznamy lze přeskočit, protože máme informaci o posledním zpracovaném záznamu a díky seřazení víme, že pokud není stejný, není ani v již zpracované množině.

Uvažujte relace *employee(e_id, name, phone, room)*, *project(p_id, title, budget)* a *works_on(p_id, e_id, role)*, kde atributy *e_id* a *p_id* v relaci *works_on* jsou cizí klíče do příslušných relací (*employee* resp. *project*) a na atributu *project.title* je definováno integritní omezení UNIQUE. Uvažujte dotaz, který vybírá identifikátory a jména zaměstnanců, kteří pracují na projektu s názvem `Registr vozidel` na pozici testerů. Ve výsledku se záznamy nesmí opakovat.

Formulujte pro tento dotaz výraz SQL

```
SELECT e.e_id, e.name
FROM works_on w
        NATURAL JOIN project p
        NATURAL JOIN employee e
WHERE p.title=`Registr vozidel` AND w.role=`Tester`;
```

Rozhodněte a zdůvodněte, zda je nutné v tomto výrazu použít DISTINCT nebo nikoli

Relace *employee* je privilegovaná, protože obsahuje svůj primární klíč ve výsledku dotazu. Relace *works_on* je závislá na privilegované relaci *employee* a relace *project* je tranzitivně závislá na relaci *employee*. Z definice tedy platí že DISTINCT není potřeba.

Optimalizace vytvořením indexu. Předpokládejte relaci *R(A, B, C)*, která má primární klíč a pro kterou platí T(R)=300 000, B(R)=1608 a V(R, B)=T(R)/5, a následující dávku (workload) skládající se z příkazů (x, y, z jsou libovolné konstanty):
sa: SELECT * FROM R WHERE R.A=x
sb: SELECT * FROM R WHERE R.B=y
in: INSERT INTO R(A, B, C) VALUES (x, y, z)
Pravděpodobnosti spuštění jednotlivých příkazů p_{sa}, p_{sb}, p_{in} se součtem 1.
Rozhodněte, kdy se vyplatí vytvořit index nad atributem B, pokud
- Každý index má náklady na vyhledání konkrétní hodnoty 3 čtení
- Každý index má náklady na vložení nové hodnoty 3 čtení a 1 zápis
- Každé vložení nového řádku do relace znamená 1 čtení a 1 zápis na disk

Uveďte náklady v přístupech na disk pro jednotlivé příkazy a obě situace (bez/s indexem pro B).

Bez indexu

- sa: 3 + 1 = načtení indexu + načtení bloku = 4
- sb: B(R) = table scan a filter = 1608
- in: 4 + 2 = vložení indexu + vložení do relace = 6

S indexem

- sa: 3 + 1 = načtení indexu + načtení bloku = 4
- sb: 3 + T(R)/V(R,B) = načtení indexu + načtení bloku = 8
- in: 4 + 4 + 2 = vložení indexu + vložení indexu + vložení do relace = 10

Zformulujte vzorce nákladů provedení celé dávky v obou případech (bez/s indexem pro B).

Bez indexu

- $4p_{sa} + 1608p_{sb} + 6p_{in}$

Bez indexu

- $4p_{sa} + 8p_{sb} + 10p_{in}$

Určete a zdůvodněte, pro jaké hodnoty pravděpodobnosti má smysl vytvořit index pro B.

p_{sa} nebude ovlivněno.

$1608p_{sb} + 6p_{in} \lessgtr 8p_{sb} + 10p_{in}$
$1600p_{sb} \lessgtr 4p_{in}$
$400p_{sb} \lessgtr p_{in}$

$p_{in} < 400p_{sb}$ -> index se vyplatí
$p_{in} > 400p_{sb}$ -> index se nevyplatí

Uvažujte relaci student(id, jméno, příjmení)

Popište princip generování unikátních hodnot atributu id, který se obvykle ve vyspělých DB systémech používá, a uveďte případ konkrétního využití.

Záleží na databázovém systému. Oracle mají Sekcence. SQLServer mají IDENTITY. Sekvence je například nastavitelný generátor čísel max/min hodnota, cyklická vlastnost.

