POGLAVLJE

4

Arhitektura DBMS sistema

Ovo poglavlje se bavi jezgrom SQL Servera 2000, a ono se zove Database Management System -DBMS (sistem za upravljanje bazom podataka). Upoznaćete se sa njegovom arhitekturom za skladištenje podataka, kao i ulazom i izlazom podataka. Inače, arhitektura i procesi koji su razmatrani u ovom poglavlju uglavnom su povezani sa mašinom za skladištenje SQL Servera, dok su arhitektura i procesi koji su razmotreni u drugom poglavlju uglavnom vezani za relacionu mašinu. U osnovi, sistemi za upravljanje bazama podataka koji se zasnivaju na SQL-u rade na istim principima, isto kao što svi automobili imaju gume, gas i mašinu za sagorevanje. Ipak, neki sistemi rade bolje nego drugi i svaki DBMS sistem je jedinstven u tome kako u algoritmu za obradu upita koristi memoriju, procesor, diskove.

Dok čitate ovo poglavlje izgledaće vam kao da sam zalutao u opštu teoriju i arhitekturu i da sam se udaljio od SQL Servera 2000. Ali, nije tako. Ako ste novi u radu sa bazama podataka, K/S DBMS sistemima, SQL-om i SQL Serverom, onda ćete imati koristi od opšte diskusije o tome kako DBMS sistemi i SQL Server konkretno rade. Takođe, nije loše da proverite tvrdnje da SQL Server prednjači u mnogim oblastima obrade transakcije i da je trenutno najprilagodljiviji proizvod koji postoji. Ovo poglavlje će vam pomoći da se u to uverite.

Prvo treba da shvatite kako DBMS sistem radi, a onda da naučite kako radi SQL Server u pogledu na opšti DBMS sistem ili relacionu teoriju. Zbog toga nije loša ideja da kombinujete ovo poglavlje sa nekom dobrom knjigom koja se odnosi na principe sistema relacionih baza podataka. Kombinacija ova dva znanja će vam pomoći da bolje procenite SQL Server, da shvatite kako on radi za vas. Ovo će vam sigurno pomoći da postanete bolji DBA ili SQL Server programer kada se u narednim poglavljima uhvatite u koštac sa naprednijim stvarima.

Ako ste iskusni u radu sa SQL Serverom ili kao K/S DBMS inženjer, možete prvi deo ovog poglavlja samo da prelistate. Ako ste, pak, novi u radu sa SQL Serverom, preporučujem da prvo shvatite kako DBMS radi. Uz razumevanje ostalih komponenti i resursa, ovo će vam olakšati kasniji posao.

U glavnom delu konceptualno ćemo se pozabaviti unutrašnjim okruženjem SQL Servera, odnosno načinom na koji podaci dolaze u DBMS sistem, kako se čuvaju, vraćaju i šalju do klijenata. Ipak, da se prvo upoznamo sa nekim važnim elementima. Za početak, govoriću o subjektu oko kog svi proizvođači podižu tako puno buke. U pitanju je Transaction Processing Performance (TPC), reper za obradu transakcija.

TPC reperi

Trebalo bi da se potrudite da shvatite podatke TPC repera, koje svi važni proizvođači baza podataka publikuju. Microsoft je publikovao određeni broj repera u toku 2000. godine, prema kojima je SQL Server 2000 postao vodeći DBMS sistem na tržištu sa najnižom cenom po transakciji. Šta to znači za vas?

TPC reperi su za DBMS sisteme ono što su MIPS rang-liste za procesore ili što je konjska snaga za motor sa unutrašnjim sagorevanjem. DBMS sistem mora da zadovolji određene repere u pogledu performansi pre nego što se može smatrati vrednim pažnje ili konkurentnim u oblasti online obrade transakcija i sistema za podršku odlučivanju. Reperi se koriste za testiranje algoritma, novih verzija koda, novih hardverskih platformi itd. Zbog toga proizvođači sistema za upravljanje bazama podataka uključujući Microsoft, Informix, IBM, Oracle, Sybase i ostale, koriste repere za testiranje tačnosti i pouzdanosti sistema na određenim platformama i u određenim probnim okruženjima. TPC reper testovi su događaji kao što su trke formule 1 u automobilizmu. Proizvođači ne samo da se takmiče da dokažu superiornost ili udeo na tržištu, već u tim trkama uče i dobijene rezultate ubacuju u svoje proizvode.

Ove testove vrši neprofitabilna organizacija Transaction Processing Performance Council (TPC), koja se na Internetu može naći na adresi www.tpc.org. Njihov cilj je da definišu testove i da prikažu objektivne performanse na osnovu repera. Reperi imaju ograničene zahteve testiranja i sadrže određeni broj provera koje se odnose na trajanje i pouzdanost. Rezultati testova se nezavisno objavljuju.

Za SQL Server 2000 dva važna repera su TPC-C i TPC-H. Oni se odnose na Online Transaction Processing (OLTP) i na sisteme za podršku odlučivanju (DSS, razmotreni u četvrtom delu). Mogu se opisati na sledeći način:

- **TPC-C: OLTP** TPC-C služi za merenje performansi i prilagodljivosti OLTP DBMS sistema. TPC-C testira širok niz funkcija baze podataka, kao što su operacije selektovanja, ažuriranja i grupne transakcije. Najvažnije je, ipak, da TPC-C meri prolaz u poslovnim transakcijama u minuti. Ovo se radi za simulirani unos stavki i za distribuisano okruženje. Testovi mere koliko se novih narudžbi može generisati za minut, dok DBMS sistem izvršava četiri ostala tipa transakcija, kao što su plaćanje, ažuriranje statusa, otpremanje ili isporuka i ažuriranje na nivou skladišta.

  Prolaz, u TPC terminologiji, predstavlja meru maksimalno podržanih performansi sistema. Pet tipova transakcija ima unapred određene zahteve u pogledu vremena odziva. Vreme odziva na novu narudžbinu je podešeno na pet sekundi. Prema tome, ako vaš sistem generiše broj 500 tpmC, to znači da je sistem generisao 500 novih narudžbina za minut, ispunjavajući pri tome i četiri druga zahteva u TPC-C testu. Kada pogledate Internet aplikacije ili e-commerce rešenja, videćete da TPC-C reper postaje izuzetno važan. U vreme dok ovo pišem, TPC-C reper za SQL Server 2000 na serveru Compaq ProLiant 8500 daje rezultate koji su prikazani u narednoj tabeli. Ovo je izvedeno iz statistike koju su zajedno publikovali Microsoft i Compaq.

Hardver	TpmC	Cena/tpmC	Ukupna cena sistema
Compaq PrLiant 8500-700-96P (12 čvorova)	262,243	#20.24	$5.3M
Compaq ProLiant 8500-550-96P (12 čvorova)	229,913	$23.08	$5.3M
Compaq ProLiant 8500-700-64P (8 čvorova)	179,658	$19.75	$3.5M
Compaq ProLiant 8500-550-64P(8 čvorova)	161,719	$21.86	$3.5M

  Microsoft i Compaq su ove rezultate postigli sa SQL Serverom 2000 Enterprise Edition, pod Windowsom 2000 Advanced Server sa upotrebom klastera od 8 i 12 Compaq ProLiant 85000 servera baze podataka. Svaki server je imao procesor Intel Pentium III Xeon 700 MHz ili 550 MHz i GB memorije. Pored toga, sistemi koji su mereni imali su uključene 24-36 aplikacionih servera (3 po čvoru) koji su radili pod Microsoftovim Web serverom, odnosno pod Internet Information Serverom. Compaqov ServerNet II je obezbedio visoke brzine veze između servera baze podataka.

- **TPC-H: DSS** Ovaj reper, izražen kao TPC-H Composite Queries (složeni upiti) na sat (QphH), stimuliše sistem za podršku odlučivanju koji sadrži velike količine podataka koji su sinhronizovani sa online proizvodnim bazama podataka. Reper koristi veoma složene ad-hok upite koji su tako napravljeni da odgovaraju realnim problemima. Tu spadaju upiti koji se odnose na cenu, promociju (kao što je ciljno tržište), snabdevanje na zahtev, zaradu i gubitke itd. Ne bi trebalo da bude iznenađenje ako kažemo da SQL Server 2000 prednjači i u oblasti TPC-H repera.

TPC-C i TPC-H takođe proizvode /cenu/performanse brojeva, $.tpmC i $.QphH opsege, respektivno. Ovi brojevi su posebno korisni kada se određuje cena transakcija ili performanse analize. Opseg nije samo posledica platforme ili računara, pošto predstavlja samo mali deo kriterijuma cene DBMS sistema. Ovi opsezi uzimaju u obzir celo računarsko okruženje, softver, operativne sisteme, DBMS softver, trogodišnji plan održavanja itd.

Evo primera: Kako ćete izvesti $/tpmC za OLTP sistem? Uzme se cena celog sistema i podeli tpmC reperom. U reperima koji su ilustrovani u prethodnom primeru uzimamo cenu celog sistema (koja za prvo rešenje iznosi $5.3 miliona) i podelimo sa tpmC, odnosno 262,243, tako da se dobije $/tpmC od $20.24. Nema sumnje da bi visokoprilagodljiva serverska platforma (a ova Compaqova platforma bi se mogla smatrati malim centrom podataka) dala čak i bolje rezultate nego ovaj tpmC broj, pošto je IBM RS/6000 sa 24 procesora dao četiri puta veću $/tpmC za približno isti tpmC.

Vrlo je važno da se pozabavite i prilagodljivošću, što ćemo mi uraditi u trinaestom poglavlju. Kada se dođe do SML prilagodljivosti, često manje znači više, posebno kod DBMS sistema. Linearna SMP prilagodljivost je sveti zadatak za administratore baze podataka. Mnogi smatraju da su sistemi srednjeg nivoa ili UNIX sistemi sa podrškom operativnog sistema za stotine procesora odgovor na prilagodljivost DBMS sistema. Mnogi, takođe, ukazuju na tzv. manu Windowsa 2000 da ne može da podrži toliko procesora kao njegov rival UNIX. Ovi reperi sada dokazuju da je SMP prilagodljivost mit kada se radi o DBMS sistemima.

NAPOMENA

TPC prati i druge repere, kao što je TPC-R opseg koji meri performanse generisanja standardizovanog izveštaja. Više o reperima možete saznati na adresi www.tpc.org. ■

Pregled komponenti sistema za upravljanje bazom podataka

SQL Server 2000, kao DBMS sistem, sadrži veći broj međusobno povezanih izlazno/ulaznih arhitektura ili modula. Pre nego što pokušate da programirate sa SQL Serverom 2000, trebalo bi da se upoznate sa sledećim ključnim komponentama:

- **RELACIONA MAŠINA BAZE PODATAKA** Ova komponenta sadrži nekoliko modula ili poveznaih komponenti, koje sarađuju jedna sa drugom na bazi zajedničkog koda. S mašinom za skladištenje i njenim modulima, relaciona mašina baze podataka čini DBMS sistem. Sistem sadrži i module za obradu iskaza (kako za SQL tako i za XML iskaze), kao i sistem za obradu upita. Mašina prima svu komunikaciju od klijenta i prosleđuje je kao događaj do mrežne biblioteke. Prvo se analizira upit koji sadrži T-SQL iskaz, zatim se vrši njegova predobrada, posle čega se kompajlira u plan izvršenja, koji se dalje šalje do skladišta podataka preko mašine za skladištenje koja vadi podatke. Izvađeni podaci se nakon toga ubacuju u tabelarni format (za TDS protokol) koji se šalje nazad do klijenta, preko mrežne biblioteke.

