SUURTEN OLIOIDEN TALLENNUS TIETOKANTAAN

Marja Pennanen

23.5.2008

Joensuun yliopisto

Tietojenkäsittelytiede

Pro gradu -tutkielma

Tiivistelmä

Suuria olioita, joista puhutaan myös pitkinä bittijonoina tai suurina objekteina, ei juurikaan viime vuosituhannella tallennettu tietokantoihin. Esimerkiksi kuvasta tallennettiin yleensä vain viite sen sijaintiin tiedostojärjestelmässä. Nykyisin laitteistojen ja tietokantojen kehittyessä on myös suurten tiedostojen tallennus tietokantoihin tullut mahdolliseksi. Tässä tutkielmassa tarkastellaan niitä suurten datamäärien tallentamiseen liittyviä ongelmia tietokannan hallintajärjestelmissä, joita suuria olioita tietokannoissa käyttävät sovelluskehittäjät, järjestelmänvalvojat ja käyttäjät voivat kohdata. Tutkielmassa tarkastellaan myös tietokantojen rajapintoja ja tutustutaan hieman tarkemmin ODBC- ja JDBC-rajapintoihin ja esitellään erilaisia suurten olioiden tallennusmenetelmiä. Lisäksi tutkielmassa tarkastellaan tutkimusta, jossa suurten olioiden tallentamista tiedostojärjestelmään ja tietokantaan on vertailtu. Esimerkinomaisesti tarkastellaan myös suurten olioiden tallentamiseen kehitettyjä tietotyyppejä Oracle- ja MySQL-tietokannoissa ja sekä suurten olioiden käsittelyä niissä.

ACM-luokat (ACM Computing Classification System, 1998 version): H.2.3, H.2.2

Avainsanat: tietokanta, suuret oliot, JDBC, ODBC

Sisällysluettelo

1. **Johdanto**

Suuret oliot ovat isoja tiedostoja, näihin lukeutuvat siis kuva- ääni- ja videotallenteet sekä esimerkiksi ohjelmatiedostot ja tietenkin isot tekstitiedostot. Muutamia esimerkkejä suurista olioista:

- toimistojärjestelmien dokumentit, taulukot ja esitysmateriaalit
- tutkimus ja kehitysjärjestelmien kuva- ja mittaustiedot
- satelliittikuvat
- paikkatiedot
- terveystiedot ja lääketieteellisen tutkimuksen tiedot
- biologiset tutkimustiedot esimerkiksi genomit eli organismin koko perintöainesta koskevat tiedot
- teollisuuden tuotantotiedot
- valtion ja kuntasektorin hallinnolliset tiedot
- erilaiset julkaisut ja web sovellukset.

Tietokantoihin talletettavien tietojen suurin mahdollinen pituus on kasvanut vuosien saatossa tietokantojen ja laitteistojen tallennuskyvyn kehittyessä ja muistien halvetessa. Ensimmäisissä tietokannoissa tietuepituudetkin pyrittiin minimoimaan tallennuskapasiteetin säästämiseksi. Tallennuskapasiteetin kasvaessa ja internetin yleistyessä ovat myös tiedostokoot kasvaneet ja yhä pitempiä tiedostoja tallennetaan myös tietokantoihin.

Ensimmäiset tietokantamallit 1960-1980-luvuilla olivat kolmea tyyppiä: hierarkkiset, verkkomalliset ja (käytännössä 80-luvulta) relaatiomalliset [23]:

- hierarkkinen IBM IMS (esiteltiin 1968) (DL/1)
- verkkomallisia oli useita, esim IDMS, IDS, MDBS
- relaatiomallisia satoja mm: DB2, Oracle, Ingres, Informix.

Näiden lisäksi on nykyään myös olio- ja XML -tietokantoja ja joihinkin relaatiotietokantoihin on tullut myös oliopiirteitä. Nykyisin käytetään eniten relaatiomallisia tietokantoja, joita hallitaan SQL (Structured Query Language) -kielellä, jonka ANSI-standardi hyväksyttiin vuonna 1986 ja ISO-standardi vuonna 1987 [22].

Eniten käytetyt tietokannat ovat: Oracle, IBM DB2, Microsoft SQL Server, MySQL, PostgreSQL Sybase, Informix, Teradata, Microsoft Access, FileMaker, dBASE, Clipper, FoxPro, Adabas, Interbase, Nomad, Superbase ym.

Fyysisen tietokannan ja käyttäjän välillä on tietokannan hallintajärjestelmä (DBMS, Database Management System), joka suorittaa käyttäjän esittämät tietokantaoperaatiot [5].

Suurten olioiden ja pitkien bittijonojen sijaan puhutaan myös suurista objekteista (large objects eli LOBs). Eri tietokannoissa suurten olioiden tallentamiseen tarkoitettujen tietotyyppien nimet voivat vaihdella. Joissakin tietokannoissa voi olla myös useita eri tietotyyppejä riippuen talletettavan tiedon pituudesta ja tyypistä. Varsinaisesti tietokanta-alueelle tallentamisen lisäksi suuret oliot voidaan tallentaa tietokannan ulkopuolelle. Tietokantaan tallennetaan vain tieto siitä, missä varsinainen data sijaitsee. Tietokannoissa on usein myös funktioita näiden tietotyyppien käsittelyyn.

Tutkielman luvussa 2 tarkastellaan tietokannan hallintajärjestelmiä, luvussa 3 tietokanta-rajapintoja, joista esimerkkeinä ovat ODBC ja JDBC. Luku 4 on omistettu erilaisille fyysisille toteutuksille ja siinä käydään läpi muutamia, lähinnä jo historiallisia tietojen tallennusjärjestelmiä. Luku 5 keskittyy Microsoftin ja Berkleyn yliopiston tekemään mielenkiintoiseen suurten olioiden tallentamiseen liittyvään tutkimukseen ja sen tuloksiin. Luvut 6 ja 7 esittelevät esimerkkitietokantojen Oraclen ja MySQL:n suurten olioiden tallentamiseen liittyviä tietotyyppejä ja tallentamiseen liittyviä muita seikkoja. Luvussa 8 tehdään yhteenvetoa suurten olioiden tallennuksen nykytilasta.

2. **Tietokannan hallintajärjestelmä**

Tietokantoja ei voi käsitellä ilman tietokannan hallintajärjestelmää (Database Management System, DBMS). Siksipä tässäkin tutkielmassa on luotava katsaus sen toimintaan.

Tietokannan hallintajärjestelmä on joukko ohjelmia, jotka mahdollistavat tietokannan luonnin, käytön ja ylläpidon [28]. Eri tietokannoilla on omat hallintajärjestelmänsä ja niillä on yleensä myös omat ODBC (Open Database Connectivity) -ajurinsa. Tietokannan hallintajärjestelmä käsittelee käyttäjien ja toisten ohjelmien pyynnöt siten, että näiden ei tarvitse tietää, miten ja minne tarvittava tieto on varsinaisesti talletettu. Tietokannan hallintajärjestelmä varmistaa myös tiedon eheyden ja huolehtii tietojen suojauksesta siten, että vain ne, joilla on siihen oikeudet, pääsevät käsiksi tietoihin.

Tietokannan hallintajärjestelmä sisältää kyselykielen, jolla tietokantaa muokataan ja jolla siihen kohdistuvat haut tehdään. Yleisin tällainen kieli on SQL (Structured Query Language). SQL on ANSI (American National Standards Institute) -standardoitu kieli, josta on valitettavan monta eri versiota ja monia erilaisia laajennuksia. Ne kaikki kuitenkin sisältävät samat päähakusanat (SELECT, UPDATE, DELETE, INSERT, WHERE jne.).

Tietokantojen hallintaan kehitetyt kielet sisältävät yleensä kolmentyyppisiä osioita:

- tietojen määrittelykielen (Data Definition Language, DDL), jolla tietokantamalli määritellään

- tietojen käsittelykielen (Data Manipulation Language, DML), jolla tietokannasta haetaan tietoa sekä lisätään, muokataan ja poistetaan tietoa

- tietojen valvontakielen (Data Cntrol Language, DCL), jolla valvotaan tietojen saatavuutta.

Tietokannan hallintajärjestelmää voidaan kuvailla attribuuttien hallintajärjestelmäksi, missä attribuutit ovat jotain ominaisuutta kuvaavia tietoja [32]. Esimerkiksi ”silmien väri” on yksi ihmistä kuvaava attribuutti. Attribuutin arvo on jokin väri kuten ”sininen”, ”harmaa”, ”ruskea” jne. Suurten olioiden esim. kuvan tapauksessa kyselyjen suorittaminen voi olla todella

hankalaa. Algoritmien pitää olla hyvin kehittyneitä, jotta kuvainformaatiosta voitaisiin erottaa silmien osuus.

Tietokannan hallintajärjestelmä hoitaa kyllä säännöllisesti tarvittavat palvelut tai ainakin pienikokoisten attribuuttien hallinnan toiminnot. Näin käyttäjä saa paremman toiminnallisuuden omiin sovelluksiinsa tarvitsematta ohjelmoida kaikkea alusta alkaen.

2.1. Tietokannan hallintajärjestelmän ACID-ominaisuudet

On hyvin tavallista, että tietokannan hallintajärjestelmä niputtaa yhden tai useamman operaation yhdeksi tapahtumaksi (transaction). Transaktio on yksikkö, joka pitää suorittaa atomaarisena (atomically) ja selvästi erillään (isolation) muista tranaktioista [3]. Hyvin toimivan transaktion sanotaankin noudattavan ACID-testiä, missä

A (atomicity) atomaarisuus tarkoittaa, että kaikki tai ei mitään transaktiosta toteutetaan.

C (consistency) ristiriidattomuus tai eheys tarkoittaa, että tietokanta säilyy eheänä.

I (isolation) eristys, toiminto suoritetaan kuin muita toimintoja ei samanaikaisesti olisikaan suoritettavana.

D (durability) kestävyys, transaktion vaikutus tietokantaan ei häviä koskaan sen jälkeen kun se on suoritettu.

2.2. Tietokannan hallintajärjestelmän palvelut

Tietokannan hallintajärjestelmän palveluihin kuuluvat mm. seuraavat: kyselyjen suorittaminen, tietojen varmistus ja toisintaminen (replikaatio), sääntöjen toimeenpano, tietoturva, laskutoimitukset, muutos- ja käyttölokit, automaattinen optimointi ja metadatan talletus [32].

Kysely on menetelmä, jolla saadaan selville erilaisista lähtökohdista ja eri ominaisuuksien yhdistelmillä johdettuja attribuuttien arvoja. Esimerkiksi: kuinka moni työntekijä asuu Joensuussa? Kyselyjen tekemiseksi tarvitaan kyselykieli, raporttien tuottaja ja tietojen turvaamismenettely. *Tietojen turvaaminen* estää tietojen luvattoman katselun ja muokkauksen.

Salasanan avulla käyttäjät pääsevät käsiksi koko tietokantaan tai vain niihin osiin, joihin heillä on oikeus. Esimerkiksi työntekijätietokanta voi sisältää kaiken yksityistä työntekijää koskevan tiedon, mutta vain yksi käyttäjäryhmä voi katsella ja käsitellä palkkatietoja ja jokin toinen työhistoria ja terveystietoja. Jos tietokannan hallintajärjestelmä tarjoaa tavan tutkia, lisätä ja muokata tietokantaa interaktiivisesti, niin tämä mahdollistaa persoonallisen tietokannan hallinnan. Silti toiminnoista ei välttämättä jää jälkeä tarkistuksia varten tai tietokannan hallintajärjestelmä ei tarjoa tällaista palvelua usean käyttäjän järjestelmissä. Nämä valvontatyökalut ovat käytössä vain, kun joukko sovellusohjelmia räätälöidään erikseen jokaiselle tietoerälle ja ylläpitotoiminnolle. Suurten olioiden luominen, hakeminen ja muokkaaminen vaativat tietokannanhallintajärjestelmältä hyvin kehittyneitä ominaisuuksia.

Varmuuskopioita pitää tehdä säännöllisesti siltä varalta, että kovalevy tai jokin muu laite menee rikki. Säännöllinen varmuuskopiointi saattaa olla tarpeen myös sellaiselle etäorganisaatiolle, joka ei pääse helposti käsiksi alkuperäiseen tai alkuperäinen on kookas, kuten suurten olioiden tapauksessa usein on. Tällöin jo pelkästään tietojen hakeminen voi vaatia suhteettoman paljon aikaa.

Tietoja voidaan myös *toisintaa* (replikaatio) kahden palvelimen välillä niin, että informaatio säilyy yhtenäisenä ja käyttäjät eivät välttämättä tiedä, mitä palvelinta he käyttävät. Varsinkin suurten ja keskisuurten organisaatioiden tiedot sijaitsevat tyypillisesti eri palvelimilla. Tietoja replikoidaan, jotta paikallisilla käyttäjillä olisi nopeammat tiedonsaantiyhteydet, sekä levyrikkojen yms. aiheuttamien tietojen menetysten välttämiseksi. Suurten olioiden käsittelyssä on aiheellista huomioida, että niiden koon takia myös varmuuskopiot ja replikaatiot voivat viedä huomattavia määriä tilaa.

Attribuuteille tarvitaan usein *sääntöjä*, jotta ne olisivat selkeitä ja luotettavia. Esimerkiksi sääntönä voi olla, että autolla voi olla vain yksi moottori (moottorinumero). Jos johonkin autoon yritetään liittää toista moottoria, haluamme tietokannan hallintajärjestelmän kieltävän sellaisen pyynnön ja näyttävän virheilmoituksen. Jos kuitenkin tarvitaan muutoksia mallin määrityksiin, tässä esimerkissä hybridi poltto- ja sähkömoottori autot, niin sääntöjä voidaan joutua muuttamaan. Ihanteellisessa tapauksessa sääntöjä pitäisi pystyä lisäämään ja poistamaan tarvittaessa ilman merkittävää tietorakenteiden uudelleen suunnittelua. Suurille olioille on hyvin hankalaa asettaa minkäänlaisia sääntöjä, sillä yksittäisen tiedon eristäminen ainakin rakenteettomasta tietovirrasta on vaikeasti toteutettavissa. Bittijonon pituudelle voidaan kui-

tenkin joissain tapauksissa asettaa rajoituksia. Tästä esimerkkinä on vaikkapa web-sovellus, joissa esitellään vain pieni osa musiikkikappaleesta tai videosta.

Usein on toivottavaa rajoittaa attribuutin tai attribuuttiryhmän *katselu- ja muutosoikeuksia*. Tämä voidaan hoitaa joko yksilöllisesti tai jakamalla yksilöt ryhmiin ja antamalla oikeudet ryhmille tai hienostuneemmin sijoittamalla yksilöt tai ryhmät johonkin sääntöön, jolle sitten annetaan oikeudet.

Attribuuteille tehtäviin yleisiin *laskutoimituksiin* kuten kertominen, summa, keskiarvo, lajittelu, ryhmittely jne. kannattaa käyttää tietokannan hallintajärjestelmän palveluita. Suurilla olioilla tehdään kuitenkin vain harvoin laskutoimituksia. Esimerkiksi kahden kuvatiedoston yhteenlasku ei välttämättä tuota mitään järkevää tulosta.

Usein halutaan tietää, kuka on käyttänyt mitäkin attribuuttia, mikä on muuttunut ja milloin muutos on tehty. *Lokipalvelut* mahdollistavat tämän pitämällä kirjaa käyttötapahtumista ja muutoksista. Jos suuria olioita muokatessa lokeihin tallennetaan sekä alkuperäinen että muuttunut kenttä kokonaisena, voi lokin koko kasvaa ja viedä tapauksesta riippuen nopeasti huomattavia määriä tallennuskapasiteettia. Tällöin kannattaakin harkita, mitä tietoja lokiin talletetaan. Lisäksi lokin siivousta ja mahdollisesti sen kopioimista tai vanhempien osien siirtämistä aika ajoin jollekin ulkoiselle tallennusvälineelle on pohdittava.