Pomocí jazyka SQL realizujte naivní metodu generování unikátních hodnot.

mojeTabulkaId := UPDATE klice SET id=id+1 WHERE tabulka=`mojeTabulka` RETURNING id;
INSERT INTO mojeTabulka VALUES (mojeTabulkaId, `Roman`, `Tyčka`);

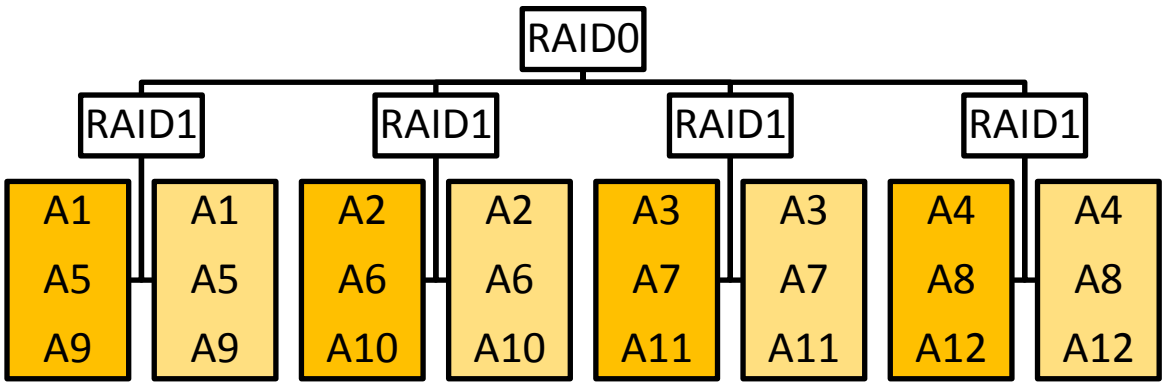
Uveďte a zdůvodněte zásadní výhodu postupu z prvního bodu oproti naivnímu přístupu ve druhém bodě.

Zde je rozdíl v rychlosti u paralelního přístupu. Build-in konstrukty databázového systému jsou na tom vždy lépe než naivní přístup. Umožnují zpracovat mnohem více paralelních dotazů.

Uvažujte diskové pole typu RAID10 složené ze stejných běžně dostupných disků. Každý disk má kapacitu 1TB a průměrná rychlost čtení i zápisu je 100MiB/s. Velikost diskového bloku je totožná s velikostí DB bloku a to 4Kib.

Navrhněte pole, které bude mít úložnou kapacitu alespoň 3,5TB, nakreslete jeho schéma a stručně popište princip ukládání dat.

Abychom dosáhly požadované kapacity na poli RAID10 s danými disky, potřebujeme 8 takových disků.



RAID0 nastripuje data mezi 4 disky, kapacita se nezmění, ale zvýší se výkon. Každá z těchto disků se stripy je pak zmirrorován pomocí RAID1, tím se zajistí větší bezpečnosti díky redundanci. Vypadnout může libovolný jeden disk z RAID1 a data zůstanou zachována. Navíc se zdvojnásobuje rychlost čtení.

Uveďte, jak bude dané pole provádět čtení jednoho bloku a z čeho se skládají náklady na toto čtení.

RAID0 nejprve vybere ten správný RAID1 s daným blokem ((i mod n)+1), v něm se přečte blok z jednoho z disků, nebo z obou paralelně.

Náklady se stávají z: velikosti bloku, rychlosti čtení, rychlosti vystavení hlavy, overheadu controlleru

Určete počet požadavků na jeden blok, které je navržené pole schopné provést současně - zvlášť pro požadavky na čtení a zvlášť pro požadavky na zápis. Své tvrzení zdůvodněte.

Čtení z jednoho stejného bloku, nebo z boku na stejném RAID1 -> až dva požadavky. Toto platí pro každé RAID1 pole zvlášť. V nejlepším případě tedy až 8 paralelních čtení.

Zápis není RAID1 urychlen, takže nejlépe 4 paralelní čtení, pokud se ale zapisuje do bloků na různých discích.

Horizontální dělení relací

Popište co je horizontální dělení a jak pracuje

Rozdělování tabulky podle řádků. Použití pro snížení objemu dat, se kterými se pracuje. Usnadňuje i mazání.

Uveďte zásadní důvod pro použití horizontálního dělení.

Archivace dat, load-balanching mezi slave nody, ...

Uvažujte DB systém, který horizontální dělení nepodporuje. Navrhněte horizontální dělení relace *počasí(id, stanice, datum_čas, aktuální_teplota)*, když víte, že každou hodinu jsou ukládány nové teploty z 10000 stanic a uživatelé se dotazují nejčastěji na vývoj teplot na všech stanicích za posledních 72 hodin. Diskutujte důsledky nasazení tohoto řešení pro aplikace vkládající záznamy a dotazující se na vývoj teplot.

Vytvořil bych job, který by záznamy starší než 72 hodin přesunul do archivační tabulky, dobu spouštění bych nastavil tak, aby se archivace vykonávala v době nejnižšího vytížení systému.

Dále bych musel zajistit, aby se dotaz na starší záznamy přesměroval na archivační tabulku. K tomu může sloužit procedura a popřípadě i view.