- **MAŠINA ZA SKLADIŠTENJE** Ove komponente su odgovorne za upravljanje datotekama baze podataka, kao i za upravljanje prostorom u okviru tih datoteka. Iako je deo DBMS sistema, ovo je u suštini centar arhitekture baze podataka SQL Servera, što je objašnjeno u drugom poglavlju. Mašina za skladištenje je odgovorna i za zapisivanje transakcija i za oporavak baze podataka.
- **ARHITEKTURA TRANSAKCIJA** Da biste mogli da shvatite šta čini SQL Server 2000 i kako da prilagodite različite osobine koje u njemu postoje, morate da shvatite arhitekturu transakcija. Postoje dve arhitekture transakcija sa kojima ćete se upoznati u ovoj knjizi. Najpre, tu je arhitektura lokalnih transakcija, koja je odgovorna za posao koji se odnosi na brisanje, ažuriranje, umetanje itd. Sa druge strane, tu je i arhitektura distribuisanih transakcija koja se odnosi na podatke iz spoljašnjih izvora. Ovo poslednje smo objasnili u dvadeset prvom poglavlju.
- **KONTROLA UPOREDNOSTI I UPRAVLJANJE TRANSAKCIJAMA** Ovo je deo arhitekture transakcija koji zaslužuje posebnu pažnju. To je sistem sa metodima i algoritmima koji omogućava da ga koristi više korisnika. Svim tim upravlja jedna instanca DBMS sistema. Uporednošću se kod SQL Servera upravlja preko arhitekture pesimističkog zaključavanja, koja je ugrađena u relacionu mašinu. Mehanizam zaključavanja omogućava da zaključate vrste, strane, nizove ključeva, indekse, tabele i cele baze. Pored onog što smo pomenuli ovde, zaključavanje je razmatrano i u dvadeset prvom poglavlju.

Relaciona mašina baze podataka

Relacionu mašinu možete da posmatrate na dva način. Možete da posmatrate njenu srž, tj. relaciono jezgro, ili možete da je posmatrate na višem nivou sa svim okolnim komponentama, od kojih jezgro prima svoje funkcije. Studija jezgra na nekom nižem nivou je izvan granica ove knjige. Ono čime ćemo se mi baviti može se uporediti sa parnom mašinom: para i vratilo su jezgro, a ostatak mašine, koji čine ložište, točkovi, dugmad, poluge... predstavlja viši nivo. Hajde da pogledamo jezgro sa svim njegovim funkcijama.

NAPOMENA

U samoj srži relacione mašine SQL Servera, u jezgru, isti je kod koji se koristi za svaku verziju SQL Servera, od CE izdanja (koje je ugrađeno u CE uređaje) do Enteprise izdanja na deljivim ili klasterovanim diskovima TerraServera. Ovo je razlog što možete da pišete kod za CE izdanje i da ga izvršite sa vrlo malim promenama na drugim verzijama SQL Servera 2000. ■

Na nivou jezgra, imate visoko optimizovan deo koda koja sakuplja T-SQL iskaze koji dolaze iz mrežne biblioteke i optimizuje ih u kod koji daje najefikasniji plan izvršenja. (Sa mnogih aspekata, relaciona mašina je kao kompajler koji kod prebacuje u objektni kod pre nego što ga prebaci u mašinski kod. Ovo se ne razlikuje od algoritama za predobradu koji postoje u različtim jezgrima, izvan sistema baza podataka.)

Memorijska arhitektura

Sofisticirana arhitektura upravljanja memorijom osigurava da SQL Server 2000 dobija svu memoriju koja mu je potrebna u okviru ograničenja koja nameće platforma. Upravljanje memorijom koje kod njega postoji je konzervativno. Mašina dobija potrebnu memoriju kada joj zatreba i oslobađa memoriju kada oslobađa resurse. Mogućnosti za napredno upravljanje memorijom koje postoje na Windows 2000 Serveru (Address Windowing Extensions ili AWE) omogućavaju da SQL Server Enterprise Edition radi sa adresnim prostorom od 8 GB na Windows 2000 Advanced Serveru i sa 64 Gb na Windos 2000 Data Centru.

Administratori baze podataka su i dalje u stanju da promene difolt podešavanja vezana za memoriju i da definišu fiksno zauzimanje memorije koje aplikacija ili okruženje zahtevaju (ovo je objašnjeno u dvanaestom poglavlju). Jedna od oblasti gde bi ovo bilo potrebno je upravljanje pulom memorije, koji ćemo ubrzo razmotriti. Ovaj memorijski pul je mesto gde SQL Server čuva svoj keš bafera za strane sa podacima.

SQL Server pokušava da održava virtuelnu memoriju na serveru između 4 MB i 10 MB, manje od fizičke memorije. Jedini način da ovo uradi je da menja veličinu svog memorijskog pula. Kada SQL Server učitava srodne procese, kao što su pozivi udaljenih uskladištenih procedura i OLE DB provajdera, ili kada se druge aplikacije ili druge instance SQL Servera pokreću, on će morati da promeni veličinu svog memorijskog pula da bi oslobodio potrebnu memoriju. Vi možete da podesite granice u kojima može da se menja memorijski pul tako što ćete podesiti vrednosti *min server memory* i *max server memory*, kojima se može pristupiti preko Enterprise Managera (vidi dvanaesto poglavlje). Svaka nova aplikacija koja se pokreće na serveru je umesto zauzete ostraničene memorije. Svaka instanca SQL Servera 2000 je prema tome u stanju da radi unutar memorije sistema, koristeći druge instance. (Kada se koristi AWE, međutim, sistem će zahtevati dodeljivanje statičke memorijske adrese.)

Memorijska arhitektura Windowsa sadrži memorijski model koji je zasnovan na ravnom, linearnom, 32-bitnom adresnom prostoru na Windowsu NT, Windows 2000 Serveru i Windows 2000 Advanced Serveru, kao i 64-bitnom adresnom prostoru za Windows Data Centru. Kod operativnih sistema Windows NT i Windows 2000 koriste se dva tipa memorije. To su *fizička* i *virtuelna* memorija. Fizička memorija je memorija u RAM čipovima koja je instalirana na sistemskoj matičnoj ploči. Nakon toga, imate virtuelnu memoriju, koja predstavlja kombinaciju celokupne memorije na sistemu i arhitekture koja omogućava da je operativni sistem koristi. Jedan važan aspekt virtuelne memorije je da ona može da uključi i memoriju hard diska.

NAPOMENA

U šestom poglavlju smo razmotrili različite konfiguracije memorije. ■

Datoteke stranica i VMM

Menadžer virtualne memorije (VMM) upravlja svom sistemskom memorijom i osigurava da aplikacije i operativni sistem imaju pristup do više memorije nego što obezbeđuju RAM čipovi koji se nalaze na matičnoj ploči. Memorijski čipovi su brži nego virtuelna memorija, što je činjenica koja je bitna za SQL Server.

VMM ima još jednu važnu ulogu. On garantuje da memorijske adrese koje se koriste od strane više instanci i aplikacija neće prelaziti u tuđi atar. U starim danima DOS-a, ovaj problem je dovodio do katastrofe. Upravljanje memorijom postoji godinama, još od prve verzije Windows operativnog sistema, pa sve do Windows 2000 DataCentra, koji je jedan od najsofisticiranijih operativnih sistema koji postoje. Sa svakom novom verzijom operativnog sistema poboljšava se upravljanje virtuelnom memorijom, tako da se algoritmi koji postoje na Windowsu 95 razlikuju od algoritama koji se koriste na Windowsu 98 itd. Upravljanje memorijom kod Windowsa NT je takođe značajno poboljšano kod verzije Windows 2000. To znači da DBMS mora da koristi različit algoritam keširanja da bi SQL Server 2000 mogao da podrži instance na svim 32-bitnim ili 64-bitnim operativnim sistemima (izuzev za Windows 95, na kojem SQL Server 2000 i ne može da radi).

VMM ima dve osnovne funkcije:

- VMM upravlja memorijskom tabelom, koja prati listu svih virtuelnih adresa koje su dodeljivane svakom procesu. Analogija bi mogla biti hotel kod kojeg su svi gosti povezani sa sobama, tako da recepcioner može da pogleda u tabelu da bi mogao da ih opslužuje. VMM se u sistemu nalazi tamo gde se nalaze podaci prebačeni u adrese. Drugim rečima, on se ponaša kao servis za translaciju, koji povezuje virtuelnu i fizički memoriju. Ova funkcija je transparentna za aplikacije koje rade nesvesne činjenice da postoji proces koji im daje iluziju da imaju pristup do fizičke memorije.
- Kada proces koristi više od raspoložive RAM memorije, VMM prebacuje sadržaj memorije na disk. Ovo se naziva straničenje.

Windows 2000 u osnovi ima pristup do adresnog prostora od 4 GB, iako je prostor virtuelan i može ga činiti RAM zajedno sa prostorom na disku. Ako se vratimo našoj analogiji sa hotelom, ovo izgleda kao da imate neke značajne goste koji se nalaze u glavnoj zgradi hotela sa pogledom na more, dok se manje važni gosti nalaze u sobama sa pogledom na ulicu. Ukupan broj soba je, međutim, jednak sumi svih soba u obe zgrade.

Iako govorimo o adresnom prostoru od 4 GB, ovaj prostor je relativan, s obzirom na to kako sistem koristi memoriju. Konkretno, na Windows 2000 Serveru adresni prostor koji aplikacije mogu da koriste je samo 2 GB, pa čak i manje od toga, pošto se 2GB dele između svih procesa koji rade u korisničkom režimu. Preostala 2GB su rezervisana za izvršenje u režimu jezgra.

Prema tome, ako pogledate memorijsku arhitekturu Windowsa NT i Windowsa 2000, videćete da su 4GB prostora podeljena u gornji i donji segment, oba sa po 2GB. Niti i procesi u režimu jezgra zauzimaju memoriju preko gornjeg prostora, a donji prostor je rezervisan i za korisnički režim i za režim jezgra. Gornji deo se takođe rezerviše za izvesne niže regione ukupnog adresnog prostora, koji se koriste za hardver.

Donji deo se pulovima za straničenje održava preko strana od 4KB. Postoje pulovi koji nisu straničeni i oni koji jesu. Strane u straničenom pulu su one koje se mogu prebaciti na disk. Obično su rezervisane za aplikacije, odnosno procese koji se mogu naći u stanju mirovanja. Pulovi bez straničenja ostaju u fizičkom RAM-u i podržavaju funkcije operativnog sistema i kritične servise. Veličina svake strane je 4KB.

Još malo o straničenju

Straničenje je proces premeštanja podataka u fizičku memoriju i iz nje, kao što je RAM memorija. Kada se memorijski pul, koji se zasniva na RAM memoriji, napuni i kada se od operativnog sistema zatraži da isporuči još memorije, VMM iz memorije izbacuje podatke koji se nalaze u stanju mirovanja i postavlja ih na disk, u strukture koje se nazivaju stranicama podataka.

VMM upravlja memorijskim pulom preko dodeljivanja prostora stranama koje su identifikovane kao *ispravne* ili kao *neispravne*. Ispravne strane su locirane u fizičkoj memoriji i njima može da se pristupi istog trenutka kada ih aplikacija zatreba. Za ove stranice možete da smatrate da su online za tu aplikaciju, kao linija kredita. Neispravne strane su offline za aplikaciju i ne može im se odmah pristupiti. Ove strane se nalaze na disku.