Jos jotkin käyttötoiminnot tai kyselyt toistuvat säännöllisesti, tietokannan hallintajärjestelmä voi *optimoida* toimintaansa parantamaan tällaisten toimintojen nopeutta. Joissain tapauksissa tietokannan hallintajärjestelmä pelkästään tarjoaa välineet suorituskyvyn tarkkailuun sallien asiantuntijoiden tekevän tarvittavat päätelmät tutkittuaan kerättyjä tilastoja.

Metadata on tietoa tiedoista. Esimerkiksi listausta, joka kuvaa mitkä attribuutit ovat sallittuja tietojoukoissa, kutsutaan metatiedoksi. Suurten olioiden metadata koostuu tyypillisesti mm. objektin nimestä, pituudesta, päivitysajankohdista, pakkaustiedoista ja formaateista.

3. **Tietokantayhteys**

Tietokantayhteys (database connection) on menetelmä, jolla asiakasohjelmisto on yhteydessä joko samassa koneessa tai muualla sijaitsevaan tietokantapalvelimeen [34]. Yhteyttä tarvitaan kyselyjen lähettämiseen ja vastausten saamiseen. Itse asiassa mitään tietokantakyselyä ei voida suorittaa luomatta ensin yhteyttä tietokantaan. Tietokantayhteys luodaan antamalla tietokanta-ajurille tai muulle yhteydentarjoajalle yhteysavain (connection string) tiettyyn tietokantaan. Yhteysavain sisältää halutun tietokannan ja palvelimen lisäksi yleensä käyttäjätunnuksen ja salasanan sekä mahdollisesti myös muita tietoja mm. kursorin. *Kursori* viittaa tulosjoukon vuorossa olevaan alkioon ja osoittaa välittömästi haun jälkeen tulosjoukon ensimmäistä alkiota.

Monet tietokannat (kuten SQL Server 2000) sallivat yhdelle tietokantayhteydelle vain yhden tietokantaan kohdistuvan operaation kerrallaan. Toisin sanoen, jos tietokantakysely (esim. SQL:n Select-lause) on lähetetty tietokannalle ja tulosjoukko palautetaan, yhteys on avoinna, mutta ei käytettävissä, kunnes asiakas lopettaa tulosjoukon käsittelyn. Toiset tietokannat taas eivät sisällä tätä rajoitetta. Kun käsitellään suuria olioita, tietokantakyselyt vievät tavallista enemmän aikaa jo pelkästään tiedon suuren määrän vuoksi. Tietokantayhteyden laatu vaikuttaa siis huomattavasti järjestelmän suorituskykyyn.

Yhteyden varastointi (connection pooling) on menetelmä, joka kehitettiin parantamaan asiakassovelluksen suorituskykyä ja poistamaan tietokantapalvelimen kuormitusta. Yhteydet ovat rajallisia, kalliita ja voivat viedä suhteettoman paljon aikaa. Siksipä nykysovellukset lähettävät yhteyden takaisin varastoon (pool), kun ne lopettavat sen ja ottavat yhteyden varastosta, kun sellaista tarvitaan, jos jokin yhteys vain on varastosta saatavilla. Tämä edistää käytäntöä, jossa yhteys avataan vain tarvittaessa ja se suljetaan niin pian kuin tehtävä on suoritettu sen sijaan, että sovellus pitäisi yhteyden avoinna koko sen elinkaaren ajan. Näin menetellen suhteellisen pieni määrä yhteyksiä voi palvella suurta määrää pyyntöjä, tätä kutsutaan usein kanavoinniksi (multiplexing).

ODBC (Open Database Connectivity) on sovellusohjelmointirajapinta (Application Programming Interface eli API), joka mahdollistaa yhteyden tietokantaan. JDBC on Java kielinen vastaavan kaltainen tietokantayhteys Muitakin tietokantayhteysrajapintoja on olemassa mm. OLE DB, ADO ja ADO.NET. OLE DB (Object Linking and Embedding, Database) on

Microsoftin kehittämä API erilaisten yhtenäisellä tavalla talletettujen tietojen hakemiseksi. Se on kehitetty ODBC:n seuraajaksi, korkeampitasoiseksi korvaajaksi ja se tukee muitakin kuin relaatiotietokantoja mm. oliotietokantoja ja laskentataulukoita [35].

ADO (ActiveX Data Objects**)** on Microsoftin 1996 julkaisema joukko COM (Component Object Model) -objekteja. ADO on tarkoitettu sovellusten kehittäjien käyttöön ja se yksinkertaistaa SQL-tietokantakyselyjen tulosjoukon käsittelyä. ADO.NET on ADO:n tyyppinen versio .NET-ympäristöön [33].

Seuraavaksi käymme lähemmin läpi avointa ODBC-tietokantayhteyttä ja sen jälkeen hieman myös Javalle kehitettyä JDBC-yhteyttä.

3.1. ODBC

SQL Access Group (SAG), jossa oli useiden valmistajien edustajia, aloitti toimintansa vuonna 1989. Se kehitti SQL Call Level Interface (CLI) -kutsurajapintamääritykset määritelläkseen sovellusohjelmointirajapinnan, joka olisi toimittajariippumaton. Microsoftilta, joka oli yksi SAG jäsenistä, ilmestyi vuonna 1992 ODBC (Open Database Connectivity), joka oli ensimmäinen kaupallinen SQL CLI-sovellus. Alkuperäinen ODBC noudatti kolmen progressiivisen tason periaatetta [21]:

- ydin (core) vastasi SQL CLI -määritysten vaatimuksia
- taso 1 (level 1) sisälsi lisäksi tuen tapahtumille, vieritettävät kursorit ja metadata-metodit pääavaimille ja tallennetuille proseduureille
- taso 2 (level 2) tuki dynaamisia parametreja, tulos- ja palautusarvoja tallennetuista proseduureista ja isolaatiotasojen muokkauksen.

Ilmestymisensä jälkeen ODBC:stä on tullut kaksi uutta versiota. ODBC 2.0 mahdollisti tuen 32-bittisille sovelluksille ja säilytti SAG-yhteensopivuuden. ODBC 3.0 tarjosi täyden yhteensopivuuden X/Open- ja CAE-määrityksiin, SQL:ään, SQL/CLI:hin ym.

ODBC määrittelee joukon matalan tason kutsuja, jotka mahdollistavat asiakasohjelmiston ja palvelimen tiedonvaihdon ja tietojen jakamisen ilman, että kummankaan tarvitsee tietää mitään toisistaan. Se soveltuu mihin tahansa asiakas-palvelin-toimintoon, olivatpa asiakas ja palvelinsovellukset samalla koneella tai eri koneilla tai jopa vaikka palvelin sijaitsisi etäkoneella, jossa on eri käyttöjärjestelmä. Suurten olioiden tapauksessa haku- ja tallennus-ajat kasvavat verkkosovelluksissa johtuen siitä, että siirrossa olio joudutaan pilkkomaan useiksi tiedonsiirtotapahtumiksi.

ODBC-yhteys sallii sovellusten pääsyn tietokannan hallintajärjestelmän tietoihin ja maksimaalisen yhteen toimivuuden. Yksi sovellus voi olla yhteydessä erilaisiin tietokannan hallintajärjestelmiin. Sovelluskehittäjä voi koodata, kääntää ja siirtää sovelluksen ilman, että tekee sen jollekin tietylle tietokannan hallintajärjestelmälle. Käyttäjät voivat sitten lisätä moduuleja, joita kutsutaan tietokanta-ajureiksi, jotka yhdistävät sovelluksen heidän käyttämäänsä tietokannan hallintajärjestelmään.

ODBC- yhteys määrittelee seuraavat [11]:

- ODBC-funktiokirjaston, joka mahdollistaa sovelluksen yhteyden tietokannan hallintajärjestelmään, SQL-lauseiden suorittamisen ja tulosten saannin
- standardin menetelmän tietokantaan kirjautumiseen
- standardin tietotyyppien esitystavan.

Yhteys on joustava mahdollistaen sovellukselle [11]:

- SQL -lauseiden rakentamisen käännös- ja ajoaikana
- saman objektikoodin käytön eri tietokannan hallintajärjestelmille
- yhteyden moniin tietokannan hallintajärjestelmiin.

ODBC-arkkitehtuurissa on yleensä ainakin neljä komponenttia [11], [9]:

- Sovellus eli taulukkolaskentaohjelma, tekstinkäsittelyohjelma, sovelluksen kehityskieli ym., joka välittää SQL-lauseet ja ottaa vastaan tulokset tietokannan hallintajärjestelmältä. Sovelluksen ei tarvitse tietää mitään tietojen tallentamiseen liittyviä yksityiskohtia. Ainoa asia, joka tarvitaan, on tietolähteen nimi (Data Source Name, DSN). Tietolähdenimi on kokoomamerkkijono, joka yksilöi annetun tietokanta-ajurin, tietokannan, tietokannan isäntäkoneen ja valinnaisesti myös tunnistustiedot, jotka mahdollistavat ODBC-sovellukselle yhdistymisen tietokantaan käyttäen standardoituja menettelytapoja.

- Ajuriohjain, DLL (kirjasto-ohjelma), joka lataa ajurit sovelluksen pyynnöstä ja hoitaa yhteyden sovelluksen ja ajurien välillä. Ajuriohjain analysoi tietolähteen nimen.

- Ajuri, DLL, joka käsittelee ODBC-funktioiden kutsut, joita se saa ajuriohjaimelta välittäen SQL-pyynnöt tietylle tietolähteelle ja palauttaen tulokset sovellukselle. Tarpeen vaatiessa ajuri muokkaa sovelluksen pyyntöjä siten, että pyyntö noudattaa asianomaisen tietokannan hallintajärjestelmän syntaksia.

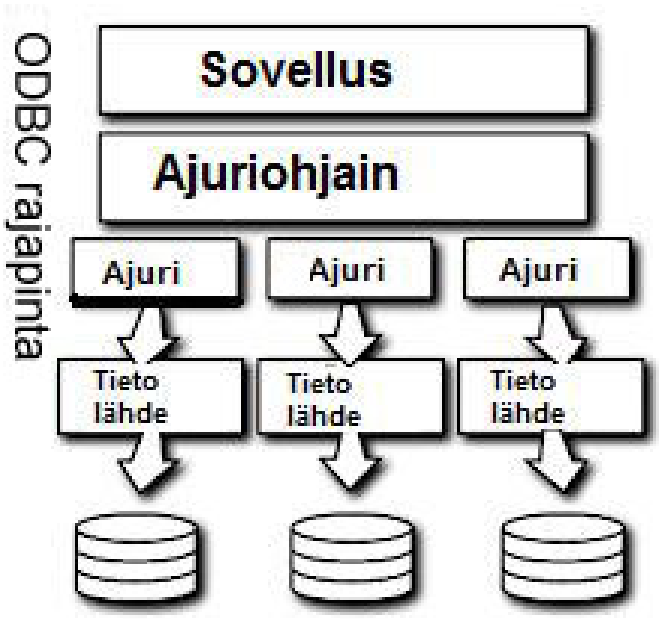
- Tietolähde muodostuu tietokannan hallintajärjestelmästä, siitä käyttöjärjestelmästä, millä tietokannan hallintajärjestelmä sijaitsee ja mahdollisesta tietoverkosta, jota on käytetty tietokannan hallintajärjestelmään pääsemiseksi.

Ajuriohjain ja ajuri näyttävät sovellukseen päin yhdeltä yksiköltä, joka käsittelee ODBC-funktiokutsut (kuva 1). ODBC määrittelee kaksi ajurityyppiä:

- Yksitasoinen (single-tier) ajuri käsittelee sekä ODBC-kutsut että SQL-lauseet.

- Useampitasoinen (multipe-tier) ajuri käsittelee ODBC-kutsut ja välittää SQL-lauseet tietolähteelle.

Yksi systeemi voi sisältää molemman tyyppisiä kokoonpanoja.

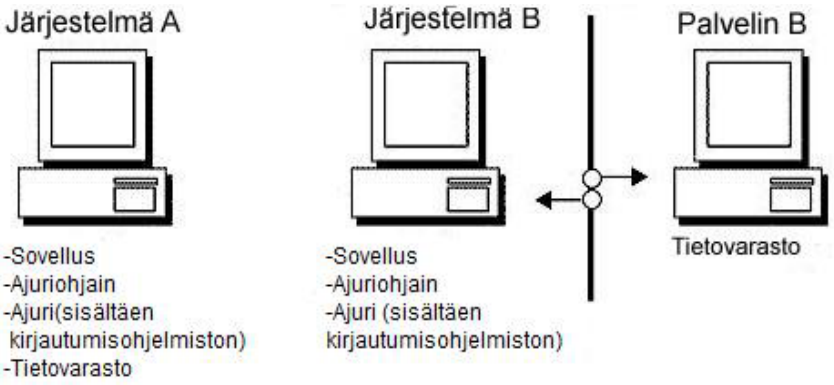


Kuva 1 Tietokantaan kytkeytyminen ODBC-yhteydellä [11]

3.1.1. Yksitasoinen kokoonpano

Yksitasoisessa toteutuksessa ajuri käsittelee tietokantaa suoraan. Ajuri käsittelee SQL-lauseet ja hakee tiedot tietokannasta. Esimerkki yksitasoisesta kokoonpanosta on ajuri, joka käsittelee sellaisia työpöytätietokantajärjestelmiä, kuten DBASE, Paradox, Fox Pro jne.

Kuva 2 esittelee kaksi yksitasoisen kokoonpanon tyyppiä. Toinen on yksinään toimiva ja toinen käyttää verkkoa. Yksitasoisen ajurin verkkoympäristöversion tiedonhakuohjelmisto sijaitsee henkilökohtaisella tietokoneella, joten kyselyiden käsittelyvastuu on asiakaskoneella.



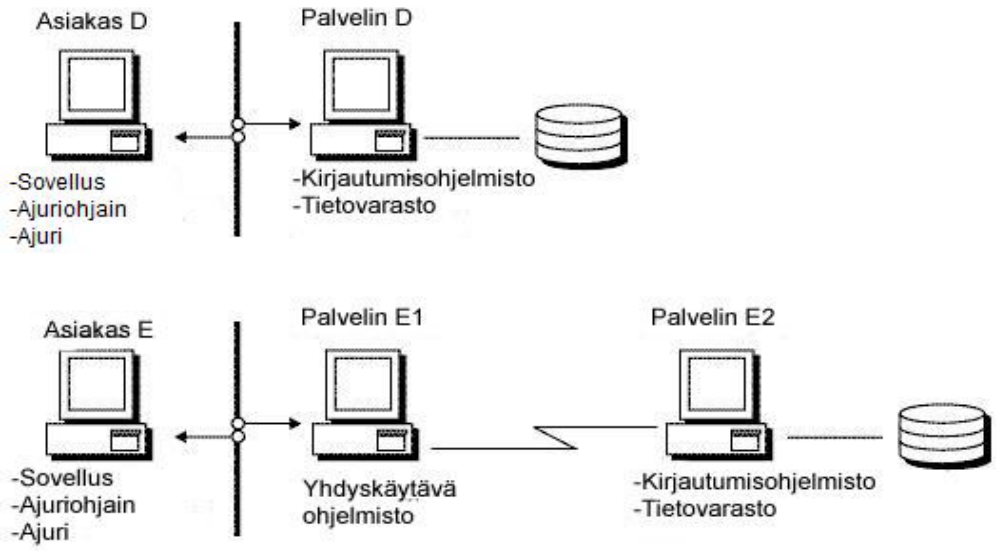
Kuva 2 Yksitasoinen kokoonpano [11]

3.1.2. Useampitasoinen kokoonpano

Useampitasoisessa kokoonpanossa ajuri lähettää SQL-pyynnöt palvelimelle, joka käsittelee ne [11]. Sovellus, ajuri ja ajuriohjain sijaitsevat yhdessä järjestelmässä, jota tyypillisesti kutsutaan asiakkaaksi. Tietokanta ja ohjelmisto, joka kontrolloi tietokantaan pääsyä sijaitsevat tyypillisesti toisessa järjestelmässä, jota kutsutaan palvelimeksi. Tämä tarkoittaa, että kyselyiden käsittelyvastuu on palvelimella.