Iako se neaktivne aplikacije i podaci čuvaju u offline stranama, VMM ima pokazivače na adrese, tako da može da obnovi podatke kada se za time javi potreba. Na primer, kada se referencira neka aplikacija ili podatak sa strane, operativni sistem treba da ih ponovo prebaci u RAM memoriju, što je brže moguće. U tom trenutku se pokreće nivoa nit koja pristupa podacima sa strane. Da bi podaci sa strane mogli da se ubace u memoriju, neaktivni podaci moraju da se pošalju na disk. Sada vidite zašto se preporučuju brzi i pouzdani hard diskovi, posebno u slučaju aplikacija koje puno rade sa podacima i traže dosta memorije.

VMM kao deo rutina za straničenje obavlja puno kućnih poslova:

- Upravlja podacima sa offline strana i to na bazi prvi-unutra, prvi-napolje. Podaci koji se najduže nalaze na disku su prvi kandidati za vraćanje u RAM memoriju kada se ona oslobodi, osim u slučaju da se neki podatak eksplicitno ne zatraži. Drugim rečima, VMM ne poziva jednostavno podatke koji su bili neaktivni na disku da postanu neaktivni u RAM memoriji. VMM nastavlja da podatke ubacuje i izbacuje iz RAM memorije dokle god postoji potreba za smeštanjem podataka na disk. "Vrući podaci" na koje VMM stalno motri nazivaju se *radnim skupom*.
- Kada podatke vraća sa strane, VMM obavlja operaciju poznatu pod imenom donošenje. Pored toga, VMM obavlja i nešto što je poznato pod terminom klasterovanje strana sa podacima. Klasterovanje strana sa podacima znači da kada VMM donese podatke, ona donosi i podatke koji se nalaze u blizini te strane, što je zasnovano na pretpostavci da ćei oni ubrzo biti potrebni. Ova tehnika, prema tome, pokušava da poveća brzinu ulaza i izlaza sa strane sa podacima.
- VMM je dovoljno pametan da zaključi da ukoliko u RAM memoriji nema dovoljno mesta za smeštanje podataka sa diska, onda poslednji podaci iz RAM-a moraju da budu zamenjeni i smešteni na disk.

Parametre sa kojima VMM radi možete da menjate i kontrolišete. Možete i da menjate veličinu datoteke sa stranicama. Ovo je vrlo bitno i treba da imate na umu.

Address Windowing Extensions

Podrška vrlo velikog adresnog prostora, kao što su 8GB na Windows 2000 Advanced Serveru ili 64GB na Datacenteru, moguća je samo preko Address Windowing Extensions (AWE). AWE je za Windows 2000 ono što su DOS extenderi i EMS biblioteke za stare Windows operativne sisteme

i ranije 16-bitne verzije Windowsa (setite se Pharlapa). AWE je uključen u 32-bitni i 64-bitni API i omogućava aplikacijama da adresiraju mnogo više od 4GB, koliko se može koristiti preko standardnog 32-bitnog API-ja.

AWE omogućava aplikacijama da kao fizičku memoriju zauzmu više memorije koja nije straničena. Nakon toga, on dinamički povezuje poglede na nestraničenu memoriju sa 32-bitnim adresnim prostorom. Ovo bi se moglo nazvati *združenim adresiranjem memorije*. Istraživanje AWE-a je izvan onoga čime se ova knjiga bavi, pošto vi kao DBA ili programer aplikacija možete da ostavite da Windows 2000 i SQL Server upravljaju dodatnim adresiranjem za vas. Sve što treba da uradite je da omogućite upotrebu AWE-a.

NAPOMENA

Više o AWE-u možete naći u MSDN Library. ■

AWE se za SQL Server 2000 aplikacije uključuje preko sistemske uskladištene procedure sp_configure, koja može da *uključi* ili *isključi* AWE.

Adresni prostor SQL Servera

Svaka instanca SQL Servera koja je pokrenuta na računaru zauzima izvestan adresni prostor koji se deli na određeni broj oblasti koje sadrže različite module sa kodom koji čini DBMS. Svaki adresni prostor, za svaku instancu, sadrži sledeće komponente:

- **IZVRŠNI KOD** Ova oblast adresnog prostora sadrži Open Data Services (ODS), kod SQL Server mašine, kao i SQL Server mrežne biblioteke. Pored toga, distribuisani upiti mogu da nezavisno učitaju DLL biblioteku OLE DB provajdera u svoj adresni prostor. Ovo važi i za proširene uskladištene procedure i OLE Automation objekte.
- **MEMORIJSKI PUL** Ova oblast adresnog prostora se koristi za strukture podataka koje koriste memoriju u instanci SQL Servera. Tipični objekti koji koriste memorijski pul su:
  - **STRUKTURE PODATAKA SISTEMSKOG NIVOA** Ovo su strukture koje sadrže globalne podatke za isntancu, kao što je opis baze podataka i tabela zaključavanja.
  - **KEŠ BAFERA** Ovo je keš koji sadrži baferovane strane u kojima se strane sa podacima čitaju (ovo ćemo ubrzo razjasniti).
  - **KEŠ PROCEDURE** Ovo je keš koji sadrži plan izvršenja za T-SQL upite koji su bili izvršeni ili su u procesu izvršenja, sve to u okviru instance. (Vidi odeljak pod naslovom "SQL, optimizacija upita i obrada upita" kasnije u ovom poglavlju.)
  - **KEŠ DNEVNIKA** Ovo je keš baferovanih strana koji se koristi za čitanje i pisanje strana dnevnika. Kao što smo objasnili u drugom poglavlju, baferima dnevnika i pridruženim datotekama dnevnika upravlja se posebno u odnosu na bafere podataka i aktuelne datoteke sa podacima.

- **KONTEKST VEZE** Ovaj kontekst sadrži podatke koji predstavljaju svaku vezu u toj instanci. Podaci se odnose na informacije o stanju, vrednosti parametara za upite i uskladištene procedure, pozicije kursora i tabele koje se referenciraju.

Memorijski pul je promenljiva struktura. SQL Server konstantno prilagođava količinu memorije koja se zauzima za različite komponente, a sve u cilju optimizacije performansi. Svaki put kada se klijent poveže, SQL Server mora da zauzme memoriju za novu vezu. Kada klijent prekida vezu, ta memorija treba da se oslobodi. Nove i promenjene baze podataka i svi njihovi objekti konstantno menjaju zauzimanje i upotrebu memorije. Keš bafera, keš procedura, kao i keš dnevnika takođe su faktori koji utiču na to kako se memorija koristi i oslobađa.

SAVET

Aplikacije koje kreirate i način pisanja koda mogu imati direktan uticaj na performanse SQL Servera. Broj odlazaka na server, kao i količinu podataka koji se šalju, morate da smanjite što je više moguće. Ovo ću detaljnije objasniti u četrnaestom i devetnaestom poglavlju. ■

SQL Server, kao i svi DBMS sistemi, teži da minimizira upotrebu diska tako što koristi usluge keša bafera u memoriji. Ovaj keš bafera sadrži strane koje su pročitane iz baze podataka. Članovi razvojnog timaa SQL Servera proveli su mnogo vremena da bi bili sigurni da SQL Server 2000 maksimalno koristi keš bafera i da mimizira ulazno/izlazne operacije iz datoteka baze podataka. Kada bafer istera memoriju, operativni sistem počinje sa promenom strana u datoteci sa stranama, preko procesa koje smo ranije opisali. Prirodno je da će sistem biti sporiji, pa je za sisteme čija je misija bitna dobro dodati više realne memorije kada VMM počne mnogo da se koristi. Što je veći keš bafera, to je bolje vreme odgovora SQL Servera.

Ulazno/izlazna arhitektura SQL Servera

Ulazno/izlazna arhitektura, koja definiše kako se podaci čitaju i kako se zapisuju u bazu podataka, jeste arhitektura koja pomaže sve sisteme za upravljanje bazama podataka. Ona je jedna od ključnih komponenti ovih proizvoda i jedna od ključnih arhitektura koja odvaja jedan sistem od drugog. Sistem koji ima najbržu ulazno /izlaznu arhitekturu, a uz to i sa najviše inovacija, pobeđuje u ratu koji se odnosi na učitavanje transakcija. Ovo smo već objasnili u delu koji se odnosio na TPC repere.

SQL Server 2000 ima pred sobom velike ciljeve. Sa jedne strane mora da zadovolji potrebe malih preduzeća, dok sa druge mora da se takmiči sa celim svetom. Verujem da je Microsoft odgovorio izazovu, pošto SQL Server može da se primenjuje i za jednokorisnička rešenja, ali i za podršku hiljade korisnika preko Interneta. Hajde da izbliza pogledamo ulazno/izlaznu arhitekturu koja omogućava ovakav podvig.

Postoje dve lokacije sa kojih instanca SQL Servera može da čita (i zapisuje) podatke: iz memorije i sa fizičkog hard diska. Ne treba mnogo truda da se zamisli šta bi postalo od hard diska ukoliko bi se svi podaci konstantno čitali iz baze. Niz uzastopnih transakcija bi mogao da izazove i požar na serveru (u literalnom smislu). SQL Server, kao i svi DBMS sistemi, u memoriji čuva virtuelnu kopiju podataka kojima se naviše pristupa, kao i podataka koji su poslednji korišćeni. Drugim rečima, pisanje i čitanje ne zahteva fizičko premeštanje podataka.

Kada se SQL Server pokrene i kada se zatraže njegovi podaci, oni se učitavaju u virtuelnu memoriju i zauzimaju prostor u tzv. kešu bafera, koji je za njih rezevisan. Kao što ste naučili u drugom poglavlju, podaci se čuvaju na stranama od 8KB. Strane su kontinualne do veličine od 64KB. Prema tome, virtuelne strane sa podacima u kešu bafera su takođe veličine 8KB.

Sada se virtuelno čitanje (ponekad se koristi i termin logičko čitanje) dešava svaki put kada klijent zatraži podatke koji se nalaze u kešu bafera. Ako podaci koji se traže nisu u kešu bafera, onda SQL Server mora da ih dobije sa hard diska, odnosno mora da izvrši *fizičko čitanje*. Nasuprot tome, *virtuelno zapisivanje* se javlja kada se podaci u memoriji menjaju ili ažuriraju. Pod *virtuelnim zapisivanjem* se podrazumeva da su podaci ažurirani u kešu, a ne na disku. Podaci, međutim, nisu perzistentni i ako struja nestane, oni se gube. S vremena na vreme podaci moraju biti *fizički zapisani* na disku. Ipak, performanse imaju prednost nad perzistentnošću, tako da DBMS sistemi imaju nekoliko virtuelnih zapisivanja, pre nego što se podaci i fizički zapišu na disku.

DBMS sistem prema tome mora da održava odgovarajući bafer, tako da i performanse budu na nivou i da server ne mora intenzivno da piše i čita podatke sa diska. Sa druge strane, veliki keš bafera ostavlja manje resursa za ostale aplikacije. To može rezultuje tako da server štedi na fizičkom čitanju i pisanju, ali da sa druge strane gubi na stalnom straničenju koje se dešava da bi se podržale druge aplikacije.

SQL Server daje tzv. ulazno/izlaznu envelopu tako što koristi ulazno/izlazne osobine Windowsa NT. To su *Scatter-Gather* ulaz/izlaz i *Asynchronous* ulaz/izlaz. Scatter-Gather ulaz/izlaz se često navodi kao nova osobina Windowsa 2000, ali se on pojavio pre nekoliko godina sa verzijom Windows NT 4.0 SP2. Nema sumnje da je korišćen za proširenja kod Windowsa 2000, tako da je sada ugrađen u operativni sistem. U suštini, ova tehnologija je odgovorna za premeštanje podataka iz nepovezanih oblasti RAM-a do kontinualnih blokova podataka na disku. Iako vi, kao DBA ili administrator sistema, nemate neku kontrolu nad time kako operativni sistem koristi ovu tehnologiju, sistemi i softver moraju biti specijalno napisani tako da mogu da koriste prednosti koje ovo donosi. SQL Server 2000 je takav sistem, odnosno Scatter-Gather je i kreiran za SQL Server.