Yksi useampitasoisen kokoonpanon muunnelma on yhdyskäytävä (gateway) -arkkitehtuuri, missä ajuri välittää SQL-pyynnöt yhdyskäytävälle prosessoitavaksi. Sieltä pyynnöt ohjataan tietolähteelle. Yhdyskäytävä voi tässä skenaariossa olla osa laitteistoa tai tiedonhakuohjelmisto matalan tason rajapintana etätietokantaan.

Kuva 3 esittelee kaksi useampitasoisen kokoonpanon tyyppiä. Sovelluksen kannalta molemmat kokoonpanot ovat identtisiä.



Kuva 3 Useampitasoinen kokoonpano [11]

Useat yritykset ovat kehittäneet ODBC-ajureita, jotka tarjoavat yhteyden useisiin tietokantoihin. Kaupallisia ODBC-ajureita on tarjolla mm. seuraavilla yrityksillä [24]:

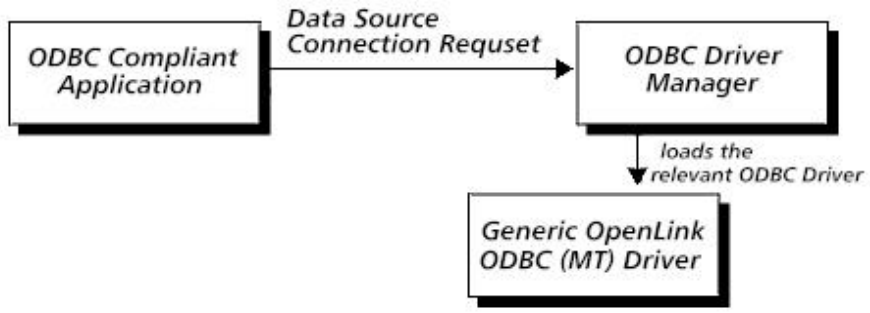
- OpenLink
- SyBase
- Simba Technologies
- Visigenic Software, Inc.
- Intersolv
- PostgreSQL
- Cincom Systems, Inc. (SUPRA SQL-relaatiotietokannalleen)
- AugSoft (Macintosh, Win32 ja Linux yhteys useisiin tietokantoihin)

MySQL ja PostgreSQL tietokantoihin on saatavilla myös ilmaisia ODBC-ajureita.

Seuraavassa on esitelty tarkemmin OpenLink ODBC-ajurien toimintaa [11]. Muiden valmistajien ajurien toiminta on samankaltaista.

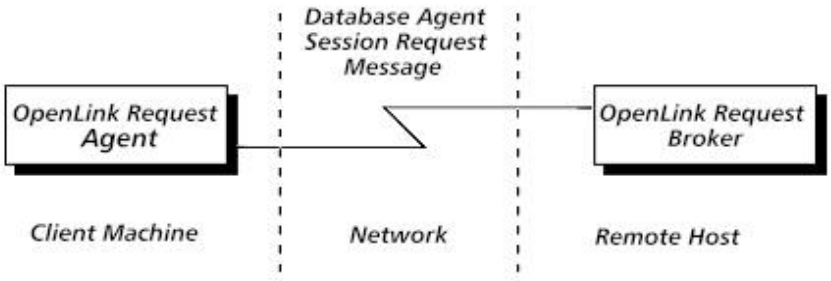
3.1.3. Yhteyden muodostaminen tietolähteeseen

Kun ODBC:tä noudattava sovellus kirjautuu ODBC-tietolähteeseen, se tekee sen ODBC-ajurin avulla. Tämän toiminnon lopputuloksena tietolähteen yhteyspyyntö välitetään ajurille (Generic OpenLink ODBC (MT) Driver) kuvan 4 mukaisesti.



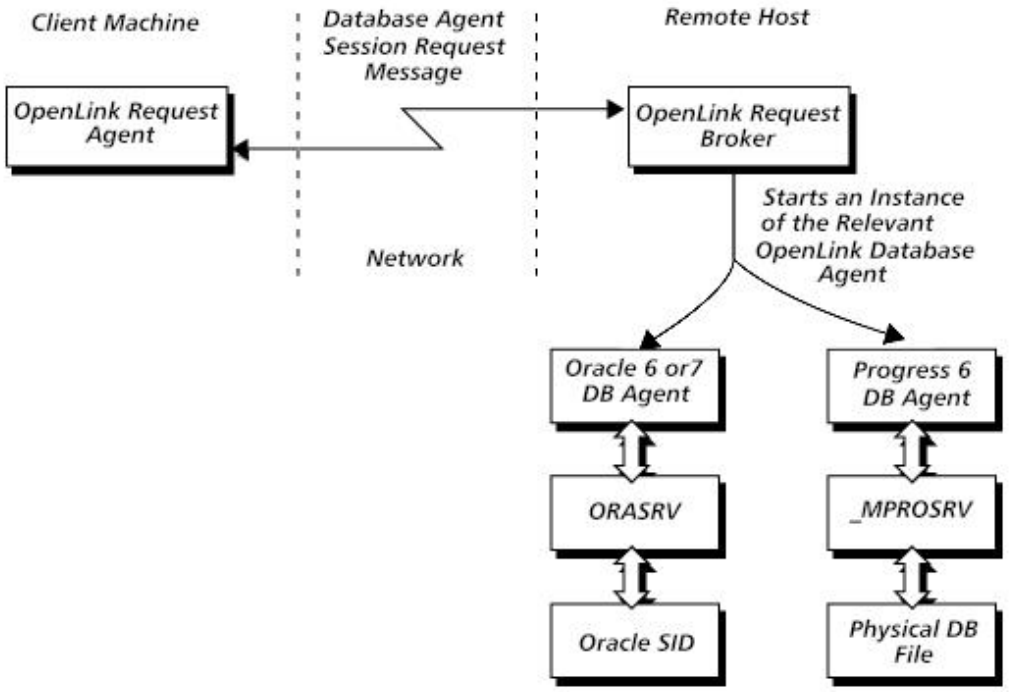
Kuva 4 Yhteyden muodostaminen [11]

Ajuri kokoaa viestin liittämällä ODBC-ajuriohjaimen muodostamaan ODBC-viestiin lisäinformaatiota kuten tietokantatyypin, määränpäänä olevan isäntäkoneen, lähdekoneen tyypin ja lähdesovelluksen nimen. Pyyntöagentti (OpenLink Request Agent) muuntaa tämän viestin tietokanta-agentin (OpenLink Database Agent) istuntopyynnöksi ja välittää sen sitten verkon läpi pyynnön välittäjälle (OpenLink Request Broker), joka sijaitsee isäntäkoneella (kuva 5).



Kuva 5 Yhteyspyynnön välitys [11]

Pyynnön välittäjä tulkitsee viestin ja ratkaisee tarvitseeko sen luoda yhteys pyyntöagentin ja olemassa olevan tietokanta-agentin välille (kuva 6). Se kopioi jo olemassa olevan tietokanta-agentti-instanssin tai luo uuden sille tietokantatyypille, joka on määritelty viestissä. Tämä prosessi myös liittää tietokanta-agentin juuri siihen tietokantatyyppiin, johon ODBC:tä noudattava sovellus halusikin luoda yhteyden.



Kuva 6 Yhteyspyynnön tulkinta [11]

3.1.4. Viestin välitys

Kun yhteys pyyntöagentin ja tietokanta-agentin välillä on muodostettu, kaikki seuraavat ODBC-viestit välitetään työasematietokoneen sovellukselta ODBC-ajurin kautta pyyntöagentin avulla kyseessä olevalle tietokanta-agentti-instanssille. Tietokanta-agentin vastaukset välitetään pyyntöagentin kautta ODBC-ajurille ja viimein myös työasemakoneen sovellukselle. Suurten olioiden tapauksessa sekä pyynnöt että vastaukset voivat olla kooltaan suuria asettaen omia vaatimuksiaan sekä pyyntöagentin että sen välittäjän suorituskyvylle.

Tässä vaiheessa ODBC-ajurinohjain ja pyynnön välittäjä menevät passiiviseen tilaan kunnes joko toinen ODBC:tä noudattava työasemasovellus haluaa ODBC-yhteyden tai lisää tietolähdeyhteyspyyntöjä saadaan jo olemassa olevista ODBC-istunnoista.

3.1.5. Arvioita ODBC:stä

Ensimmäiset ODBC-ajurit olivat tyypillisesti yksitasoista kokoonpanoa noudattavia ja käyttivät upotettua SQL:ää tai CLI:tä (Call Level Interface). Ongelmana on, että sovellukset ja ajurit ovat täysin riippuvaisia niiden samojen valmistajien yhteystuotteista, jotka mahdollistavat yhteydet etätietokantoihin. Nämä tuotteet eivät yleensä ole optimoitu ODBC:n kannalta ja joissain tapauksissa eivät edes muodosta todellista asiakas-palvelin-protokollaa. Yhteysteknologia on siis joka tapauksessa tietokantariippuvaista, vaikkakin jotkut toimittajat tarjoavat yhdyskäytäviä (gateway) toisiin tietokantoihin. ODBC-standardin alkuperäinen ajatuksena käyttäjien ja kehittäjien kannalta oli pitää pelkästään tietokantatoimittajien velvollisuutena kehittää ODBC-ajureita, mutta nykyisin on tarjolla myös kaupallisia ajureita. Kaikki ajurit ovat hieman erilaisia ja niissä on eroja myös niiden noudattamien standarditasojen määrän sekä ajonaikaisen suorituskyvyn ja joustavuuden suhteen [11].

Suurten olioiden tuen osalta jotkin ODBC-ajurit saattavat olla jopa puutteellisia ja varsinkin kilpailijoiden tuotteiden ajurit eivät ole ensisijalla tärkeysjärjestyksessä. Esimerkiksi marraskuussa 2007 ei Microsoftilla vielä ollut suuria olioita tukevaa ODBC-ajuria Oracle-tietokannalle [16].

3.1.6. Esimerkkejä

Esimerkiksi ODBC-tietolähde luodaan Windows XP:ssä ODBC-ajurien asennuksen jälkeen seuraavasti [18] :

- Valitaan ohjauspaneelin lisäasetusten Data Sources (ODBC).
- Käyttäjätietolähteet (DSN) välilehdellä painetaan Lisää-painiketta.
- Avautuvasta ikkunasta selataan rekisteröitävä ajuri ja painetaan Valmis-painiketta.

- Seuraavaksi määritellään tietolähteelle nimi. Tietokantaan voi ottaa yhteyden käyttämällä joko Tietolähdenimeä (Data Source Name, DSN) tai TNS-palvelunimeä (TNS Service Name). Kun tarvittavat tiedot on annettu, painetaan Ok-painiketta.

Alla oleva PHP-koodinpätkä näyttää, kuinka ODBC-tietokantayhteys luodaan, tietoja haetaan ja tulokset näytetään HTML-taulukossa [29]:

```
<html>
<body>
<?php
$conn=odbc_connect('northwind','','');
if (!$conn)
   {exit("Connection Failed: " . $conn);}
$sql="SELECT * FROM customers";
$rs=odbc_exec($conn,$sql);
if (!$rs)
   {exit("Error in SQL");}
echo "<table><tr>";
echo "<th>Companyname</th>";
echo "<th>Contactname</th></tr>";
while (odbc_fetch_row($rs))
{
   $compname=odbc_result($rs,"CompanyName");
   $conname=odbc_result($rs,"ContactName");
   echo "<tr><td>$compname</td>";
   echo "<td>$conname</td></tr>";
}
odbc_close($conn);
echo "</table>";
?>
</body>
</html>
```

Tässä tietokantayhteys luodaan odbc_connect-nimisellä php-funktiolla, jonka syntaksi on seuraava [20]:

```
resource odbc_connect ( string $dsn , string $user , string $password
[, int $cursor_type ] )
```

Funktio palauttaa joko yhteysidentifikaation (connection_id) tai nollan virheen sattuessa. Pysyvää yhteyttä varten on oma odbc_pconnect()-funktio.

3.2. JDBC

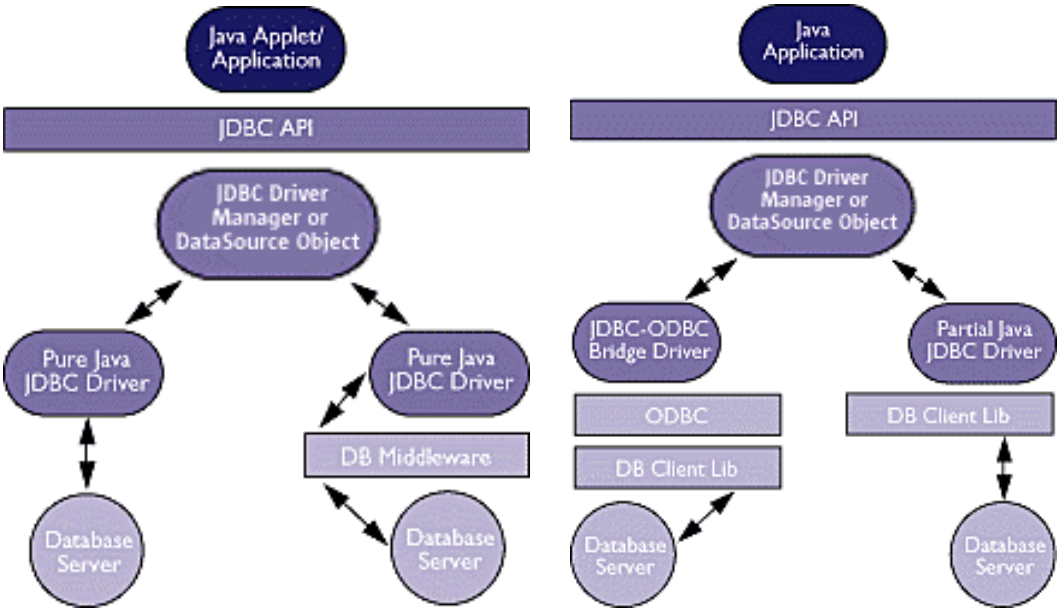
JDBC (Java Database Connectivity) on Java API (Application Programming Interface**)**, joka mahdollistaa SQL-lauseiden suorittamisen Java-sovelluksille. JDBC:n kehitti JavaSoft, joka on Sun Microsystemsin tytäryhtiö. JDBC on samankaltainen kuin ODBC, mutta se on suunniteltu erityisesti Java-ohjelmille, kun taas ODBC on kieliriippumaton [31]. JDBC API mahdollistaa kolme asiaa: muodostaa yhteyden tietokantaan tai mihin tahansa taulukkotietolähteeseen, lähettää SQL-lauseita ja käsitellä tuloksia.

3.2.1. Ajurityypit

JDBC API sisältää kaksi isompaa rajapintaa: ensinnäkin sovellusten kirjoittajille ja toinen on matalamman tason JDBC-ajuri [26]. JDBC-ajurit voidaan jakaa neljään luokkaan. Sovellukset ja appletit voivat saada yhteyden tietokantaan JDBC API:n avulla käyttämällä puhtaita Java-teknologian perusajureita tai käyttämällä ODBC-ajureita ja olemassa olevia tietokanta-asiakaskirjastoja, kuten kuvassa 7 näytetään.

Kuvassa 7 on kuvattu ajurityypit:

- puhdas Java JDBC -ajuri (tyyppi 4)
- puhdas Java JDBC -ajuri tietokannan väliohjelmistolle (tyyppi 3)
- JDBC-ODBC-silta ja ODBC-ajuri (tyyppi 1)
- osittainen Java JDBC -ajuri (tyyppi 2).



Kuva 7 JDBC-ajuriluokat [26]

Tyypin 4 ajuri kääntää JDBC-kutsut tietoverkon protokolliksi, joita tietokanta voi käyttää suoraan. Tarjoaa ratkaisun intranet-yhteyksille.

Tyypin 3 ajuri taas kääntää JDBC-kutsut väliohjelmiston tarjoajien protokolliksi, jotka väliohjelmistopalvelin sitten kääntää tietokannan hallintajärjestelmän protokolliksi.