NAPOMENA

Ovo nije potpuno ista stvar koju radi VMM, pošto se prebacivanje podataka preko VMM vrši bez znanja aplikacije. ■

Podizanje performansi SQL Servera je očiglednije kada DBMS sistem čita podatke sa diska. Na primer, ako SQL Server čita u opsegu od 64 KB, on ne mora da u kešu bafera rezerviše oblast od 64 KB za te podatke. Može da prebaci osam baferovanih strana direktno u keš bafera, a da kasnije skupi osam strana time što će održati adrese u kešu za osam strana.

Druga ulazno/izlazna osobina SQL Servera je Asinhroni ulaz/izlaz. Ako su ulaz/izlaz sinhroni, svaki put kada SQL Server odvaja nit za čitanje ili zapisivanje podataka, ceo sistem bi morao da čeka da se ta operacija završi pre nego što ponovo dobije kontrolu da bude u stanju da se prebaci na sledeći proces. Kod Asinhronog ulaza/izlaza, SQL Server može da nastavi da radi na drugim zadacima, a da periodično proverava status ulazno/izlazne niti.

Preko asinhronih ulazno/izlaznih operacija, SQL Server može da obavlja više uporednih ulazno/izlaznih operacija. Vi možete da regulišete maksimalni broj ulazno/izlaznih operacija tako što ćete podesiti opciju *max async*. Podrazumevana vrednost je 32 niti za ulaz/izlaz. Samo najzahtevnije aplikacije mogu da traže da podesite neku veću vrednost. Ovo je sada dobar trenutak da pogledamo kako izgleda arhitektura niti i zadataka SQL Servera.

Arhitektura niti i zadataka SQL Servera

Svi DBMS sistemi, kao i sve glavne aplikacije, zasnivaju se na sistemima sa više niti. Mi se nećemo ovde detaljno baviti tehnologijom niti, ali ćemo reći da ona omogućava da složeni procesi efikasnije koriste procesor. Ako ste novi u softverskom inženjeringu ili prvi put čujete za niti, postoji mnogo knjiga koje se bave ovim problemom. Ovo je dobro objašnjeno i u MSDN Library u Microsoftovoj onilne bazi znanja.

U ne tako davnoj prošlosti, jedini način za uporedno izvršavanje procesa u sistemu bio je da se pokrenu višestruke instance aplikacije. Multitasking jezgro (OS) je bilo odgovorno za odlučivanje o tome koje aplikacije treba da koriste procesor i kada to treba da rade. Ovo je i način na koji je Windows u početku radio. Kasnije je za neku aplikaciju postalo moguće da dobije određeni nivo uporednog pristupa procesoru i to unutar adresnog prostora same aplikacije. Ovo je bilo moguće preko niti sa procesima. Više niti vodi do pouzdanijeg softvera, pošto operativni sistem sada može da na pravilan način upravlja redosledom i prioritetom izvršenja niti.

Bitno je da shvatite da niti ne možete na jednom procesoru izvršavati u isto vreme, pošto su one sinhrona tehnologija. Operativni sistem može, međutim, da odredi kada treba da prekine neku nit ili da dozvoli da se neka nit izvrši pre drugih (prioritet). Kod sistema sa više procesora niti mogu da se izvršavaju u potpuno uporednom stanju, tako što se izvršavaju na paralelnim procesorima. Drugim rečima, ako računar ima osam procesa, on može istovremeno da izvršava osam niti.

Sa nitima su povezana vlakna. Vlakna su slična sa nitima, ali ih kontroliše sama aplikacija, a ne operativni sistem. Tehnologija vlakana ne zahteva troškove operativnog sistema, koji se javljaju kod niti. U pitanju su resursi operativnog sistema, koji se koriste kako u korisničkom tako i u režimu jezgra. Vlakna nisu potpuno odvojena od niti. Ona su izvedena iz njih. Nit može da obuhvati više vlakana.

Izvršenje vlakana je napredna tehnologija i samo Enterprise Edition SQL Servera 2000 može da je podrži. Kod SQL Servera se u isto vreme izvršava mnogo različitih aktivnosti i svi ti procesi zauzimaju vreme procesora. U jezgru DBMS sistema postoji funkcija koja radi sličnu stvar kao i operativni sistem koji radi sa nitima. Jezgro SQL Servera, prema tome, omogućava da sistem radi i sa nitima i sa vlaknima, a da se pri tome ne zauzimaju jezgro ili resursi korisničkog režima operativnog sistema. Svaka instanca SQL Servera ima svoj pul niti i vlakana, koji se izvršavaju u ovom nivou vlakana i niti.

Ovo je samo jedna od osobina SQL Servera koja mu omogućava da bude toliko moćan. Konfiguraciju pulova kontrolišete preko opcije *max worker threads* (vidi dvanaesto poglavlje). Kod verzije Enterprise Edition SQL Servera možete i da definišete da li će se koristiti samo niti ili i niti i vlakna. Ako izaberete i niti i vlakna, server se postavlja u tzv. *režim vlakna*.

U tom režimu, SQL Server uspostavlja jednu nit po procesoru, a nakon toga daje jedno vlakno po uporednoj vezi klijenta. Broj vlakana se određuje na osnovu opcije *max worker threads* (difolt vrednost je 255).

Kada SQL Server, preko žice, primi T-SQL iskaz, on uzima slobodnu nit ili vlakno (u zavisnosti od režima rada) sa steka. Ako nema slobodnih niti ili vlakana i ako nije dostignuta vrednost max worker threads, onda SQL Server uzima novu nit ili vlakno. Ako je dostignuta maksimalna vrednost niti, onda SQL Server mora da pronađe neku nit koja se ne koristi i da je oslobodi.

Iako možete doći u iskušenje da opciju *max worker threads* podesite na veću vrednost od difolt vrednosti, ipak treba da budete svesni da postoji trenutak kada performanse SQL Server počinju da se smanjuju. Razlog je jednostavan. Upravljanje nitima i vlaknima samo po sebi traži resurse, a scenario sa ovakvim nitima može dovesti do zastoja. U nekom trenutku većina niti koje su kreirane ne rade ništa, tako da se slobodno mogu osloboditi i ponovo upotrebiti bez ponovnog zauzimanja resursa, tako da vrednost *max worker thread* nikad ne treba da podešavate na vrednost koja bi bila veća od 255.

Arhitektura transakcija SQL Servera

Svaki DBMS sistem mora da bude dobar za online obradu transakcija (OLTP). To znači da se preko transakcija, koje se u bazi podataka dešavaju kad god se podaci čitaju, zapisuju ili ažuriraju, mora obezbediti integritet podataka.

Transakcija je kompletna jedinica posla, koja traje od početka upita do njegovog kraja. Transakcija se ne može prekinuti, suspendovati ili ostaviti za neko drugo vreme a da se pri tome ne pojavi opasnost po integritet podataka, pošto komanda ne može da bude ekskluzivni servis celog DBMS sistema. Ostale transakcije takođe treba da se izvršavaju (istovremeno), što i jeste cilj DBMS ili OLTP sistema. Transakcija ne može da bude vlasnik DBMS sistema i da isključivo zadržava neki posao sve dok se on ne završi u nekoj neodređenoj budućnosti.

Integritet se osigurava preko filozofije "sve ili ništa" (atomičnost). Drugim rečima, transakcija se mora završiti bez grešaka ili se ne može uopšte izvršiti. Tipična transakcija se može predstaviti sledećim kodom:

```
BEGIN TRANSACTION
      INSERT INT CustDetails (CustId) VALUES ('15')
COMMIT TRANSACTION
```

Tabela baze podataka je u konzistentnom ili verodostojnom stanju na početku transakcije. Klijent signalizira početak transakcije. Ovo se može uraditi eksplicitno ako se cela transakcija stavi između iskaza BEGIN TRANSCTION i COMMIT TRANSACTION. Transakcija se može pokrenuti i implicitno, bez iskaza BEGIN .. COMMIT, tako što se pošalje T-SQL upit, koji je postavljen u režimu autocommit.

Za vreme transakcije, DBMS sistem obavlja sve korake koji su potrebni za izvršenje upita, a u cilju održavanja integriteta podataka koje transakcija kontrološe. U ovom trenutku je transakcija vlasnik podataka sa kojima trenutno radi, ali u nekom trenutku ti podaci mogu biti u nekonzistentnom stanju. Ako se u jednoj transakciji koristi više tabela, prirodno je da sve one ne mogu biti ažurirane u istom trenutku. Za vreme trajanja transakcije, neke vrste mogu biti ažurirane a neke ne.

Ako se za vreme transakcije javi neka greška, tabele koje još uvek nisu bile uzete u obzir od strane transakcije biće izvan petlje. U slučaju greške, potrebno je pozvati iskaz ROLLBACK TRANSACTION da bi se obnovili svi podaci koji su promenjeni u transakciji. Ako nema grešaka, odnosno ako je transakcija uspešno završena, izvršava se iskaz COMMIT TRANSACTION. Nakon potvrđivanja, sve modifikacije postaju deo baze podataka.

Kao što smo objasnili u drugom poglavlju, dnevnik transakcija je taj koji nam daje mogućnost da poništimo promene baze podataka, odnosno da je vratimo u prethodno stanje. Ovo važi čak i kada je transakcija bila uspešna. U dnevniku transakcija zapisuju se sekvence događaja koji se dešavaju od početka do kraja transakcije. U dnevniku se nalazi dovoljno informacija da transakcija može da se poništi ili da se ponovo izvrši. Oporavak i posao koji je vezan za dnevnike transakcija potpuno je objašnjen u devetom i dvadeset prvom poglavlju.

Arhitektura izolacije, uporednosti i zaključavanja

Upravljanje izolacijom i uporednošću deo je upravljanja transakcijama. Svi DBMS sistemi imaju sistem za izolaciju i uporednost. Ovo se koristi da bi se osiguralo da višestruke transakcije, koje predstavljaju najmanje dva, a možda i više hiljada korisnika, i koje u istom trenutku rade u istoj bazi podataka, mogu da pristupaju istim podacima skoro u isto vreme, ne utičući jedne ne druge. Algoritam koji se odnosi na izolaciju, a koji je implementiran kod SQL Servera, obezbeđuje da transakcije koje su započete od strane uporednih sesija ne utiču jedna na drugu.

U osnovi metoda za osiguranje uporednosti nalaze se dve specifikacije. To su pesimistička i optimistička kontrola uporednosti. Evo njihovih definicija:

- *Pesimistička kontrola uporednosti* se sastoji od sistema mehanizama zaključavanja koji sprečavaju da više aplikacija u istom trenutku pristupi istom podatku. Razlog što se ovo zove *pesimistička kontrola uporednosti* je u tome da se on primenjuje u situaciji kada postoji veliki broj transakcija koje pripadaju velikom broju veza, koje će verovatno napasti iste podatke u nekom nekontrolisanom stanju. Većina aplikacija sa transakcijama spada u ovu kategoriju. Potrošačke korpe na Webu su dobar primer za okruženje sa mnogo transakcija, kod kojeg srednji nivo mora da otvori veliki broj veza sa istom bazom podataka i da pristupa istim vrstama za sve te veze.
- *Optimističa kontrola uporednosti* obuhvata situacije u kojima veze ili klijentske aplikacije ne zaključavaju podatke kojima pristupaju. Sistem jednostavno proverava podatke, i ako dođe do neke svađe, neko mora da povuče svoju transakciju i da je počne iz početka. Optimistička kontrola je moguća kod visokokontrolisanih okruženja sa malim saobraćajem.

Vaše aplikacije i rešenja će morati da uzmu u obzir cenu zaključavanja objekata baze podataka za vreme transakcija. Mnoge aplikacije sa velikim ulogom, ili one čija je misija kritična, mogu da se kontrolišu preko jednog sistema za terminiranje koji je u potpunosti svestan svih niti za transakcije sa bazom podataka koje je on uspostavio, kao što je, na primer, PBX sistem. Zaključavanje i otključavanje traži dosta resursa, a i milisekunda koja se doda može da dovede do ozbiljnih problema za tu aplikaciju. Sistem za terminiranje mora da bude svestan načina i trenutka dolaska do konflikta, nakon čega sa njime radi preko algoritma za optimističku kontrolu uporednosti.

Sa druge strane, većina DBMS okruženja za obradu transakcija koriste mehanizam zaključavanja, posebno kada se sve sesije i veze uspostavljaju od strane nezavisnih aplikacija. Najčešće, veze uspostavljaju ljudi koji nemaju pojma da još neko radi sa tom istom bazom podataka.

Brave i reze

Implementacija pesimističke kontrole uporednosti je kod SQL Servera moguća preko sofisticirane arhitekture zaključavanja. Zaključavanje je najbitnije kod sistema sa desetinama hiljada korisnika koji rade sa istim skupom podataka. Zaključavanje je način za osiguranje integriteta podataka. U pitanju je sistem osiguranja koji omogućava procesu da sigurno radi sa podacima i da podaci ne budu promenjeni negde "iza scene".

Zaključavanjem se upravlja na bazi veza. Drugim rečima, veza AA koja zahteva zaključavanje ne može da koristi zaključavanje nastalo od veze AB, čak i ako su obe veze inicirane od iste klijentske aplikacije. Svaka veza mora da ima zaključavanje kreirano isključivo u kontekstu nje same. Do izuzetaka može doći kada se koriste vezane veze, ali to ću ostaviti za dvadeset prvo poglavlje.

SQL Server podržava nekoliko tipova zaključavanja. To su: *deljivo*, *sa ažuriranjem*, *isključivo*, *sa namerom* i *zaključavanje šeme*. Načini zaključavanja ukazuju na nivo zavisnosti koji veza ima sa zaključanim objektom. Vi, na primer, ne možete da dobijete isključivo zaključan objekat ako je na njemu već primenjeno deljivo zaključavanje.

Niti ili vlakna SQL Servera postavljaju zaključavanje na bazama podataka, tabelama, vrstama, indeksima, ključevima, opsezima ključeva i stranama. Ovaj prilaz je poznat pod imenom *granularnost zaključavanja*. Granularnost zaključavanja se dinamički određuje od strane SQL Servera (za vreme formulisanja plana upita, koje je objašnjeno kasnije) i ne traži nikakvu konkretnu akciju od vas ili aplikacije. Ovo ne sprečava aplikaciju da zatraži konkretan način zaključavanja. SQL Server određuje koji je nivo zaključavanja potreban za svaki T-SQL iskaz koji vraća. Jedan upit može dovesti do kreiranja zaključavanja vrste, dok drugi može generisati zaključavanje koje se odnosi na celu bazu podataka. Veze uzimaju u obzir zaključavanje, bez obzira na to o kom se nivou radi. Drugim rečima, ako veza AA ima definisano zaključavanje tabele, veza AB ne može da uspostavi zaključavanje vrsta u istoj tabeli.

S vremena na vreme, SQL Server može da odluči da intenzivira zaključavanje. Na primer, ako zaključavanje na nivou vrste zauzima veći deo tabele, relaciona mašina može da postavi zaključavanje za celu tabelu. Procesor za obradu upita obično određuje ispravno zaključavanje potrebno za određeni upit.

Zaključavanje se održava isključivo onoliko vremena koliko je potrebno da se zaštite podaci, na nivou koji klijent zahteva. Na primer, deljivo zaključavanje SQL Server zadržava za vreme koje zavisi od nivoa izolacije transakcija. Difolt nivo izolacije transakcija je READ COMMITED, što odgovara deljivom zaključavanju koje traje onoliko koliko traje čitanje podataka. Skeniranje takođe dovodi do zaključavanja, ali je trajanje mnogo kraće, obično onoliko dugo koliko je potrebno za skeniranje strane i postavljanje zaključavanja na narednu stranu. Ostala zaključavanja traju koliko traje i transakcija, kao kada je nivo izolacije transakcija podešen na REPEATABLE READ ili SERIALIZABLE READ (ovo su čista čitanja, koja je zahtevaju da nema preklapanja sa drugim procesima).

Kursori su takođe zaštićeni od uporednih operacija. Kursor može da ima zaključavanje sa pregledom u deljivom režimu da bi se zaštitilo uzimanje podataka. Zaključavanje sa pregledom se obično zadržava do kraja transakcije. Ažuriranje takođe zahteva zaključavanje podataka, i to za vreme koje je potrebno za izvršenje transakcije.

Kao i svi moderni DBMS sistemi, i SQL Server ima mehanizme za sprečavanje konflikta prilikom zaključavanja. Ako veza pokuša da dobije zaključavanje nad podacima koji su već zaključani, poslednja veza se blokira i čeka dok prva veza ne završi transakciju. Vi ćete morati da napišete sopstvenu funkciju za upravljanje zaključavanjem i da to stavite u klijentsku aplikaciju ili uskladištenu proceduru ako želite da odlučujete koliko dugo nit aplikacije treba da čeka na mogućnost da zaključa podatke. Aplikacija dobija zaključavanja na principu "ko je prvi došao, prvi se i uslužuje".

SQL Server podržava i detekciju zastoja. Postoji algoritam koji se primenjuje za istraživanje i osiguravanje da niti nisu zamrznute, što se dešava kada su dve ili više veza blokirale jedna drugu u tzv. *zagrljaju smrti.* Ako SQL Server otkrije takvu situaciju, on će završiti neku od niti transakcije, čime će se omogućiti da preostale niti nastave rad (vidi tekst "Zastoji" u dvanaestom poglavlju).

Reze i mašina za skladištenje

Mašina za skladištenje upravlja drugim mehanizmom za uporednost, zvanim reze. Reza se dešava kada relaciona mašina zatraži od mašine za skladištenje da vrati vrstu za vreme obrade upita. Mašina za skladištenje postavlja rezu na stranu sa podacima da bi osigurala da je neki drugi proces ne promeni za vreme transakcije. Na primer, reza može da osigura da se strana sa stavkama u tabeli ne promeni sve dok se transakcija ne završi.

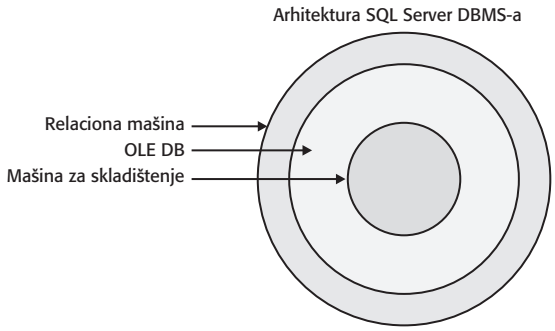
Čitanje i pisanje na stranama sa podacima

Kod prethodnih verzija SQL Servera, odnosno kod verzije 6.5 i prethodnih, kao i kod drugih starijih DBMS proizvoda, funkcija skladištenja i relaciona funkcija su bile u istoj jedinici. Danas su ove dve funkcije odvojene u dve zasebne jedinice. To su mašina za skladištenje i relaciona mašina. Obe mašine imaju određeni broj podmodula, koji su odgovorni za čitanje i zapisivanje podataka. U većini slučajeva ne morate da brinete o osnovnoj arhitekturi, ali vam znanje o tome kako SQL Server pristupa podacima može pomoći da odlučite da li treba da poboljšate performanse sa dodatnim indeksima ili da ponovo napišete upite i sl.

Uloga koju ima mašina za skladištenje opisana je u drugom poglavlju. Ona blisko sarađuje sa relacionom mašinom preko OLE DB mosta, što znači da postoji interfejs od dve komponente. Ovo je ilustrovano na slici 4.1. Svi zahtevi za čitanjem se generišu od strane relacione mašine, što znači da se za čitanje podataka iz tabela koriste specijalizovani algoritmi pristupka (u okviru objekata baze podataka), sa ili bez indeksa.

Metodi pristupa nisu deo standarda, kao što je SQL. Jezik je suviše apstraktan da bi se za to koristio (premda ću ja razmotriti jezik za definisanje podataka u jedanaestom poglavlju, kao i sintaksu za kreiranje i rad sa indeksima). Metodi pristupa su odgovarajući za svaki proizvod, odnosno svaki proizvođač je slobodan da doda nešto posebno.

Relaciona mašina određuje metode pristupa koji su joj potrebni za dobijanje podataka. Ovi metodi mogu da obuhvate lokalno skeniranje tabela, skeniranje udaljenih tabela, skeniranje stranih tabela, čitanje datoteka, skeniranje indeksa, čitanje sa ključevima itd. Kombinacija ovih metoda definiše generalni obrazac za čitanje koji se primenjuje na SQL Serveru. SQL Server može da omogući upotrebu različitih metoda pristupa, uključujući i Indexed Sequential Access Method (ISAM), koji se koriste kod desktop sistema baza podataka, kao što su Microsoft Access ili Microsoft FoxPro. Arhitektura baze podataka koja je objašnjena u drugom poglavlju omogućava čitanje i pisanje podataka na način koji je najbrži i najefikasniji.



SLIKA 4.1 *Relaciona mašina i mašina za skladištenje su posebne komponente koje komuniciraju preko OLE DB-a*

Kako se podaci zapisuju

SQL Server radi s jednostruko povezanim listama koje sadrže adrese slobodnih strana bafera. Čim nit koja čita podatke zatraži baferovanu stranu, ona koristi prvu iz liste slobodnih strana. Mi kažemo da su strane sa podacima *prljave* kada sadrže podatke koji nisu zapisani na disku. Svaka strana bafera sadrži brojač reference i indikator koji definiše da li ta strana sadrži prljave podatke. Svaki put kada upit referencira stranu bafera, brojač referenci se uvećava za jedan.

Svaka instanca SQL Servera sadrži lazywriter nit, koja skenira keš bafera. Njen prvi posao posle startovanja sistema je da proveri veličinu liste bafera. Ako je slobodna lista bafera ispod određene zadate vrednosti, koja zavisi od veličine keša, dodaje se još slobodnih strana. Lazywriter niti skeniraju strane bafera i prilikom svakog skeniranja brojač referenci u zaglavlju strane deli se sa 4. Ostatak deljenja se odbacuje. Na kraju, kada se brojač referenci svede na nulu, proverava se indikator prljavih strana. Ako indikator ukazuje da je strana prljava, lazywriter terminira događaj zapisivanja podataka na disk. Događaj pisanja se poklapa sa unapred definisanom vrednošću slobodnih strana bafera, koji je interno određen.