Tyypin 1 yhdistelmä tarjoaa JDBC-yhteyden ODBC-ajurien kautta. ODBC-binaarikoodi ja monessa tapauksessa myös tietokannan asiakaskoodi, on ladattava jokaiselle asiakaskoneelle, joka käyttää siltaa. Tämä ajuri on sopiva koekäyttöön ja tilanteisiin, joissa mitään muuta ajuria ei ole saatavilla.

Tyypin 2 ajuri kääntää JDBC-kutsut asiakas API-kutsuiksi mm. Oracle, Sybase, Informix, DB2, tai muille tietokannoille. Tämän tyyppisen ajurin käyttö vaatii myös binaarikoodin lataamista asiakaskoneelle, kuten edellinen silta-ajurikin.

Esimerkiksi Oraclella on tarjolla seuraavat JDBC-ajurit [13]:

- Thin-ajuri, tyypin 4 ajuri asiakaspuolen käyttöön

- OCI-ajuri, tyypin 2 Oracle-alustariippuvainen ajuri

- palvelinpuolen Thin-ajuri, joka tarjoaa saman toiminnallisuuden kuin asiakaspuolen Thin-ajurikin, mutta on tarkoitettu Oracle-palvelimella ajettavalle koodille etäpalvelimien kutsumiseksi

- palvelinpuolen sisäinen ajuri Oracle-palvelimella ajettavalle koodille Oracle-istunnon sisällä

3.2.2. JDBC:n etuja ja ominaisuuksia

Sun Microsystems [26] kertoo JDBC:n eduista seuraavaa:

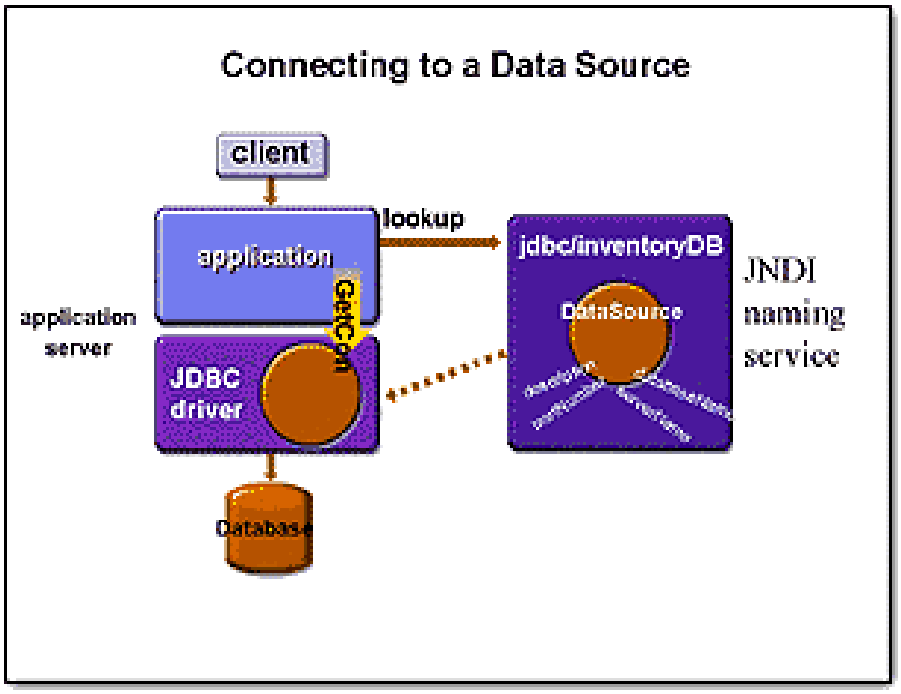
- JDBC-teknologialla yritykset eivät ole sidoksissa mihinkään yksinoikeusarkkitehtuuriin ja voivat jatkaa omien jo asennettujen tietokantojensa käyttöä ja saada tietoa helposti, vaikka se olisi tallennettu erilaisiin tietokantajärjestelmiin.

- Java API:n ja JDBC API:n yhdistelmä tekee sovelluskehityksen helpoksi ja taloudelliseksi. JDBC piilottaa monien tietokantatehtävien monimutkaisuuden tehden suurimman osan raskaasta työstä ohjelmoijan puolesta. JDBC API on helppo oppia ja käyttää ja halpa ylläpitää.

- JBDC API ei vaadi mitään konfigurointia asiakkaan puolelta. Java-kielellä kirjoitetulla ajurilla kaiken tiedon, jota yhteyden muodostamiseen tarvitaan, määrittelee joko JDBC URL tai JNDI -yhteydellä rekisteröity tietolähdeobjekti (kuva 8).

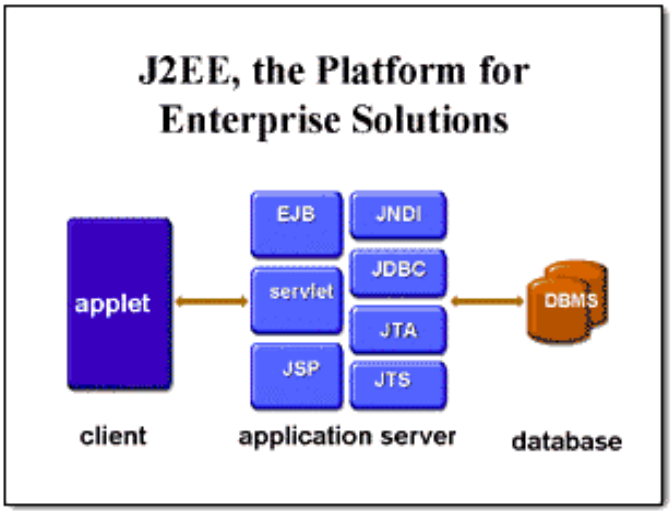
JDBC:n avainominaisuuksia ovat [26]:

- JDBC API tarjoaa pääsyn metadataan, joka taas mahdollistaa sellaisten sovellusten kehityksen, joiden on ymmärrettävä tietyn tietokannan perusominaisuudet.



Kuva 8. Tietolähteeseen yhdistäminen JDBC-ajurilla [26].

- Osana Java 2 -alustaa JDBC API on saatavilla missä tahansa, missä alustakin (kuva 9). Tämä tarkoittaa sitä, että kerran kirjoitetut tietokantasovellukset ovat helposti siirrettävissä.



Kuva 9. Esimerkki J2EE-pohjaisesta arkkitehtuurista, joka sisältää JDBC-asennuksen [26].

3.2.3. Esimerkkejä

Seuraavana on esimerkki, jossa muodostetaan yhteys tietokantaan Oracle JDBC Thin-ajurilla, käyttäen ns. palvelinpuolen sisäistä ajuria, tehdään tietokantakysely ja käsitellään tulosjoukkoa [13]:

```
import java.sql.*;
import java.math.*;
import java.io.*;
import java.awt.*;
import oracle.jdbc.*;
import oracle.jdbc.pool.OracleDataSource;

class JdbcTest {
   public static void main (String args []) throws SQLException {
       // Create DataSource and connect to the local database
       OracleDataSource ods = new OracleDataSource();
       ods.setURL("jdbc:oracle:thin:@//myhost:1521/orcl");
       ods.setUser("scott");
       ods.setPassword("tiger");
       Connection conn = ods.getConnection();

       // Query the employee names
       Statement stmt = conn.createStatement ();
       ResultSet rset = stmt.executeQuery ("SELECT ename FROM emp");
       // Print the name out
       while (rset.next ())
         System.out.println (rset.getString (1));

       //close the result set, statement, and the connection
       rset.close();
       stmt.close();
       conn.close();
   }
}
```

Oraclen OLAP (On-Line Analytic Processing) API käyttää myös JDBC:tä tietokantayhteyksissä. Yhteyden muodostaminen tapahtuu seuraavasti [19]:

- Ensin ladataan JDBC-ajuri ja rekisteröidään se JDBC DriverManagerilla:

```
Class.forName("oracle.jdbc.driver.OracleDriver")
```

- Haetaan JDBC Connection-olio DriverManagerilta:

```
String url = "jdbc:oracle:thin:@lab1:1521:orcl";
String user = "hepburn";
String password = "tracey";
oracle.jdbc.OracleConnection conn = (oracle.jdbc.OracleConnection)
java.sql.DriverManager.getConnection(url, user, password);
```

- Muodostetaan TransactionProvider:

```
ExpressTransactionProvider tp = new ExpressTransactionProvider();
```

- Muodostetaan DataProvider:

```
ExpressDataProvider dp = new ExpressDataProvider(conn, tp);
dp.initialize();
```

4. Fyysisiä tallennusjärjestelmiä

Suurten olioiden käsittely on merkittävä tekijä monille epätavanomaisille tietokantasovelluksille [1]. Tällaisia sovelluksia ovat mm. maantieteelliset sovellukset, kuva-analyysit, tietokoneavusteinen suunnittelu ja toimistoautomaatio dokumenttien käsittelyineen, julkaisutoimintoineen ja multimediaesityksineen. Nämä tarvitsevat tehokasta muistia ja suurten objektien käsittelyä. Suurten objektien tehokas käsittely on kuitenkin tärkeää missä tahansa objektiperusteisissa olio- ja relaatiotietokannoissa.

Sellaisen tietokantajärjestelmän näkökulmasta, joka tallettaa objektit sivuille (page), objekti on pieni, jos se mahtuu kokonaan yhdelle sivulle, muutoin objekti on iso. Ihanne tapauksessa rajoittamattoman kokoinen virtuaalinen objekti (käytettävissä olevan muistin rajoissa) pitää käsitellä ja tallettaa siten, että sen sisäinen pirstoutuminen minimoidaan. Toisin sanoen muistin hyödyntämisen tulisi olla lähes sata prosenttista. Suurten objektien tietokantaan luomisen kustannusten vähentämiseksi suuren levylohkomäärän varaamisen ja vapauttamisen vaatima kustannus pitäisi olla mahdollisimman pieni, ihanne tapauksessa yksi levyhaku tapahtumaa kohti riippumatta tarvittavan tilan koosta.

Koska objekti voi todellakin olla hyvin suuri, seuraavat objektiin tai sen osaan kohdistuvat toimenpiteet pitäisi olla mahdollisia ja taloudellisesti toteutettuja: objektin sisäisen tavujonon luku ja korvaaminen, tavujen lisääminen ja poistaminen tietystä kohdasta objektia (tietyn bittimäärän päästä objektin alusta) ja viimeiseksi tavujen lisääminen objektin loppuun. Näistä ylläpitotoiminnot voivat joko kasvattaa tai pienentää objektin kokoa.

On ainakin kaksi syytä miksi tällaiset suurten objektien osaan kohdistuvat toimenpiteet ovat tärkeitä. Ensinnäkin koko ison objektin lukeminen tai päivitys voi olla epäkäytännöllistä tai mahdotonta ohjelmiston tai laitteiston rajoitteista johtuen, mm. ohjelman muistiavaruus voi olla objektin kokoa pienempi. Yleensäkin on epätodennäköistä ja joissain tapauksissa mahdotonta luoda hyvin suurta objektia yhdellä kertaa, todennäköisempää on, että pienempiä, äärellisiä tavumääriä lisätään objektin loppuun perätysten.

Toiseksi sovellus, joka käsittelee suurta objektia saattaa tarvita kerrallaan vain osan siitä. Esimerkiksi objekti voidaan haluta hakea perätysten vain pieni osa kerrallaan eikä koko

objektia samalla kertaa, tällaisia ovat mm. digitaaliset äänitallenteet ja elokuvat. Samoin lisäykset ja poistot pitkään suurena objektina talletettuun listaan voivat kohdistua mihin tahansa kohtaan listaa ja multimediasovelluksissa kuvia ja elokuvissa ruutuja voidaan lisätä tai poistaa mistä kohdasta tahansa.

Sekä haja- (random) että peräkkäis- (sequential) haun pitäisi siis saavuttaa hyvä suorituskyky. Hajahaku on tehokasta, jos tietyn tavun paikan hakeminen objektin sisältä ei ole riippuvainen objektin koosta. Tämä tarkoittaa sitä, että sivujen linkitykseen perustuvat lineaaristen listojen tyyliset ratkaisut eivät sovellu suurten objektien talletukseen. Peräkkäishaku on tehokasta, jos suurten objektien (tai sen suurten osien) haun I/O-nopeudet ovat lähellä siirtonopeuksia. Jotta tämä olisi mahdollista, olisi levyhakujen viiveet minimoitava, mikä puolestaan johtaa siihen, että levytila pitäisi varata suurina yksiköinä mieluummin kuin peräkkäisinä lohkoina. Fyysinen läheisyys on myös eduksi. Sellaisissa tietokoneympäristöissä, joissa tietoa siirretään asiakas- ja palvelinkoneiden välillä, on kokeellista tietoa siitä, että on tärkeää voida siirtää suuria tietomääriä kerrallaan ja siten yleensäkin vähentää tietojen siirtelyä. Päivitysten suhteen pienten muutosten pitäisi vaikuttaa vain vähän. Esimerkiksi muutamien tavujen lisäämisen objektin keskelle ei pitäisi aiheuttaa koko objektin uudelleen organisointia.

Seuraavassa esitellään muutamia suurten olioiden tallentamiseen kehitettyjä järjestelmiä ja menetelmiä, joista osa on ollut käytössä jo kauan. Aivan uusimmista on hankala saada tietoa, sillä monille tietokantavalmistajille ne saattavat olla liikesalaisuuksia.

4.1. System R

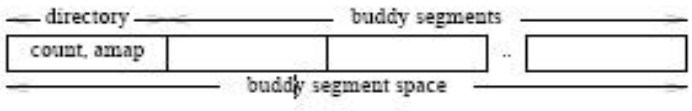
Suurten objektien käsittely alkoi relaatiomallien yhteydessä. System R tuki pitkiä kenttiä, joiden pituus oli jopa 32 kilotavua [1]. Pitkä kenttä toteutettiin pienten segmenttien lineaarisena linkitettynä listana, kunkin segmentin ollessa 255 tavua pitkä. Pitkän kentän tunniste osoitti listan alkuun. Osittaisia lukuja ja päivityksiä ei tuettu. Tämän ratkaisun laajennuksena oli järjestelmä, jossa pitkät kentät talletettiin 4 kilotavun sivujen jonoina, joille tuettiin osittaista lukua ja päivitystä. Pitkän kentän tunniste on taulukko, joka sisältää kunkin sivun osoitteen ja pituuden. Pitkän kentän maksimikoko oli noin 2 Gigatavua.

4.2. Wisconsin Storage System

Wisconsin Storage System:ssä (WiSS) suuret objektit talletetaan siivuiksi (slices) kutsuttuihin muistisegmentteihin. Hakemisto näihin siivuihin tallennetaan normaalina pienenä tiedostona, jonka koko voi kasvaa suunnilleen sivun kokoiseksi. Se sisältää osoitteen ja jokaisen siivun koon. Kukin siivu voi olla korkeintaan yhden sivun pituinen. Täten 4 kilotavun siivuin hakemistossa voi olla noin 400 siivua, joka määrää objektin maksimikooksi noin 1,6 megatavua. WiSS on ollut O2-tietokannan muistinhallintamenetelmänä [1]. O2 on ODMG:n mukainen (Object Database Management Group) kaupallinen oliotietokantatuote [10].