Dnevnici transakcija se ovde pojavljuju zato što se podaci koji se koriste u kešu bafera prvo zapisuju u dnevnike. Cilj prethodnog zapisivanja u dnevnik je da se omogući da SQL Server poništi proces ili ga ponovo izvrši pre nego što podaci budu potvrđeni u tabeli. Nakon toga, u slučaju da se javi neki problem koji traži da se podaci ne zapisuju u tabeli na fizičkom disku, podaci mogu da se potvrde ili vrate. Nakon što su podaci zapisani na disk i potvrđeni, ili ako nema prljavih podataka, strana se oslobađa. Veze između strana u baferu i fizičkih podataka se

uklanjaju. Na kraju se strana iz bafera vraća u slobodnu listu. Veličinu liste slobodnih strana bafera interno određuje SQL Server, a ne može da konfiguriše DBA.

Zapisivanje u dnevniku transakcija je izvan oblasti upravljanja niti koje rade sa baferom i stranama sa podacima. Direktiva *commit* dovodi do zapisivanja svih nezavršenih transakcija na disk.

SQL, optimizacija i obrada upita

Možda će vam se činiti da se reči *query* (upit) pridaje velika pažnja u SQL jeziku. Vrlo je bitno da kao programer baze podataka za PC ili administrator shvatite specijalno značenje i važnost ove reči kod SQL Servera, čak i kod SQL Server Desktop Engine, koja će zameniti Jet, kao difolt Access bazu podataka. Većina korisnika Microsoft Accessa, na primer, nikad nije radila ili koristila upite koji bi krenuli dalje od osnovnih SQL upita, pošto se tamo većinom radi sa vrlo visokim, često vizuelnim nivoom, a malo sa osnovnom bazom. Kod K/S baza podataka i SQL Servera sve počinje i završava se sa SQL upitima.

Kada se jednom suočite sa njim, upit nije ništa drugo do pitanje u vezi s podacima u DBMS sistemu, ali je SQL upit i nešto više a ne samo pitanje. SQL upit može da se kombinuje sa jezikom za definisanje podataka (DDL) i jezikom za manipulaciju podacima (DML) da bi se kreirali objekti u bazi ili da bi se tim objektima upravljalo. Podaci koji se nalaze u objektima za skladištenje podataka mogu se brisati, dodavati i transformisati, a mogu se i analizirati. Sistemima za upravljanje baze podataka se upravlja preko SQL-a.

Svaki proizvođač baze podataka podržava SQL standard. Da bi se podržala korisnička proširenja njihovih proizvoda, oni na izvestan način proširuju SQL. Da bi proširio SQL jezik tako da može da podrži proširenja koja postoje na SQL Serveru, Microsoft je uveo jezik Transact-SQL. Sada se Transact-SQL jezik, ili kraće T-SQL, koristi za komunikaciju sa SQL Serverom. Ako poznajete standardni SQL, onda će prilagođavanje na T-SQL proći bez problema. T-SQL je objašnjen u trećem delu.

Dobra klijentska aplikacija za rad sa bazama podataka nikad ne bi trebalo da breme konstruisanja SQL upita ostavi korisnicima aplikacije. Ja sam tokom godina razvio mnoge aplikacije za rad sa bazama podataka i svaki klijent je zahtevao da se korisniku omogući da sortira, grupiše i interpretira podatke. Čak i najmanje kompanije sa kojima sam radio, koje su imale tri-četiri računara, žele su da znaju šta je to što se ove godine radi bolje ili gore u odnosu na prošlu i tražile su podatke koji će im pomoći da poprave biznis. Mi, kao iskusni administratori baze podataka, i dalje ponekad verujemo da je analiza podataka zadatak samo za Fortunu 500.

Kako omogućiti korisnicima da postavljaju upite i da analiziraju podatke a da ne znaju SQL? To je izazov. Ako mislite da možete da im date jednostavni SQL koji će moći da koriste, onda ste zaboravili na jezik kao što su OLAP kocke. Kada sam se prvi put spotakao o višedimenzioni izraz (MDX), ja sam utrostručio zdravstveno osiguranje.

Ovim poslom ćemo se pozabaviti u narednim poglavljima, posebno u dvadeset trećem, koje se bavi English Query modelom. Dobar početak na ovom put je da shvatite da je SQL jezik vrlo visokog nivoa. Njega su programeri i administratori široko prihvatili kao jezik komunikacije sa bazom podataka, ali na nivou programera i administratora a ne na nivou korisnika. Na ovo ćemo se vratiti u narednim poglavljima.

Nema DBMS sistema, uključujući i SQL Server, koji koristi (ili može da koristi) SQL kao proceduralni ili jezik instrukcija. On se koristi da DBMS sistemu isporuči zahteve klijenta. Server dalje omogućava upotrebu optimalnog računarskog jezika koji se može programirati da bi zadovoljio zahteve korisnika, i to nakon što je SQL kod prebačen.

Na nižem nivou od SQL-a, metodi koji se koriste za manipulaciju i analizu podataka po prirodi su algebarski i računski. Drugim rečima, procedura koja zadovoljava upit ili neku direktivu nad podacima zasniva se na matematičkim formulacijama. Ako bismo svi mi morali da sudiramo "sirovu" relacionu algebru ili relacioni račun, onda ne bi bilo mnogo administratora baze podataka ili SQL programera. SQL Server je precizno programiran da radi sa ovim. Pogledajte sledeći izraz:

```
TX WHERE EXISTS CREDITX
    (CREDITX < 5 AND
    T ( T#:TX, CREDIT:CREDITX, STATE:'NEW YORK'))
```

Ovo je relacioni račun, a SQL verzija je:

```
SELECT * FROM CUSTOMERS WHERE CreditRating < 5 AND State = 'New York'
```

Za većinu nas, ovakva komunikacija sa računarom u najmanju ruku je teška. SQL nam, sa druge strane, omogućava da sa računarom komuniciramo preko pogodnijih termina. Hebrejski jezik je jedno od najboljih poređenja. Ovaj jezik sadrži izvesne glasove i vokale usvojene kod starih sveštenika koji ovaj jezik koriste za komunikaciju sa Bogom (da li bog odgovara drugo je pitanje). Danas, međutim, postoji i varijanta hebrejskog bez vokala i varijacija za biblijski kod, koja ga čini pogodnijim za upotrebu, i nju koriste Jevreji u Izraelu. Postoji i treća varijanta hebrejskog, koja se koristi za pisanje.

Pogled iz SQL-a na osnovne podatke nije mnogo različit od pogleda iz VB-a na mašinski kod. Ako ste pisali kod nižeg nivoa i komunicirate direktno sa hardverom, mogli biste jedino da koristite još asembler. Na isti način, programer ne može da koristi unutrašnji jezik servera baze podataka, osim ako ne ubacuje programsku logiku u mašinu ili ako ne pravi novu mašinu.

Da zaključimo, SQL nudi velikom broju programera, administratora i analitičarima način za konverzaciju sa sistemom za upravljanje bazom podataka. SQL nije idealan i nije u mogućnosti da direktno radi sa strukturama podataka, sistemima za skladištenje i mehanizmima baze podataka.

Računar i dalje treba da prevede naše zahteve na sopstveni jezik, koji je bolje optimizovan i prilagođen za obavljanje relacione algebre i relacionog računa. Ovo polje rada sa bazom podataka naziva se *optimizacija upita*.

Šta je optimizacija upita?

Ako SELECT iskaz iz SQL-a uvodi konstantne procedure u mašinu baze podataka, onda bi svi trebalo da na isti način radimo da bismo dobili skup rezultata. Čisti rezultat bi bio arhitektura koja nije fleksibilna, kao i nemogućnost mašine da koristi prednosti novih tehnologija, nove snage procesora i miliona promenljivih.

Ako se posmatra rešenje problema analize, onda treba da naučite da uvek postoji više rešenja problema, kao i milioni mogućih rezultata, dobrih ili loših. Metodi koji se koriste za postavljanje upita nad podacima bave se problemima koji se mogu rešiti na mnogo različitih načina.

Izraz "postoji mnogo načina da se odere mačka" opisuje problem za čije rešenje se koristi optimizacija upita. Skup rezultata može biti dobijen kroz mnogo različitih koraka, a posao optimizatora je da proceni svaki upit i da odredi najbrži način za rešavanje problema i to sa najmanjon cenom. Analiza cena koju vrše mnogi serveri baze podataka, a posebno SQL Server, najviše se primenjuje na ulazno/izlazne operacije za datoteke (za razliku od ulaza/izlaza za vrste), pri čemu se u obzir uzimaju i memorija i ostali resursi. Analiza cene se nakon toga kombinuje za svaki korak u upitu, a rezultat je *optimalni plan izvršenja*.

Naravno, moguće je da neki genije pokaže da teorema koju je optimizator koristio nije u redu, na kraju krajeva ljudi su odgovorni za logiku koju računar koristi prilikom donošenja odluka. Ipak, računar je u stanju da slaže takve kockice brže nego čovek, bez obzira na koeficijent inteligencije. Računar, ili njegov modul za optimizaciju, bolje je opremljen za obavljanje optimizacije, sa prednošću da ima i memoriju koja se može adresirati i koja mu može reći kakvi su podaci u pitanju (kao što su tipovi i struktura podataka), koliko ima zapisa u tabeli, koliko različitih vrednosti itd. Optimizator je u stanju da sve to uzme u obzir i da na osnovu toga najbolje udovolji kapricima svojih korisnika.

Optimizacija je, prema tome, fundamentalna funkcija servera baze podataka, koja je obavezna da bi taj server mogao da da prihvatljive performanse i da nastavi da ih poboljšava. Ona je potrebna i zato što su instrukcije koje se primaju od ljudi, bilo u obliku SQL-a bilo kao XML, suviše visokog nivoa da bi mogle da posluže za poboljšanje performansi. Što SQL postaje šire prihvaćen, to server baze podataka ima više mogućnosti da postane bolji u onome što radi na nižem nivou. Ljudi nisu svesni koliko je SQL iskaz optimalan za server, ali će primetiti da li upit traje nekoliko milisekundi, nekoliko sekundi, minuta, pa i sati kod složenijih upita, i kada nema dovoljno resursa. Drugim rečima SQL Server mora da se pozabavi optimizacijom upita. Njegov opstanak zavisi od toga.

Proces optimizacije

Svaki server baze podataka, svaka verzija SQL Servera i svakog drugog sistema za upravljanje bazama podataka, kao što su Oracle ili Subase, moraju da budu u stanju da prihvate standardni SQL zahtev i da ga konvertuju u proceduralni jezik koji svaki pojedinačni proizvod koristi. Kako taj proizvod konvertuje SQL iskaz i šta on nakon toga radi zavisi od samog proizvođača. Algoritmi, tehnike i veštine koje se koriste za sticanje položaja na tržištu odvajaju jedan proizvod od drugog.

Iskaz UPDATE može da pokrene okidače, funkcije, uskladištene procedure i ograničenja integriteta. Ovi događaji mogu pokrenuti lanac rekacije, koji dovodi do niza poređenja i složenih proračuna vezanih podataka. Jednostavni UPDATE iskaz bi mogao da dovede do krupnog plana izvršenja... u pitanju je efekat domina.

Iako se rad optimizatora zasniva na ceni, on ne može uvek biti tačan. Štaviše, može se desiti i efekat smanjenja onoga što se vraća u trenutku kada optimizator ne može da vrši dalju optimizaciju i da performanse mogu čak i da se pogoršaju. Zbog toga u DBMS sistem mora biti ugrađena izvesna heuristička logika, koja mu omogućava da zna kada treba da se podvuče crta ... ili da odluči kada više postaje manje. SQL Server nije izuzetak.