4.3. Buddy-järjestelmä

Buddy-järjestelmä (buddy system) hallitsee joukkoa fyysisesti peräkkäisiä, suuria, kiinteämittaisia levylohkoja, joita kutsutaan segmenttialueiksi (buddy segment spaces) [1]. Segmentit ovat vaihtuvamittaisia jaksoja fyysisesti peräkkäisiä levysivuja, jotka on otettu yhdestä segmenttialueesta. Sisäisesti segmenttejä käsitellään ikään kuin niiden koot olisivat kakkosen kokonaislukupotensseja. Kooltaan 2^t segmentin sanotaan olevan tyyppiä *t*. Jokainen segmenttialue sisältää yksisivuisen hakemiston, joka koostuu lukumäärätaulukosta *count* ja sivujen varauskartasta *amap* (kuva 10). Lukumäärätaulukko osoittaa jokaisen mahdollisen segmenttityypin vapaiden segmenttien määrää. Jos maksimi segmenttityyppi on *k*, niin taulukossa on *k*+1 merkintää 0-*k* (sisältäen *k*:n) ja *count*[*t*] on tyypin *t* (siis kooltaan 2^t) vapaiden segmenttien lukumäärä. Varaustaulukko osoittaa segmenttialueen jokaisen segmentin tilan (vapaa tai varattu) ja tyypin. Mitään kontrollitietoja ei talleteta segmentteihin (esim. samantyyppisiä vapaita segmenttejä ei linkitetä). Koko segmenttien varaus- ja vapautusprosessi tehdään vain hakemistosivulla. Täten niille tietokannoille, jotka mahtuvat yhdelle yhdistelyalueelle, tarvitaan korkeintaan yksi levyhaku lohkon varaamis- tai vapautuspyynnölle riippumatta segmentin koosta.

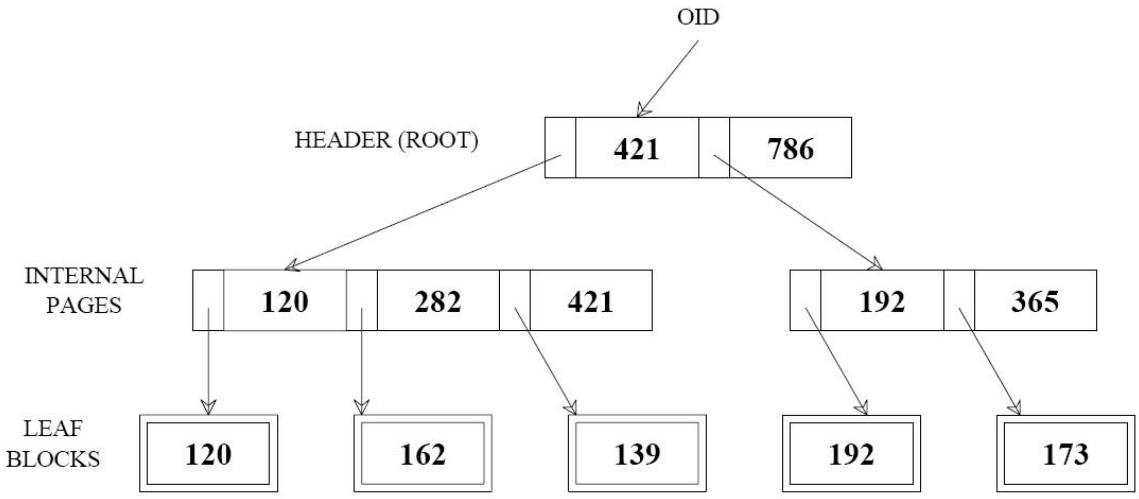


Kuva 10 Buddy-segmenttialueiden järjestys

Koska hakemisto on aina yhden sivun kokoinen, niin segmenttialueen maksimi segmenttikoko riippuu sivun koosta. Tietylle sivukoolle PS maksimi segmenttikoko on 2PS sivua. Esim. 4 kilotavun levysivuille suurin segmenttikoko, jota voidaan ylläpitää on 2^{13} (32 Mb), segmenttityypeillä 0-$\log_2$(2x4096)=13. Jos oletamme, että lukumäärätaulukon alkio vie 2 tavua, niin varaustaulukko voi olla enintään 4096-2*14 =4068 tavua pitkä. Tällöin voidaan ylläpitää yhdistelysegmenttialuetta, joka on korkeintaan 4068x4=16 272 sivua (n. 63,5 Mb).

4.4. Exodus

Exodus muistijärjestelmä käsittelee rajoittamattoman kokoisia suuria objekteja tallentamalla ne tietosivuille, jotka on indeksoitu B-puun tyylisellä rakenteella, jossa solmuihin tallennetaan objektin tavumääriä [2]. Esimerkiksi juurisolmun vasen lapsi sisältää tavut 1-421, kuten kuvassa 11 on esitetty.



Kuva 11 Esimerkki Exoduksen suuresta objektista [2]

Tämä on dynaaminen rakenne, joka tukee tavujen lisäystä ja poistoa. Parantaakseen suurten tavumäärien lukemisen suorituskykyä, asiakkaat voivat asettaa kaikkien suurten objektien datasivujen koon tiedoston sisällä joksikin kiinteäksi levyblokkien määräksi. Tämä mekanismi ei kuitenkaan auta sovelluksia, jotka haluavat sananaikaisesti optimoida sekä hakuaikoja että muistin käyttöä, koska lehtisivun koolla on päinvastaiset vaikutukset näihin toimintoihin. Suuret sivukoot tuhlaavat paljon tilaa osittain käytettyjen sivujen lopusta, vaikka

tarjoavat hyvät hakuajat. Pienet sivukoot taas saavat aikaan hyvän muistinkäytön, mutta vaativat monia hakuja.

SHORE (Scalable Heterogeneous Object Repository) on Exoduksen laajennus [25]. PREDATOR-tietokantajärjestelmä perustuu SHORE-järjestelmään.

4.5. EOS

EOS (Experimental Object Store) on suurten objektien kokeellinen tallennusjärjestelmä [1]. Sen pääperiaatteita ovat:

1. Tuki rajoittamattoman kokoisille objekteille (fyysisen muistin rajoissa).

2. Tuki osittaisille operaatioille eli tavujen lisäämiselle loppuun, tavujonojen lukemiselle ja korvaamiselle ja tavujen lisäämiselle tai poistamiselle satunnaisesta paikasta objektia.

3. Yllämainittujen osittaisten operaatioiden aiheuttamien kustannusten on oltava riippuvaisia kyseessä olevasta tavumäärästä eikä objektin koosta. Erityisesti levyhakuja minimoidaan siten, että I/O-nopeus on lähellä siirtonopeutta.

4. Suuren fyysisesti jatkuvan levytilan varaamisen tulisi olla nopeaa, ihanteellisesti yksi levyhaku riippumatta tarvittavan tilan määrästä tai tietokannan koosta.

5. Muistin hyödyntämisen pitäisi olla hyvin lähellä sataa prosenttia.

6. Suurten objektien tulisi olla suojattuja siirto- ja systeemihäiriöiltä.

Levytilan varaus perustuu buddy-systeemiin (ks. kohta 4.3), joka suorittaa vaihtelevan kokoisten levylohkojen varaamisen ja vapauttamisen nopeasti minimaalisin I/O- ja CPU-kustannuksin.

Kun suuri objekti luodaan, niin se talletetaan suurten, vaihtelevan kokoisten, fyysisesti jatkuvien levysegmenttien jonoksi. Sitten, kun suurelle objektille on suoritettu tavujonojen lisäyksiä ja poistoja, sen segmentit ovat voineet jakautua pienemmiksi. Täten ne segmentit,

jotka muodostavat suuren objektin, voivat vaihdella suunnattomasti kooltaan. B-puun tapainen rakenne (samantapainen kuin kohdassa 4.4) luetteloi tavujen paikat objektin sisällä

Suuret objektit talletetaan siis tulkitsemattomina bittijonoina buddy-järjestelmästä otettuihin vaihtelevankokoisten segmenttien sarjoiksi. Missään segmentissä ei ole aukkoja, sillä kaikkien segmenttien tulee olla täysiä, paitsi viimeinen voi olla vain osittain täytetty. Segmentit osoitetaan sitten paikkasidonnaisella puulla, joka on B-puumainen rakenne, jossa avaimina ovat objektin tavujen sijainnit segmentin sisällä. Mitään kontrollitietoja ei talleteta datasegmentteihin, sillä kaikki niiden hakemiseen tarvittava tieto pidetään vanhempihakemistosivuilla. Juurta lukuun ottamatta jokainen puun sisäinen solmu on talletettu yhdelle sivulle. Kuten B-puissa yleensä sisäisten solmuparien on oltava vähintään puolittain täysiä.

4.6. SLOB

SLOB (Structured Large Objects) on yksi uusimmista suurten olioiden tallentamiseen liittyvistä termeistä [6]. Se esiteltiin vuonna 2006 ja on tarkoitettu Oraclen versiolle 10g.

Esimerkiksi biologiset, genomi- (organismin koko perintöaines), biolääketieteelliset, video-, valokuva-, paikkatieto- ja multimediatietokannat sisältävät monimutkaisia, hyvinkin rakenteellisia ja vaihtelevan mittaisia tietoja. LOB-tyyppisiä tietotyyppejä taas ei ole perinteisesti suunniteltu tallentamaan rakenteellista tietoa ja ne eivät siten sisällä metodeja, joilla haettaisiin monimutkaisten objektien sisäisiä rakenneosia. LOB:ien päivittäminen tapahtuu yleensä tavujen päälle kirjoittamalla ja siirtämällä tietoa. Esimerkiksi keskellä videota olevan kahden minuutin segmentin korvaaminen kolmen minuutin segmentillä vaatii kaiken päivityspaikan jäljessä olevan tiedon siirtämistä, jotta uudelle osalle tulisi tilaa. Kaiken lisäksi LOB:eilla on tyypillisesti eri tietokannan hallintajärjestelmissä erilaiset käyttöliittymät erilaisine toimintoineen, joten tietokantariippumattomien sovellusten kehittäminen on hankalaa. Monissa tietokannoissa on myös tietyn ongelman ratkaisemiseksi räätälöityjä tutkimusprototyyppejä, joiden siirtäminen kaupallisiin versioihin voisi olla hyödyllistä.

Näitä ongelmia helpottamaan on Floridan yliopistossa kehitetty uusi SLOB-tietotyyppi, joka perustuu kahteen käsitteeseen: rakenneindeksiin (structure index) ja peräkkäisindeksiin (sequence index).

Rakenneindeksi auttaa säilyttämään sovellusobjektin rakenteellisen kokoonpanon rakenteettomassa LOB-muistissa. Se koostuu kahdesta osasta, joista ensimmäinen on tiedon rakenteen kuvaus ja toinen varsinaiset tiedot. Rakenteellista komponenttia käytetään viittauksena haettaessa tiedon rakenteellista hierarkiaa. Se käsittelee rakenteellista objektia kuin se koostuisi joukosta vaihtelevan mittaisia osaobjekteja, missä kukin osaobjekti voi olla joko rakenteellinen objekti tai perusobjekti (objekti, jolla ei ole rakennetta). Kunkin rakenteellisen objektin sisällä sen osaobjektit sijaitsevat peräkkäisesti numeroiden sijoitettuina. Rakennehierarkian lehtisolut sisältävät perusobjektit.

Peräkkäisindeksi taas tukee satunnaisia päivityksiä LOB-ympäristössä ja sitä käytetään tietoa päivitettäessä. Peräkkäisindeksi perustuu siihen, että uusi tieto talletetaan fyysisesti LOB:in loppuun ja tietojen oikea järjestys säilytetään peräkkäisindeksillä. Peräkkäisindeksi on toteutettu käyttämällä järjestettyä listaa, jonka alkioina ovat tavujoukko osoitteita. Molemmat käsitteet vaativat LOB-sovellukselta vain minimaalista toiminnallisuutta.

Järjestelmä sisältää menetelmät, joilla SLOB:eja voidaan luoda, hakea ja käsitellä (lisätä, poistaa, päivittää).

5. Tiedostojärjestelmä vai tietokanta?

Microsoft ja Berkleyn yliopisto ovat tehneet vertailun suurten olioiden tallettamisesta tietokantaan ja tiedostojärjestelmään [7]. Seuraavaksi tarkastellaan tätä tutkimusta ja sen tuloksia.

5.1. Microsoftin ja Berkleyn yliopiston tutkimus

Sovellussuunnittelija joutuu valitsemaan tallentaako hän suuret objektit tiedostojärjestelmään vai tietokantaan. Yleensä tämä päätös perustuu sellaisille tekijöille kuin sovelluksen yksinkertaisuus tai ylläpidettävyys. Suorituskyky vaikuttaa usein näihin tekijöihin.

Perinteisesti tietokantaa pidetään tehokkaampana käsittelemään suuria määriä pieniä objekteja kun taas tiedostojärjestelmä on tehokkaampi suurille objekteille. Mikä on raja-arvo? Milloin suuren objektin tallettaminen tietokantaan on edullisempaa kuin tiedostojärjestelmään?

Tämä riippuu tietenkin kyseessä olevasta tiedostojärjestelmästä, tietokantajärjestelmästä ja kyseeseen tulevasta työkuormasta. Microsoftissa on tehty asiaa koskeva tutkimus [7], jossa on tarkasteltu NTFS (New Technology File System) -tiedostojärjestelmää ja SQL Server 2005-tietokantajärjestelmää erilaisilla toimenpiteillä (create, read, replace, delete) kuormittaen ja erikokoisia objekteja käyttäen. Tutkimuksessa mitattiin tietokantapalvelimen suorituskykyä tyypillisessä web-sovellusympäristössä ja suorituskyvyn muuttumista ajan kuluessa ja muistin fragmentoituessa. Tutkimuksessa käytetyn NTFS-tiedostojärjestelmän fragmentoitumisen hallintajärjestelmä oli parempi kuin tietokannan hallintajärjestelmän.

Tiedostojärjestelmät ovat jo kauan käyttäneet kehittyneitä muistinvarausmenetelmiä välttääkseen fragmentoitumisen tallettaessaan objekteja levylle. NTFS-tiedostojärjestelmä käyttää nauhamaista (banded) varausmenetelmää metadatalle, mutta ei tiedoston sisällölle. Se varaa tilaa tiedostolle ajonaikaisesta hakuvälimuistista (run based lookup cache). Koon ja tilavuuspoikkeamien vähentämiseksi järjestetään jatkuvista, vapaista varausyksiköistä jonoja. Uusi tila pyritään varaamaan ulommilta kaistoilta ja mikäli siinä ei onnistuta, käytetään vapaata välimuistia. Mikäli tämäkään ei onnistu, tiedosto pilkotaan. Kun tiedosto poistetaan,

varattua tilaa ei käytetä välittömästi, vaan NTFS-tapahtumalokimerkintä on tehtävä ennen kuin tila voidaan käyttää uudelleen.

Tiedostojärjestelmillä ja tietokannoilla on erilaiset lähestymistavat olemassa olevan objektin muokkaamiseen. Tiedostojärjestelmät on optimoitu tiedoston jatkamiseen ja lyhentämiseen. Paikalleen tapahtuvat tiedostojen päivitykset ovat tehokkaita, mutta kun tietoa lisätään tai poistetaan tiedoston keskeltä, kaikki muutoskohdan jälkeinen tietosisältö on kirjoitettava kokonaan uudelleen. Jotkin tietokannat kirjoittavat myös muutetut suuret oliot kokonaan uudelleen, tämä uudelleenkirjoitus on läpinäkyvä sovellukselle. Toiset kuten myös SQL Server omaksuvat Exodus-mallin, joka siis tukee tehokasta objektin sisäistä lisäystä ja poistoa käyttämällä B-puuhun perustuvaa suurten objektien tallennusta. Jotkin tietokannat (mm. IBM:n DB2) tallettavat suuret oliot tiedostojärjestelmään ja käyttävät tietokantaa tiedoston metadataan yhdistävänä hakemistona.

Parantaakseen tietokannan huonoa suurten, fragmentoituneiden objektien käsittelyä, sovellus voi tehdä oman eheytyksensä tai muistinsiivouksensa. Jotkin sovellukset yksinkertaisesti tallettavat kunkin objektin suoraan tietokantaan yksinkertaiseksi BLOB:iksi (ks. luvut 6 ja 7) tai yksinkertaiseksi tiedostoksi tiedostojärjestelmään. Toiset sovellukset taas käyttävät monimutkaisempia muistivaraus-järjestelmiä palvellakseen tehokkaammin käyttäjän pyyntöjä. Objektit voidaan esimerkiksi jakaa useisiin pieniin osiin, jotka talletetaan BLOB:eiksi. Videotallenteet ositetaan usein tähän tapaan. Toisaalta myös pienempiä objekteja voidaan koota yhteen pitkään bittijonoon tai tiedostoon (TAR, ZIP tai CAB tiedostot).

5.1.1. Tutkimusjärjestelyt

Tutkimuksessa käsiteltiin vain kokonaisten objektien lisäystä, korvausta tai poistoa. Korvaustapauksessa luotiin uusi versio ja vanha tuhottiin. Kokonaisten objektien turvallinen tallennuskaan ei ole aina niin suoraviivaista. Suurin osa tiedostojärjestelmistä suojelee sisäisiä metadatarakenteita, kuten hakemistoja ja tiedostonimiä korruptoitumiselta epätyypillisten sulkemisten (systeemin kaatuminen, virtakatkokset,…) yhteydessä. Tiedoston sisällölle ei ole kuitenkaan välttämättä tällaista varmistusta. Varsinkin tiedostojärjestelmä ja käyttöjärjestelmä, jolla se sijaitsee, organisoivat kirjoituspyynnöt uudelleen parantaakseen suorituskykyä. Vain osa näistä pyynnöistä voi olla suoritettuina äkillisen järjestelmän sulkemisen yhtey-

dessä. Tämän takia monet työpöytäsovellukset käyttävätkin tekniikkaa, jota kutsutaan turvalliseksi kirjoittamiseksi (safe write) saavuttaakseen äkillisen sulkemisen jälkeen tilanteen, jossa tiedoston sisältö on joko uusi tai vanha, mutta ei uuden ja vanhan sekoitus. Koska turvallinen kirjoitus takaa, että vanha tiedosto varmasti korvataan, ne myös pakottavat tiedoston täydellisen kopioinnin levylle, vaikkakin suurin osa tiedoston sisällöstä olisi muuttumatonta. Turvallisella kirjoituksella tiedoston uusi versio luodaan väliaikaisella nimellä ja seuraavaksi uusi data kirjoitetaan väliaikaiseen tiedostoon ja viedään levylle. Viimeiseksi uusi tiedosto nimetään uudelleen pysyvälle tiedostonimelle ja siten poistetaan vanhempi tiedosto. Unixissa rename-funktion taataan kirjoittavan vanhan tiedostoversion päälle, Windowsissa ReplaceFile-kutsua käytetään tiedoston korvaamiseen toisella.

Tietokannoissa taas käytetään tyypillisesti tapahtumatyyppisiä päivityksiä, jotka sallivat, että sovellus niputtaa monia muutoksia yhteen yksikköön, jota kutsutaan tapahtumaksi (transaction). Valmiit tapahtumat suoritetaan kokonaisina tietokantaan riippumatta epäpuhtaista systeemin, tietokannan tai sovelluksen kaatumisista. Tietokantasovelluksesta ja päivityksen luonteesta riippuen voi olla tehokkaampaa päivittää vain pieni osa suuresta oliosta koko olion päällekirjoittamisen sijaan. Tietokanta turvaa tapahtumaa lukitsemalla sekä metadata- että datamuutokset tapahtuman ajaksi. Kun kyseessä oleva loki on tallennettu levylle, siihen liittyvien tietokantamuutoksienkin taataan tulevan suoritetuiksi. Loki kirjoitetaan peräkkäisjärjestyksessä ja vastaavat levykirjoitukset tietokantaan voidaan järjestää uudelleen hakujen minimoimiseksi. Jokaisen muutoksen lokimerkinnät kirjoitetaan, mutta tietokantamuutokset voidaan sulauttaa. Datan ja metadatan merkitseminen lokiin takaa oikeellisuuden kaiken tiedon kahteen kertaan kirjoittamisen kustannuksella, kerran kantaan ja kerran lokiin. Suurten objektien tapauksessa tämä merkitsee huomattavaa suorituskykyyn vaikuttavaa haittaa tietokannalle. Tietokantalokiin peräkkäisesti kirjoittaminen on suurin piirtein vastaavaa kuin peräkkäiskirjoitus tiedostoonkin. Seuraava kirjoitus tietokantasivuille muodostaa ylimääräisen peräkkäiskirjoituksen, jos kaikki tietokannan sivut ovat jatkuvia tai monia hakuvaltaisia kirjoituksia, jos sivut eivät sijaitse toistensa lähellä. Suuret objektit myös täyttävät lokin aiheuttaen tarpeen säännöllisempään lokin typistämiseen ja varmistamiseen, mikä vähentää mahdollisuuksia uudelleen järjestää ja yhdistellä sivumuutoksia.

Tässä tutkimuksessa on käytetty SQL Serverin bulk-logged optiota, joka parantaa tätä epäkohtaa. Tämä optio on erikoistapaus BLOB:n muistin varauksesta ja vapauttamisesta. Alkuperäisiä BLOB:iin kirjoituksia ja niiden lopullista poistoa ei kirjoiteta lokiin, johon viedään

pikemminkin metadatamuutokset (varaus- ja vapautustiedot) Kun toiminto suoritetaan (commit), myös BLOB-muutokset suoritetaan. Tämä tukee toiminnon peruutusta ja lämmintä uudelleenkäynnistystä, mutta ei tietovälineiden elpymistä (suuresta oliosta ei ole toista kopiota lokissa).

Yleisesti tietokantoja pidetään parempina pienille objekteille ja tiedostojärjestelmää suurille. Pienen ja suuren raja on kuitenkin sumea. Syitä tähän ovat mm. seuraavat:

- Tietokantakyselyt ovat nopeampia kuin tiedoston avaamiset, tiedostokahvan avaamisen rasite hallitsee suorituskykyä, kun käsitellään pieniä objekteja.

- Suurten tiedostojen lukeminen tai kirjoittaminen on nopeampaa kuin suurten tietokantaobjektien hakeminen. Tiedostojärjestelmät on optimoitu suurten tietovirtojen käsittelyyn.

- Tietokanta-asiakasyhteydet eivät ole hyviä käsittelemään suuria objekteja. Etätietokanta-asiakasyhteydet ovat perinteisesti optimoitu lyhytkestoisia, pieniä tietomääriä palauttavia kyselyitä silmällä pitäen.

- Tiedostojen avaaminen on CPU-rasitteista.

Lisäksi on muitakin huomionarvoisia näkökohtia. Esimerkiksi sovelluksilla, jotka tallettavat suuret objektit tiedostojärjestelmään, on ongelmana pitää tietokantaobjektin metadata ja tiedostojärjestelmän objektin data toistensa tasalla. Erityisesti ongelmia aiheuttaa tarpeettomien tiedostojen kokoelma (garbage collection) silloin, kun tiedosto on poistettu tietokannasta, mutta ei tiedostojärjestelmästä. Puutteita löytyy myös esim. replikaatiosta, tietojen varmistuksesta, ongelmatilanteista toipumisesta ja fragmentoitumisesta.

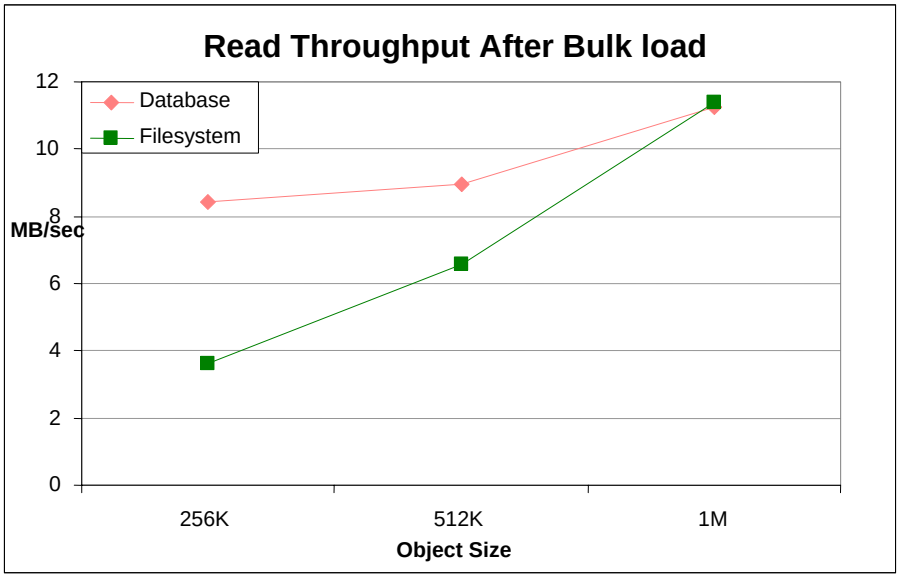
Eri järjestelmissä on yllättävän erilaiset lähestymistavat fragmentoitumisongelmaan Esim. FFS (Fast File System) pystyy pitämään fragmentoitumisen hallinnassa niin kauan kuin taltiot pidetään alle 90% täytettyinä. Unix-muunnelmat varaavat tietyn määrän vapaata tilaa sekä hätätilanteista toipumiseen, että estääkseen liiallisen fragmentoitumisen. NTFS on kokenut monta uudistusta, kun sen suorituskykyä fragmentoitumisen suhteen on eri Windows-versioissa paranneltu.

Tutkimuksessa kuormitettiin sekä tietokantaa että tiedostojärjestelmää vektoriperusteisen työkuormakehittimen tuottamalla kuormalla, joka vastaa hyvin todellista tilannetta. Molempien testit oli suunniteltu niin samankaltaisiksi kuin suinkin mahdollista. Testeissä oletettiin, että on yhtä todennäköistä, että objekteja kirjoitetaan kuin luetaan ja kussakin testissä käytettiin suurin piirtein samankokoisia objekteja, koska objektin koon vaihtelun uskottiin vaikuttavan tuloksiin.

Tutkimuksessa keskityttiin erityisesti tukimaan tallennusjärjestelmiä ajan kuluessa ja muistin vanhetessa. Muistin iälle jopa ehdotetaan uutta mittaria, joka on niiden tavujen määrän, joita taltiolla on joskus ollut, suhde taltion kokonaistavumäärään. Tutkimus käyttää suoritustehoa pääasiallisena toimintakyvyn mittarina. Tutkimus aloitettiin tyypillisenä suorituskykytutkimuksena ja sitten keskityttiin pitkä aikaisen fragmentoitumisen vaikutuksiin 256KB-1MB kokoisiin objekteihin, missä sekä tiedostojärjestelmällä että tietokannalla on vertailukelpoiset suorituskyvyt.

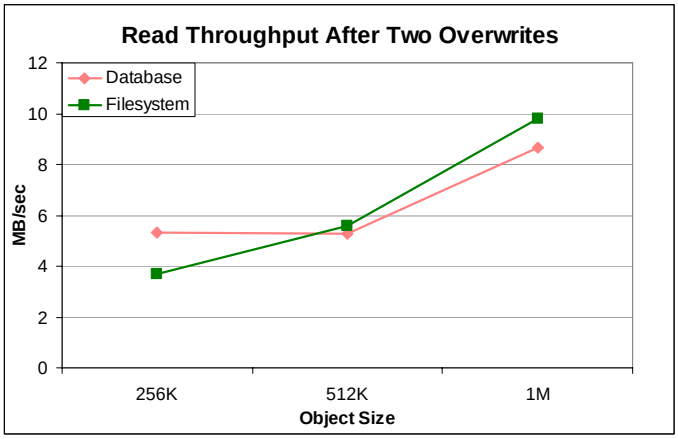
5.2. Tuloksia

Seuraavassa on esitelty muutamia kokeissa saatuja tuloksia osin myös graafisina käyrinä.

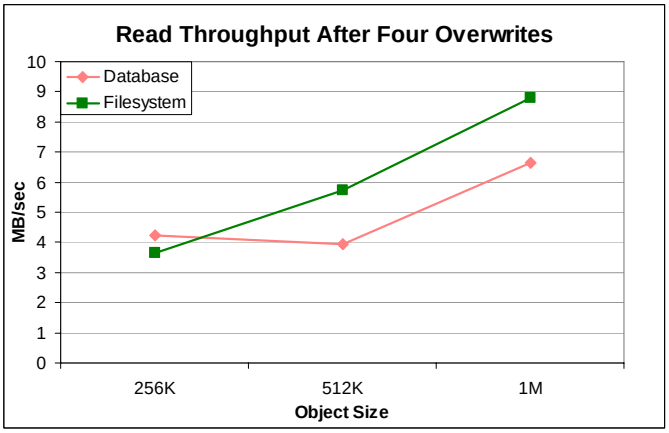


Kuva 12 Lukemisen suoritusteho heti tietojen massalataamisen jälkeen [7]

Kuvan 12 mukaan tietokanta on nopeampi pienillä objekteilla. Kun objektien koko kasvaa, tiedostojärjestelmän suoritusteho kasvaa nopeammin kuin tietokannan.

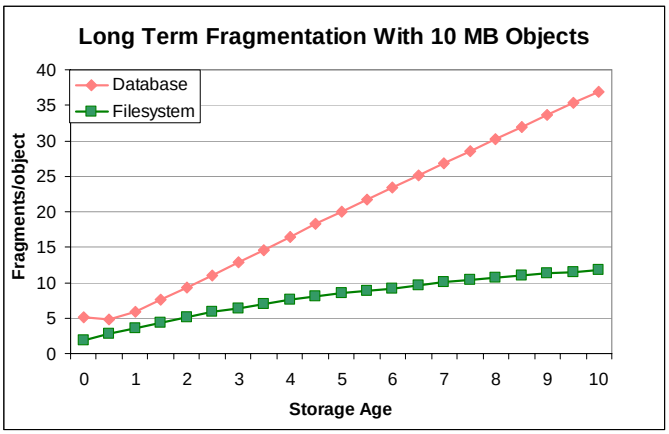


Kuva 13 Suorituskyky kahden uudelleenkirjoituksen jälkeen [7]



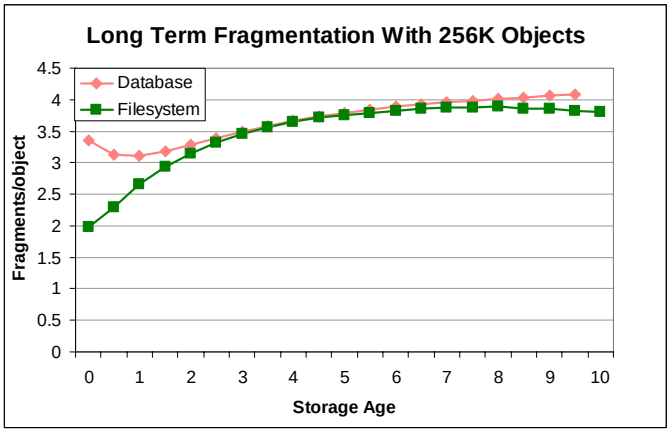
Kuva 14 Suorituskyky neljän uudelleenkirjoituksen jälkeen [7]

Fragmentoituminen huonontaa suorituskykyä ajan myötä. Tiedostojärjestelmä ei ole tälle niin altis kuin tietokanta. Ajan myötä tiedostojärjestelmä suoriutuu paremmin kuin tietokanta, kun objektit ovat 256KB suurempia. Kuvassa 13 objekti on korvattu uudella versiolla kaksi kertaa ja kuvassa 14 neljä kertaa.



Kuva 15 Pitkän aikavälin fragmentoituminen suurilla objekteilla [7]

Suurille objekteille tiedostojärjestelmä hallitsee fragmentoitumisen tehokkaammin kuin tietokanta. Kuvissa 15 ja 16 ”Storage Age” on se keskimääräinen määrä kertoja, jolla jokainen objekti on korvattu uudemmalla versiolla.