SQL Server obavlja izvestan broj osnovnih koraka koji su potrebni da bi se izvršila optimizacija upita, ali je ovde vredno pomenuti tri:

- Mra prvo da transformiše upit u jezik nižeg nivoa koji se koristi kod mašine. On to radi preko analize SQL iskaza.
- Dalje mora da identifikuje moguće procedure koje se koriste za zadovoljenje upita. Upit može da bude za manipulaciju, za rezultat ili za oboje. Ovo je osnovni proces optimizacije.
- Nakon toga, moraju da se generišu planovi upita i da se izabere onaj sa najmanjom cenom.

Hajde da pogledamo prve korake koje SQL Server obavlja prilikom optimizacije:

SELECT iskaz ili neki drugi SQL iskaz je standardni format zahteva i nema uticaja na procedure, poziv i funkcije mašine i njenu mogućnost da vadi ili ažurira podatke. Odnosi se samo na zahtev za podacima ili za njihovim ažuriranjem i predstavljanjem tih podataka na izvestan način. Ne govori serveru kako da dobije te podatke, ali server razume šta se od njega traži. On to razume na osnovu sintakse i ključnih reči iskaza SELECT. Drugim rečima, SELECT iskaz je želja, ili jedna strana.

NAPOMENA

Ranije sam pomenuo XML. Važno je da shvatite da je XML proširenje T-SQL-a koje se koristi kao novi jezik za definisanje zahteva na SQL Serveru. U proizvod je sada ugraden snažan XML parser. XML sam razmotrio u nekoliko poglavlja trećeg dela, posebno u dvadeset trećem. ■

Kod zapisa se SELECT iskaz koristi da bi definisano samo sledeće:

- Format skupa rezultata koji se vraća do klijenta. Obično je difolt format dovoljan za klijenta, pošto on sledi semantički raspored kolona, ali se često dešava da dodatni kvalifikatori, kao što su klauzule GROUP BY ili ORDER BY daju mašini instrukcije da predstavi podatke u alternativnom formatu.
- Objekte (uglavnom tabele) koji sadrže podatke ili delove podataka koje klijent traži. FROM klauzula je pomoćni element SELECT iskaza koji identifikuje izvor podataka. U ovoj klauzuli se može zadati više izvora.
- Način na koji se tabele ili drugi izvori podataka povezuju radi dobijanja skupa rezultata. Klauzula JOIN je druga pomoćna klauzula koja definiše logičke veze između dva ili više izvora podataka.
- Uslove ili kriterijume koje vrsta sa podacima mora da zadovolji pre nego što bude ubačena u proces izbora. Pomoćne klauzule ovde su WHERE i HAVING.

Pre nego što se SQL iskaz ili drugi servisi koji se zahtevaju proslede do optimizatora upita, mora da se proceni njegova tačnost. Ovo radi parser iskaza, koji je deo veće relacione mašine, nasuprot jezgru o kome smo ranije govorili. Parser osigurava da je iskaz sintaksički tačan i da je razumljiv i produktivan. Ako SQL kod koji šaljete ima grešku, parser hvata izuzetak i vraća sintaksičku grešku do klijentske aplikacije. (Greška se šalje do klijenta izvan opsega, odnosno preko posebnog kanala, a ne preko veze koja je uspostavljena sa klijentom. Mehanizam za izveštavanje o greškama SQL Servera je detaljnije objašnjen u šesnaestom poglavlju.)

Ako sintaksa prođe, parser deli iskaz u logičke jedinice. Te jedinice su ključne reči, parametri ili argumenti, operatori i identifikatori. Parser skenira tekst, proverava delimitere i određuje šta su objekti, a šta vrednosti i promenljive. Prilikom slanja, možete da definišete način na koji SQL Server treba da procenjuje. Na primer, možete da promenite karakter koji predstavlja delimiter u SQL Server iskazu. Ovo se može uraditi interaktivno preko podešavanja u Enterprise Manageru ili preko T-SQL koda (na sličan način kako to radi iskaz define u C ili C++ kodu). Ovakve napredne stvari ćemo detaljnije objasniti u trećem poglavlju.

SAVET

Mnogi eksperti će u radu sa SQL Serverom doći u situaciju koja od njih zahteva da posete optimizator i daju mu sugestije ili "savete" za optimizaciju. Iako vam ovo možda i ne izgleda tako važno, treba da shvatite koji faktori utiču na performanse. ■

Kada se analiza završi, parser obavlja funkciju grupisanja, preloma složenih iskaza u manje logičke kolekcije rutina i operacija.

Na taj način, preko tipičnih tehnika za rešavanje problema određuje se najbolji plan za izvršenje upita. Drugim rečima, poslednja funkcija parsera je da pomogne optimizatoru, koji dolazi nakon toga, da razlikuje "stabla od šume" u složenim zahtevima i da prema tome predloži najbolji plan izvršenja. Zato se format u koji se prevodi originalni T-SQL iskaz naziva *stablom upita*.

NAPOMENA

Stablo upita se ponekad naziva i stablo sintakse ili stablo sekvenci. ■

Jezgro na kraju kompajlira stablo upita u niz operacija koje se obavljaju nad skupovima vrsta iz baze podataka. Jezgro dobija pristup do baze podataka preko servisa mašine za skladištenje. Relaciono jezgro i jezgro za skladištenje komuniciraju preko OLE DB API-ja, i to preko standardnih SQL SELECT iskaza koje OLE DB može da prosledi do mašine za skladištenje (kao što znate, OLE DB je složen za kodiranje, odatle i ADO, ali je zaista efikasan i idealan za komunikaciju dva jezgra). Kod koji sledi predstavlja jednostavan SELECT upit koji je prosleđen od relacione mašine do mašine za skladištenje:

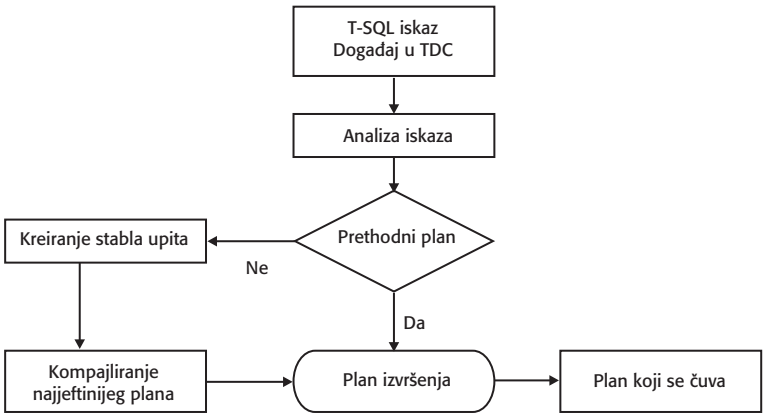
```
SELECT CustId FROM CustDetails
```

Očigledno je da ovde plan izvršenja nema mnogo šta da kaže jezgru, jer se radi o najjednostavnijem iskazu. Plan bi se, prema tome, u ovom slučaju sastojao od jednostavnog SELECT iskaza. Ovo je predstavljeno jednostavnim dijagramoma toka sa slike 4.2.

Operacije se definišu nad tabelama i indeksima ili i nad jednim i drugim. Ako nema indeksa koji su kreirani za te podatke, plan izvršenja se pravi jedino da bi se obavilo neophodno skeniranje tabele.

Ali, šta se dešava kada preko žice dođe neki složeniji iskaz? Evo nečega što je malo složenije, ali i dalje relativno jednostavno:

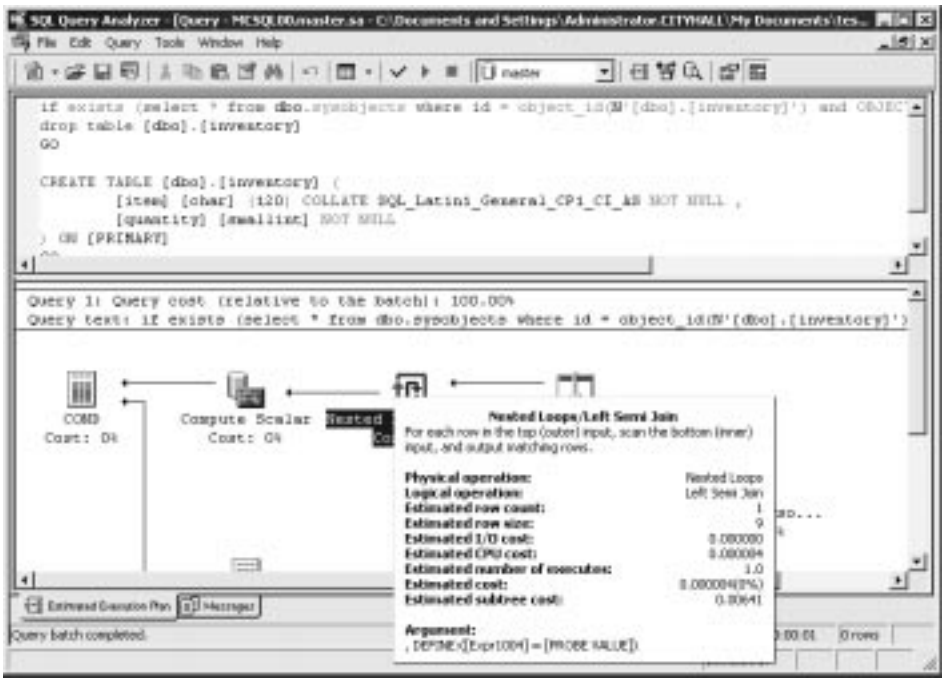
```
USE PUBS
SELECT OD.OrderID, OD.CustomerID, CUST.ContactName
 FROM dbo.Orders OD INNER JOIN
 dbo.Customers CUST ON OD.CustomerID = CUST.CustomerID
```



SLIKA 4.2 *Koraci koje SQL Server obavlja prilikom analize i optimizacije T-SQL iskaza i kreiranja plana izvršenja*

Ovde je prikazan zahtev za dva skupa rezultata iz dve tabele koje su spojene u treći skup rezultata. Iako ovo još uvek nije najsloženiji SELECT iskaz, jezgro ga razlaže na dva koraka, po jedan za svaku tabelu. Po jednom, za svaku tabelu, biće pozvan OLE DB metod OPENROWSET, a nakon toga će se obaviti potrebna spajanja za skup rezultata koji se vraća. Kada relaciono jezgro završi posao sa mašinom za skladištenje, formatira se skup rezultata koji se šalje do klijenta. Tom prilikom se dobija potrebni tabelarni format, koji se šalje nazad preko žice.

Kakve intervencije, uvid ili kontrolu imate nad procesom koji sam upravo opisao? Dobra vest za programere i administratore baze podataka je da je SQL Server 2000 opremljen nekim čudesnim alatima. Osim saveta upita, postoji alat po imenu SQL Query Analyzer (ilustrovan na slici 4.3) koji omogućava da testirate sintaksu upita, proveravate plan izvršenja, procenjujete plan izvršenja itd. On čak omogućava da proveravate metriku koja je izvedena u procesu optimizacije, u proračunima, kao i u koracima koji su izvršeni da bi se dobio plan izvršenja. Ja neću ulaziti u detalje o Query Analyzeru, pošto ćete ga mnogo više koristiti u poglavljima trećeg dela. Kasnije ćete istražiti i rad sa čarobnjakom Index tuning Wizard (za podešavanje indeksa), koji smo pomenuli u drugom poglavlju.



SLIKA 4.3 *Query Analyzer u akciji. Priprema plana izvršenja za složeni upit*

NAPOMENA

DML, DDL itd. imaju specijalne zahteve prilikom obrade, koje ćemo detaljnije razmotriti u poglavljima 15, 16, 19 i 20. ■

Način na koji procesor za obradu upita SQL Servera rešava indekse, poglede, podeljene poglede, uskladištene procedure itd, biće objašnjen u poglavljima koja se bave tim oblastima.