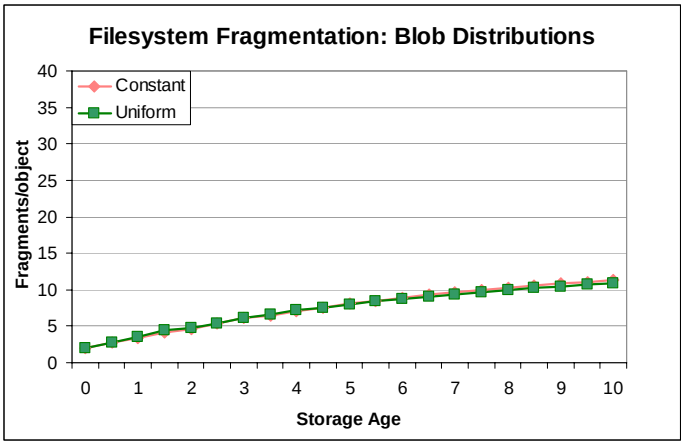
Kuva 16 Pitkän aikavälin fragmentoituminen pienillä objekteilla [7]

Kuten kuva 16 osoittaa, niin pienillä objekteilla järjestelmien käyttäytyminen on samankaltaista.

Tulokset siis viittaavat siihen, että kun muistin ikä kasvaa, niin piste, jossa tiedostojärjestelmällä ja tietokannalla on samankaltainen suorituskyky, laskee 1MB:stä 256KB:iin. Objektit, jotka ovat korkeintaan 256KB kokoisia, kannatta tallettaa tietokantaan ja 1MB:tä suuremmat tiedostojärjestelmään. Tällä välillä olevien objektien kohdalla asia riippuu siitä, kuinka

kirjoitusherkkä järjestelmä on sekä muistin vanhenemisesta. Tämän tutkimuksen mukaan muistin ikä oli yksi määräävistä tekijöistä, sillä muistin vanhetessa myös sen pirstaloituminen lisääntyy.

Lisäksi tutkimuksessa havaittiin, että muistin koko ei vaikuta suorituskykyyn isommilla tallennusvolyymeillä, mutta pienemmillä todettiin, että kun vapaan tilan määrä suhteessa objektien kokoon kasvaa, niin suorituskyky heikkenee. Olisi suositeltavaa, että n. 10%:a muistikapasiteetistä olisi vapaana, tällöin suorituskyvyn heikkeneminen on vielä lähes merkityksetöntä.



Kuva 17 Tiedostojärjestelmän fragmentoituminen kiinteänkokoisilla ja vaihtuvankokoisilla objekteilla [7]

Suurikokoisilla objekteilla (10MB) tiedostojärjestelmän fragmentoituminen lisääntyy hitaasti, tietokannan nopeasti. Kuitenkaan fragmentoitumisella käytettäessä kiinteänkokoisia objekteja, ei ole huomattavaa eroa vaihtuvamittaisiin, satunnaisesti valittuihin, suurin piirtein samankokoisiin objekteihin verrattaessa (kuva 17).

6. Suuret oliot Oracle-kannassa

Oracle on tällä hetkellä ehkä eniten käytetty relaatiotietokanta, jossa on nykyisin myös olio-suuntautuneita piirteitä.

6.1. Suurten olioiden tietotyyppejä

Oraclessa on useita eri tietotyyppejä suurten olioiden tallentamiseen: BLOB (Binary Llarge Object), CLOB (Character Large Object), NCLOB (National Character LOB) ja BFILE. Oraclen LOB-tietotyyppeihin voidaan tallettaa 8-128 teratavua käytettäessä DBMS_LOB PL/SQL-pakettia, OCI:a tai JDBC:tä. Muutoin maksimi on 4 gigatavua. Yli 4 kilotavuisen LOB-ilmentymän LOB-arvo tallennetaan kuitenkin tietokannan ulkopuolelle [12].

6.1.1. BLOB

BLOB-tietotyyppiä käytetään tallennettaessa isoja binääritiedostoja mm. kuvia, ääntä, videoita, taulukoita ja multimediaa sekä jopa binaarista suoritettavaa ohjelmakoodia tietokantaan [12]. Myös esim. Informixissä on BLOB-tietotyyppi ja kenttään mahtuu 4 teratavua [4].

6.1.2. CLOB

CLOB-tietotyyppiä käytetään tallennettaessa tietokantaan pitkää tekstitietoa tai isoja tekstitiedostoja, jotka käyttävät tietokannan kirjaimistoa [12]. CLOB-kenttää ei tietenkään kannata käyttää, mikäli talletettava teksti mahtuu esimerkiksi CHAR- tai VARCHAR2-tyyppisiin kenttiin. CHAR-tietotyyppiin voidaan tallettaa maksimissaan 2000-merkkisiä tekstejä ja VARCHAR2-tyyppiseen 4000 merkkisiä [15].

Oraclen lisäksi CLOB löytyy mm. seuraavista tietokannoista: IBM DB2 UDB (maksimikoko 2GB) [30] ja iSeries DB2/400 (maksimikoko myös 2GB) [27].

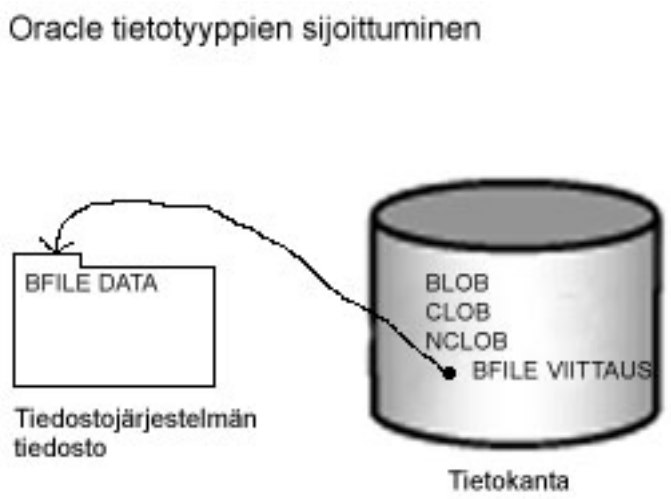
6.1.3. NCLOB

NCLOB-tietotyyppi on Oraclen suurten yksi- tai useampitavuisten kiinteämittaisten tekstien tietotyyppi. Toisin sanoen kirjaimiston vaatima tavumäärä voi olla vaihteleva. Maksimikoko on neljä gigatavua.

6.1.4. BFILE

BFILE on Oraclen suurten binaaritiedostojen tietotyyppi, jossa tiedot talletetaan varsinaisen tietokannan ulkopuolelle (kuva 18), mutta ovat saatavilla tietokantaan read-only periaatteella. BFILE siis tallettaa staattista tietoa, joita ei tarvitse muokata sovelluksessa. Tietokanta käyttää viittauksia BFILE sarakkeilla.

Mitä tahansa tietoa, ts. mitä tahansa käyttöjärjestelmään tallennettavaa tietoa, voidaan tallettaa tämän tyyppiseen kenttään, esimerkiksi binaarista tietoa, joka ei muutu sovelluksen ajon aikana (esim. grafiikkaa ja kuvia) tai vain luettavaa tietoa, joka on suhteellisen pitkää tilan säästämiseksi tietokannassa. Mikä tahansa käyttöjärjestelmässä oleva peräkkäisluettava laite voi sisältää BFILE-tietoa (kovalevy, CD-ROM, DVD,…).



Kuva 18 Oracle tietotyyppien sijoittuminen tietokantaan ja tiedostojärjestelmään

6.2. Tallennusrakenteista ja niiden käsittelystä

Oraclen suurten olioiden talletukseen tarkoitetut tietotyypit tukevat satunnaishakua, kun taas LONG tietotyyppi on tarkoitettu peräkkäishakuun.

Tietokannan alueelle suuria olioita tallentavien tietotyyppien (BLOB, CLOB, NCLOB eli tietokannan sisäisten tietotyyppien) käsittely on optimoitu tilaa säästäväksi ja tehokasta hakua tukevaksi.

Tietokannan sisäiset suuret oliot (LOB) voivat olla joko pysyviä tai väliaikaisia. Pysyvä LOB on LOB-ilmentymä, joka on talletettu tietokantataulun riville. Väliaikainen LOB-ilmentymä luodaan, kun LOB luodaan vain paikallisen sovelluksen vaikutusalueelle. Väliaikaisesta ilmentymästä tulee pysyvä, kun se sijoitetaan tietokantatauluun riville.

ACID-ominaisuudet pätevät myös pysyviin suuriin olioihin.

LOB:illa on paikannin (locator) ja arvo. Paikannin on viite paikkaan, johon LOB-arvo on fyysisesti sijoitettu. Arvo on se tieto, joka on tallennettu LOB-kenttään. LOB-paikantimeen ei tarvitse viitata osoittimen tavoin.

Sisäiset LOB:it käyttävät kopiointisemantiikkaa eli sekä paikannin että arvo kopioidaan lisäyksen, päivityksen tai sijoittamisen yhteydessä. Tämä varmistaa sen, että jokainen taulun solu tai muuttuja, joka sisältää LOB:in, pitää sisällään ainutkertaista LOB-ilmentymää.

Ulkoiset LOB:it käyttävät viittaussemantiikkaa eli vain LOB-paikannin kopioidaan lisäys-operaatiossa. Päivityshän ei ole sallittua ulkoiselle LOB:lle, koska ne ovat read-only-tyyppisiä.

LOB-sarakkeen solu voi olla kolmessa tilassa. Se voi olla NULL eli se on luotu, mutta ei sisällä paikanninta eikä arvoa. Se voi olla tyhjä, jolloin sillä on paikannin, mutta ei arvoa. Solun pituus on tällöin nolla. Ja viimein solu voi olla käytössä, jolloin sillä on sekä paikannin, että arvo.

Rivin voi lukita ja täten estää muita tietokannan käyttäjiä tekemästä muutoksia LOB:iin toiminnon aikana. Lukitseminen tapahtuu lisäämällä FOR UPDATE rivin haun yhteyteen.

LOB ilmentymä voidaan avata joko luettavaksi tai kirjoitettavaksi. Jos se on avattu, niin se on myös suljettava jossain vaiheessa istuntoa. Avoimelle ilmentymälle voi suorittaa niin monta toimintoa kuin tarpeen, edellyttäen, että avausmoodi kyseiset toiminnot sallii.

Tietokannan LOB-ilmentymä voidaan käsitellä myös LOB-paikantimien avulla. Jos LOB-ilmentymä on NULL, niin sillä ei siis ole paikanninta. Ennen kuin LOB-ilmentymää voidaan käyttää, sen on sisällettävä paikannin. Koska tyhjällä LOB:lla on paikannin, asia voidaan hoitaa esimerkiksi lisäyksen yhteydessä käyttämällä EMPTY_CLOB (CLOB) ja EMPTY_BLOB (BLOB, NCLOB) -funktioita arvon asemasta. BFILE-tyyppiset kentät taas alustetaan käyttämällä BFILENAME-funktiota.

LOB-kenttiin liittyy myös muutamia rajoitteita. LOB:ia ei mm. voi määritellä pääavaimeksi ja hajautettuja LOB:eja ei tueta. Myöskään aivan kaikki SQL-lause vaihtoehdot eivät ole mahdollisia [12].

6.3. Esimerkkejä

Esimerkki BFILENAME-funktion käytöstä: BFILENAME-funktiolla, jolle annetaan polku hakemisto-objektina ja tiedoston nimi, saadaan paikannin tiedostoon:

```
CREATE DIRECTORY "IMG" AS 'C:\IMAGES';


CREATE TABLE Lob_table (
   Key_value NUMBER NOT NULL,
   F_lob BFILE)

   INSERT INTO Lob_table VALUES
      (21, BFILENAME('IMG', 'Image1.gif'));
   INSERT INTO Lob_table VALUES
      (22, BFILENAME('IMG', 'image2.gif'));
```

Oraclessa LOB-tietotyyppejä käyttävien kenttien tuki on rakennettu moniin SQL-funktioihin. Tämä tuki mahdollistaa LOB-kenttien käsittelyn käyttämällä SQL-semantiikkaa. Seuraavassa PL/SQL esimerkissä AD_FINALTEXT haetaan final_ad nimiseen VARCHAR-puskuriin:

```
DECLARE
final_ad VARCHAR(32767);
BEGIN
   SELECT AD_FINALTEXT INTO final_ad FROM print_media
   WHERE PRODUCT_ID= 2056 and AD_ID= 12001 ;
   DBMS_OUTPUT.PUT_LINE(final_ad);
…
END;
```

VARCHAR-tyyppisen muuttujan maksimikoko on PL/SQL:ää käytettäessä 32767 tavua.

6.4. Oracle Multimedia

Sekä Oracle 11g Standard Edition että Enterprise Edition sisältävät myös Oracle Multimedian (aiemmin Oracle interMedia), joka käsittelee multimediatietoja (kuvia, ääntä ja videoita) [17]. Se tuntee tavallisimmat multimedia-formaatit ja voi automatisoida metadatan talteenottoa ja peruskuvankäsittelyä. Oracle Multimedia mahdollistaa metadatan ja tietojen tallettamisen yhdessä tietokannan alueelle, mutta tarjoaa myös vaihtoehdon tallettaa tiedot tietokannan ulkopuolelle. Esimerkiksi relaatiorajapinnan (relational interface) avulla metadataa voidaan käsitellä BLOB- tai BFILE-tietotyyppeihin tallentaen ilman, että tarvitsisi käyttää Oracle Multimedian omia objektitietotyyppejä.

Sovellukset pääsevät käsiksi Oracle Multimediaan käyttäen relaatio-, objekti- tai SQL Multi Media (ISO-standardi) -rajapintoja. Java, C++ tai perinteiset 3GL-sovellukset voivat käyttää sitä luokkakirjastojen, PL/SQL:n tai Oracle Call Interfacen (OCI) avulla. Suurissa erissä multimediaa voidaan ladata tietokantaan joko SQL Loaderin (Oracle-ominaisuus) avulla, PL/SQL:llä tai Java-proseduureilla BLOB:eista, tiedostojärjestelmästä tai URL-tietolähteistä Oracle Multimedian objektityyppikenttiin.

Oracle Multimedian objektitietotyypit ovat samanlaisia kuin Javassa ja C++:ssa. Näitä objektitietotyyppejä kutsutaan vastaavasti ORDImageksi, ORDAudioksi ja ORDVideoksi.

Oracle Multimedia sisältää myös objektin, joka tunnetaan ORDDoc:ina. Tämä multimedia kenttätyyppi voi sisältää mitä tahansa kuva-, ääni- ja video-objektien sekoituksia.

Näiden objektitietotyyppien ilmentymä koostuu attribuuteista kuten formaatti, metadata- ja multimediasisältö mukaan lukien metodit. Multimedia sisältö koostuu varsinaisesta kuvasta, äänestä tai videosta. Metadata on formaattitietoa multimediasisällöstä sisältäen mm. objektin pituuden, pakkaustietoja tai formaatin, mime typen ym. Metodit ovat proseduureja, jotka voidaan suorittaa objektille, esimerkiksi import, export, getMetadata ja processCopy kompressoitaessa tai konvertoitaessa kuvaformaatteja

Sovellukset käyttävät Oracle Multimediaa lisäämällä yhden tai useamman multimedia-sarakkeen (ORDImage, ORDAudio, ORDVideo tai ORDDoc) olemassa olevaan tauluun tai luomalla uusi taulu, jossa on multimediasarakkeita. Näitä sarakkeita voidaan lisätä tauluun kuinka monta tahansa.

6.4.1. Esimerkkejä

Luodaan kuvataulu ja lisätään kaksi riviä, joissa on tyhjät ORDImage sarakkeet ja alustetaan objektiattribuutit [14]. Käyttäjällä on oltava CREATE TABLE oikeudet.

```
SET SERVEROUTPUT ON;
SET ECHO ON;

DROP TABLE image_table PURGE;
CREATE TABLE image_table ( id NUMBER,
   image ORDImage )
LOB(image.source.localData) STORE AS SECUREFILE;

INSERT INTO image_table VALUES(1,ORDImage.init());
INSERT INTO image_table VALUES(2,ORDImage.init());

COMMIT;
```

Seuraavassa PL/SQL esimerkissä haetaan kaksi kuvatiedostoa (img71.gif ja img50.gif) MEDIADIR-hakemistosta edellä luotuun tauluun:

```
SET SERVEROUTPUT ON
SET ECHO ON

DECLARE
   obj ORDIMAGE;
   ctx RAW(64) := NULL;
BEGIN
image blob.
   select Image into obj from image_table where id = 1 for update;
   obj.setSource('file','MEDIADIR','img71.gif');
   obj.import(ctx);
   update image_table set image = obj where id = 1;
commit;

   select Image into obj from image_table where id = 2 for update;
   obj.setSource('file','MEDIADIR','img50.gif');
   obj.import(ctx);
   update image_table set image = obj where id = 2;
commit;
END;
/
```

7. Suuret oliot MySQL-kannassa

MySQL on eniten käytetty avointa lähdekoodia käyttävä tietokantaohjelmisto. Myös MySQL:ssä on useita tietotyyppejä suurten olioiden tallentamiseen.