Keširanje i ponovna upotreba planova izvršenja

Većina SELECT iskaza je identična ili se razlikuje samo po vrednostima koje su zadate u kriterijumu pretraživanja. Pogledajte sledeće SELECT iskaze:

```
SELECT * FROM CustOrds WHERE OrdValue < '400'
SELECT * FROM CustOrds WHERE OrdValue < '1000'
SELECT * FROM CustOrds WHERE OrdValue < ?
```

Nema sumnje da jedan inteligentan optimizator može da otkrije da će prethodni SELECT iskazi verovatno dovesti do istog ili vrlo sličnog plana izvršenja. Prema tome, nema smisla da SQL Server ponovo kompajlira plan izvršenja, ako je pre izvesnog vremena neki od iskaza već bio kompajliran.

Zbog toga SQL Server održava pul u memoriji, tzv. *keš procedura*, u kome se čuvaju i planovi izvršenja i baferi podataka. Ja bih mogao da dodam da su prethodni primeri vrlo jednostavni, tako da SQL Server može da odluči da je cena skladištenja takvog plana veća nego da ga ponovo napravi. Onda se plan možda i neće čuvati. Veličina memorijskog pula je različita i može da varira od sistema do sistema. Ona se takođe menja u zavisnosti od sistema. Bilo koji broj korisnika može ponovo da koristi plan izvršenja.

Da bi dalje podržao mogućnost ponovne upotrebe, SQL Server održava i parametarsku strukturu podataka, koja se koristi za čuvanje parametara koji su poslati od svakog korisnika. Ove strukture su poznate pod imenom *kontekst izvršenja*. Svakog korisnika, odnosno svaku vezu koja izvršava upit, SQL Server postavlja u kontekst izvršenja.

Plan izvršenja koji odgovara upitu korisnika može da se koristi umesto kompajliranja novog plana. SQL Server jedino treba da dobije parametar ili promenljivu za SQL iskaz i da to ukombinuje sa planom izvršenja koji već postoji. Rezultat je značajno smanjenje vremena i resursa, što dovodi do boljeg vremena odziva.

U trenutku kada se iskaz pošalje na server, server prvo proverava keš procedura, da bi video da li postoji plan izvršenja koji zadovoljava SQL iskaz.

NAPOMENA

Postoje određeni faktori koji mogu da dovedu do toga da SQL Server odbaci plan. Promena tipa podatka je takav faktor. Ovim ćemo se pozabaviti u dvadeset prvom poglavlju. ■

Ušteda dolazi od otklanjanja potrebe za kreiranjem novog plana izvršenja. Očigledno je, prema tome, da u svom kodu možete mnogo uraditi da biste bili sigurni da vaš SQL iskaz ima veliku verovatnoću da bude uparen sa nekim planom izvršenja. Vaš cilj kao administratora baze podataka i programera sistema je da povećate šansu da SQL Server ponovo koristi složene planove koji su već kompajlirani. Upotreba parametara u uskladištenim procedurama i markera parametara u ADO-u, OLE DB-u ili ODBC aplikacijama predstavlja duži način da to uradite. Nema sumnje da se stvari komplikuju kada počnete da radite sa NULL vrednostima i sa trodimenzionalnim ili četvorodimenzionalnim vrednostima, čime ćemo se pozabaviti u šesnaestom poglavlju.

Plan izvršenja i algoritam pronalaženja istog konteksta su se pojavili u verziji SQL Server 7.0 i SQL Server 2000. Ovo je dovelo do značajnog poboljšanja performansi u odnosu na prethodne verzije. SQL Server 2000 čuva plan izvršenja u kešu procedura sve dok je potrebno. Samo stari planovi i oni koji se ne koriste izbacuju se iz keša u trenutku kada je potrebno još prostora i, očigledno, kada se restartuje sistem.

NAPOMENA

Direktiva koju možete da upotrebite za čišćenje keša procedura je DBCC FREEPROCCACHE. Ovim se SQL Server tera da ponovo kompajlira proceduru. ■

Algoritam održava brojač koji sadrži kopiju faktora cene upita. Ova kopija se povećava za vrednost cene u polju za starost, svaki put kada se objekat referencira u nekoj vezi. Na primer, neka plan upita ima faktor cene osam. Svaki put kada se referencira, vrednost brojača se duplira, odnosno dobija vrednosti 16, 32, 64 itd. Proces lazywritera, koji smo objasnili ranije u ovom poglavlju, periodično skenira keš procedura i smanjuje svaki broj u polju za starost za jedan. Eventualno, ako plan izvršenja nije korišćen, smanjuje se na 0 i prema tome se ignoriše. Plan se čuva u kešu procedura sve dok SQL Serveru ne zatreba memorija. Bilo koji plan koji ima brojač nula biće isteran.

Postoje i drugi razlozi za uništenje plana izvršenja. To su:

- Šema je promenjena, ili su objekti, na primer tabele, uklonjeni ili promenjeni (primer iskaza koji dovode do pražnjenja keša su ALTER TABLE i ALTER VIEW).
- Indeks koji se koristi u planu je izbrisan.
- Eksplicitno ste rekli da želite da se izvrši ponovna kompilacija (ovo se radi tako što se pozove uskladištena procedura sp_compile).
- Ključevi su se značajno promenili. Ovo bi moglo da rezultuje prekomernom obradom transakcija, koja bitno menja tabelu koja se koristi u upitu.
- Za tabele koje se koriste u planu izvršenja postavljeni su okidači, uskladištene procedure, funkcije itd.
- Neka funkcija koja dovodi do značajnog rasta tabele može da učini da plan izvršenja postane zastareo.

Optimizacija i planovi izvršenja su možda zamorni, ali su vrlo bitni kada dođe do finog podešavanja i optimizacije sistema i aplikacija, kod DBMS sistema generalno i pojedinačno kod SQL Servera. U narednim poglavljima će se vršiti nadgradnja, koja počiva na osnovama datim u ovom poglavlju.

Unutar tokova podataka, ponovo

U prethodnom poglavlju, u odeljku "Unutar tokova podataka SQL Servera", razmotrili smo tabelarne tokove podataka (TDS), mehanizam za premeštanje paketa koji podatke smešta u specijalni format poslat na server i sa njega. TDS je protokol koji se konstantno razvija, a SQL Server sazreva kao najbolje vino, tako da morate da budete sigurni da stari klijenti koji su konfigurisani za neki stariji nivo TDS-a mogu da šalju i primaju podatke sa servera.

U većini slučajeva, podrška nižeg nivoa ne zahteva specijalne intervencije ili rad sa vaše strane, pošto SQL Server 2000, koji ima verziju TDS 8.0, može da komunicira sa TDS verzijama 7.0 i verzijom TDS 4.2. Ipak, postoje trenuci kada morate i vi da se uključite.

Podrška naprednih tipova podataka koju ima TDS 8.0 može da zaguši TDS 4.2 klijenta. TDS 4.2 je originalno bio uveden u verziji SQL Server 4.2, protokol je dalje nasleđen i u verzijama SQL Server 6.0 i 6.5. Postao je zastareo nakon velikog promena u SQL Serveru urađenih 1997. i 1998, kada su se pojavili SQL Server 7.0 i TDS 7.0. TDS se ponovo menja sa pojavom verzije SQL Server 2000.

Konkretno, TDS 4.2 klijenti ne podržavaju Unicode tipove podataka. Kada se desi da TDS 4.2 klijent vuče Unicode podatke iz tabele sa verzijom 8.0, relaciona mašina pokušava da konvertuje podatak u neki stariji tip pre nego što ga pošalje do klijenta. Pošto taj Unicode karakter nema ASCII ekvivalent i pošto postoji više od 65.000 takvih karaktera, prilikom slanja šalju se praznine. Podaci koji se vraćaju do klijenta u tom slučaju će biti puni praznina.

NAPOMENA

Unicode, ili Unicode Characet System (UCS), jeste sistem od dva bajta (16-bitni) koji predstavlja sve jezike koji postoje na svetu i to preko skupa od 65.536 karaktera. ASCII je 7-bitni ili 8-bitni sistem koji može da podrži samo 256 karaktera. Poređenje TDS verzija 4.2 i 8.0 je kao poređenje rečnika deteta od dve godine i rečnika odrasle osobe. ■

TDS 4.2 klijenti ne podržavaju tipove podataka char, varchar, binary i varbinary, koji su se pojavili u verzijama 7.0 i 8.0, a koji su veći od 255 bajtova. Prema tome, SQL Server će skratiti ove vrednosti kada treba da se do klijenta vrate preko TDS 4.2. Kao što smo pomenuli u drugom poglavlju, prethodno pomenuti tipovi podataka mogu da sadrže podatke veličine do 8.000 bajtova.

TDS 7.0 klijenti su malo kraći u odnosu na SQL Server 2000 i TDS 8.0. Tip podatka SQL_variant (detaljnije objašnjen u šesnaestom poglavlju) nov je u verziji SQL Server 2000. Relaciona mašina mora da, pre slanja, bilo koji podatak tipa SQL_variant konvertuje u Unicode.TDS 4.2 klijenti ne mogu da ga primene.

Šta kada se vaša strategija za predstavljanje podataka zasniva na TDS 8.0? Ako ste novi u radu sa podrškom za SQL Server i ako se tek upoznajete sa funkcijoma DBA, onda morate da znate šta u tom slučaju treba uraditi.

Prvo, kao što ću često ponavljati u ovoj knjizi, "tanje je bolje". Što je klijent tanji, manje ćete posla kasnije morati da uradite, bilo kao programer bilo kao DBA. To kasnije se odnosi na trenutak pojave nove verzije SQL Servera (otprilike na dve godine). Za administratora nije ništa gore nego kada mora da ažurira stotine ili hiljade punih klijenata, koji imaju više komponenti nego hronometar iz ranih dvadesetih. Mi ćemo se u četrnaestom poglavlju ponovo vratit na debatu o obradi kod klijenta ili servera, kao i u poglavlju 23 kada ćemo se baviti strategijama za razvoj aplikacija u n nivoa.

Drugo, kada se klijent povezuje sa SQL Serverom, njegovi TDS paketi se prave na osnovu logike koja je ubačena u provajder podataka koji se koristi. Provajder podataka bi mogao da bude ili Microsoft OLE DB za SQL Server, ili ODBC drajver ili DB-Library DLL (koji je trenutno na zalasku karijere). Prema tome, treba da osigurate da vaši klijenti pozivaju pravu verziju drajvera ili provajdera. Ako ne uradite tako, onda će TDS paketi koje oni šalju biti usmereni na prethodne verzije SQL Servera. Ovo neće dovesti do velikih problema sa podacima koji dolaze, pošto će biblioteke na strani servera biti u stanju da razumeju pakete. Problem se javlja kada se paketi šalju nazad do klijenta i kada podaci treba da se transformišu da bi se prepoznali. Grudvice od hiljade bajtova se neće prebaciti na drugu stranu, pošto SQL Server zna da ste vi povezani sa TDS-om 4.2 i sam vrši potrebna prilagođavanja.

Hvatanje relacionog voza

Ono što sam opisao u ovom poglavlju premalo je u odnosu na ono što SQL Server radi da bi upite izvršio najbrže što može i da bi podatke brzo vratio do nestrpljivih klijenata. Postoji veliki broj povezanih procesa koji se dešavaju, a koje ću opisati u narednim poglavljima kada bude potrebno. Takođe, upoznaću vas sa tehnikama koje mogu da proizvedu efikasnije upite, što dovodi do bržeg odgovora SQL Servera 2000.