7.1. Tietotyyppejä

7.1.1. BLOB

MySQL-tietokannassa on neljä BLOB (binary large object) tyyppiä: TINYBLOB, BLOB, MEDIUMBLOB ja LONGBLOB riippuen talletettavan tiedon pituudesta.

Niiden tallennuskapasiteetit ovat:

TINYBLOB < 2^8 tavua ja yksi tavu

BLOB < 2^{16}+2 tavua

MEDIUMBLOB < 2^{24}+3 tavua

LONGBLOB < 2^{32}+4 tavua

Lisätavut (1-4) johtuvat siitä, että vaihtelevamittaisten kenttien tapauksessa myös kentän pituus talletetaan. Fyysisesti kentän pituus talletetaan ennen varsinaista kentän sisältöä. [9]

7.1.2. TEXT

TEXT-tietotyypit ovat MySQL:n CLOB (character large object)-tietotyyppejä ja niitä käytetään tallennettaessa isoja tekstitiedostoja tietokantaan.

MySQL-tietokannassa on neljä TEXT tyyppiä: TINYTEXT, TEXT, MEDIUMTEXT, ja LONGTEXT. Näiden koot ovat vastaavat kuin samanalkuisten BLOB-tietotyyppienkin.

7.2. Tallennusrakenteista ja niiden käsittelystä

BLOB-kentillä ei ole kirjaimistoa ja lajittelu ja vertailu perustuvat kenttien tavujen numeerisiin arvoihin. TEXT-kentillä on kirjaimisto ja arvot talletetaan ja niitä vertaillaan kirjaimistoon perustuen.

Pitkien kenttien käsittelyssä on muistettava, että MySQL sisältää muutamia niiden käsittelyä rajoittavia tekijöitä. Esimerkiksi lajittelussa vain ensimmäiset *max_sort_length*-optiossa määritellyt tavut vertaillaan. Oletusarvo on 1024 ja sitä voidaan muuttaa käyttämällä max_sort_length=N-optiota, kun mysqld-palvelin käynnistetään. Kukin asiakassovellus voi myös muuttaa sitä istunnon ajaksi seuraavalla tavalla:

```
mysql> SET max_sort_length = 2000;
```

BLOB- tai TEXT-objektin maksimikoko määräytyy sen tyypin mukaan, mutta todellisuudessa suurin asiakkaan ja palvelimen välillä siirrettävissä oleva arvo määräytyy käytettävissä olevan muistin ja tiedonsiirtopuskurien koon mukaan. Viestipuskurin koko voidaan muuttaa muuttamalla *max_allowed_packet*-muuttujan arvoa sekä asiakas- että palvelinkoneella. Kukin BLOB- tai TEXT-kentän arvo on sisäisesti erillisesti varattu objekti päinvastoin kuin kaikki muu tieto, jolle muistia varataan kerran kutakin taulun saraketta varten [9].

Myös tehtäessä indeksejä BLOB- tai TEXT-kentille, on aina määriteltävä indeksipituus. Se tapahtuu *col_name (N)*-syntaksilla, missä N tarkoittaa sitä, että vain kentän N ensimmäistä merkkiä otetaan huomioon indeksoinnissa.

Esimerkki:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

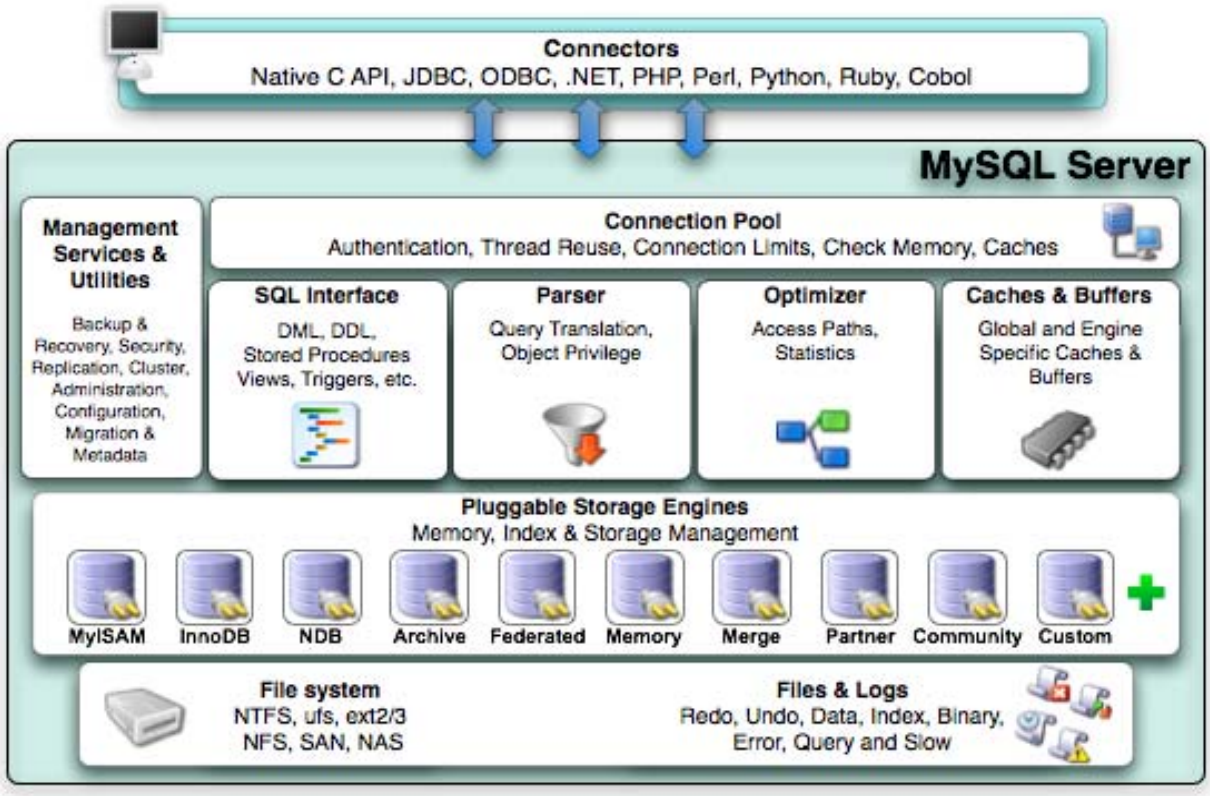
Näin indeksitiedoston koko pysyy paljon pienempänä.

Myös muuttujaa max_allowed_packet (yhden paketin tai minkä tahansa luodun tai väliaikaisen merkkijonon maksimipituus) on kasvatettava suuria olioita käsiteltäessä. Pakettiviestipuskuri on alustettu *net_buffer_length*:iin, mutta kasvaa tarvittaessa

max_allowed_packet:iin. Yläraja on kuitenkin 1GB. Pakettikokoa pidetään tarkoituksella pienenä, jotta mahdolliset virheelliset, isot paketit huomattaisiin. Suuret oliot vain eivät välttämättä ole virheellisiä.

Muuttuja net_buffer_length, joka mainittiin yllä, on yhteys- ja tulospuskureiden oletuskoko, joka kasvaa tarvittaessa automaattisesti korkeintaan max_allowed_packet –kokoon. Toiminnon suorittamisen jälkeen puskurien koko on jälleen oletusarvoisesti net_buffer_length (voidaan asettaa korkeintaan 1MB suuruiseksi).

Sovelluskehittäjät voivat valita MySQL-kannassa monista erilaisista muistinhallintajärjestelmistä (storage engines). MySQL-palvelinarkkitehtuuri eristää sovelluskehittäjän ja tietokanta-admistraattorin muistijärjestelmän matalan tason toteutuksesta ja tarjoaa näin yhtenäisen ja helpon sovellusmallin ja API:n. MySQL-arkkitehtuurin tarjoamat muistinhallintavaihtoehdot on esitelty kuvassa 19.



Kuva 19. MySQL-arkkitehtuuri käyttää eristettyä muistinhallintaa (Pluggable Storage Engines) [9]

7.3. Esimerkkejä

Pitkän bittijonon sisältävän taulun luominen MySQL:ään kirjautuneena suoritetaan[8]:

```
CREATE TABLE files(
file_id SMALLINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
file_name VARCHAR(64) NOT NULL,
file_size MEDIUMINT UNSIGNED NOT NULL,
file MEDIUMBLOB NOT NULL);
```

Samalla kannattaa kasvattaa pakettikokoa:

```
mysql> SET max_allowed_packet=15M;
```

Yhteyden luominen tapahtuu Microsoftin Visual Basic-ympäristössä seuraavasti:

```
Dim conn As ADODB.Connection
Set conn = New ADODB.Connection
conn.ConnectionString = GloConnectionString
conn.CursorLocation = adUseClient
conn.Open
```

missä `GloConnectionString` on esim. seuraavankaltainen:

```
"DRIVER={MySQL ODBC 3.51 Driver};" _
& "SERVER=127.0.0.1;" _
& "DATABASE=test;" _
& "UID=testuser;" _
& "PWD=12345;" _
& "OPTION=" & 1 + 2 + 8 + 32 + 2048 + 16384
```

Kuvatiedoston lataamiseksi tietokantaan käytetään ADO Recordset-oliota ja Stream-oliota, jotka määritellään seuraavasti:

```
Dim rs As ADODB.Recordset
Set rs = New ADODB.Recordset
Dim mystream As ADODB.Stream
Set mystream = New ADODB.Stream
mystream.Type = adTypeBinary
```

ADO Stream-olio voi käsitellä sekä tekstiä, että binaaridataa. Käyttämällä Type-parametrin adTypeBinary-arvoa määritellään käsiteltävän tiedon tyypiksi binaarinen.

Seuraavassa avataan tietuejoukko *rs* ja ladataan tiedosto ja viedään se tietokantaan:

```
rs.Open "SELECT * FROM files WHERE 1=0", conn, adOpenStatic,
adLockOptimistic
rs.AddNew

mystream.Open
mystream.LoadFromFile "c:myimage.gif"

rs!file_name = "myimage.gif"
rs!file_size = mystream.size
rs!file = mystream.read
rs.Update
mystream.Close
rs.Close
conn.Close
```

Seuraavana on PHP-esimerkki MySQL-tietokantayhteyden luomisesta:

```
<?php
class Connect
{
   function __construct(){
   }
   function openDatabase(){
       // Kytkeydytään palvelinkoneeseen
/*Tietokantayhteyden muodostaminen
Muuttujat:
$server:palvelin
$user:käyttäjätunnus
$password: salasana
$database: tietokanta
$mysqli: palautettava mysql-yhteys
*/
    $server="localhost";
    $user="käyttäjätunnus";
    $password="salasana";
    $database="tietokanta";
    $mysqli = new mysqli("$server","$user","$password","$database");
    if (mysqli_connect_errno()) {
          die("Yhteys epäonnistui");
     }
// Palautetaan tietokantaolio
      return $mysqli;
     }
}
?>
```

8. Yhteenveto

Suuria olioita ei juurikaan viime vuosituhannella tallennettu tietokantoihin. Tällä vuosituhannella niitä talletetaan yhä enenevässä määrin. Muistivälineiden tallennuskapasiteetit kasvavat koko ajan, laitteet pienenevät ja niiden hintakin on kohtuullinen. Automatisointi lisääntyy, yhä uusia järjestelmiä siirretään tietokoneille ja sitä kautta osin myös internetiin. Eikä muistisiruja lisätä ainoastaan koneisiin ja laitteisiin, nykyisin niitä kiinnitetään jopa kankaisiin.

Vaikka suurten olioiden tallettamismenetelmiä kehitetäänkin koko ajan, niin silti niissä on vielä paljon puutteita. Tietokantaan ei vieläkään kannattane tallettaa sellaista paljon tilaa vievää (yli 1MB) informaatiota, jota muokataan usein. Vain esimerkiksi katseltaviksi tai kuunneltaviksi tarkoitettujen videotallenteiden, kuvien, äänitteiden ja multimedian tallentaminen voi silti olla järkevää, varsinkin silloin, kun tallennus tehdään samalle työasemalle, jossa niitä myös käytetään.

Tällä hetkellä tiedostojärjestelmät pystyvät tallentamaan suuret oliot paremmin, koska niissä on yleensä kehittyneemmät menetelmät fragmentoitumisen käsittelyyn kuin tietokannoissa. Vähintään vastaavankaltaiset algoritmit pitäisi ottaa mukaan myös tietokantoihin.

Fragmentoitumista estävien algoritmien kehittymisen myötä suurten olioiden tallennus tietokantaan tulee kuitenkin ajan myötä yhä tehokkaammaksi ja kannattavammaksi.

Esimerkiksi muistin kokoa suunniteltaessa olisi otettava huomioon se, että järjestelmät toimivat sitä paremmin, mitä enemmän vapaata tilaa on käytettävissä. Säännöllinen muistin siivous ja eheytys (defragmentation) vaikuttavat suotuisasti järjestelmien suorituskykyyn.

Yleensäkin järjestelmiä suunnitellessa pitkän aikavälin toiminnallisuus pitäisi ottaa aina huomioon. Muistin vanheneminen ja sen myötä fragmentoituminen vaikuttavat sen verran dramaattisesti järjestelmien toimintakykyyn.

Viitteet

[1] Biliris Alexandros (1992) *An Efficient Database Storage Structure for Large Dynamic Objects* : http://coblitz.codeen.org:3125/citeseer.ist.psu.edu/cache/papers/cs/1764/http:zSzzSzwww.research.att.comzSz~biliriszSzpublicationszSzpaperszSz92_lo_de.pdf/biliris92efficient.pdf (26.06.2007)

[2] Carey Michael J, DeWitt David J., Graefe Goetz, Haight David M., Richardson Joel E., Schuh Daniel T., Shekita Eugene J., and Vandenberg Scott L. (1988) *The EXODUS Extensible DBMS Project: An Overview*: http://pages.cs.wisc.edu/~dewitt/includes/oodbms/exodus88.pdf (11.1.2008)

[3] Garcia-Molina Hector, Ullman Jeff, Widom Jennifer (2002) *Database Systems The Complete Book* s.13-14.

[4] IBM Informix (2005) *Large Object Data Types* internet sivut: http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.sqlr.doc/sqlrmst140.htm (5.10.2007)

[5] iWebtool (2005) *What is DBMS* internet sivu: http://www.iwebtool.com/what_is_dbms.html (15.05.2007)

[6] McKenney Mark, Pauly Alejandro, Praing Reasey & Schneider Markus (2006) *Structured Large Objects in Databases*: http://www.cise.ufl.edu/submit/files/file_368.pdf (21.4.2008)

[7] Microsoft Research (2006) *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem?* internet sivut: http://research.microsoft.com/research/pubs/view.aspx?msr_tr_id=MSR-TR-2006-45 (1.11.2007)

[8] MySQL (2002) *Blobbing Data With PHP and MySQL* internet sivut: http://www.devarticles.com/c/a/MySQL/Blobbing-Data-With-PHP-and-MySQL/ (21.4.2008)

[9] MySQL (2008) *MySQL 5.0 Reference Manual* internet sivut:
http://dev.mysql.com/doc/refman/5.0/en/index.html (13.3.2008)

[10] O2 (1998) *O2 ODMG Database System Tutorial* internet sivu:
http://www.csd.uwo.ca/courses/CS411b/pdfO2manuals/tutorial.pdf (4.1.2008)

[11] Openlinksw (1993) *Open Database Connectivity Without Compromise !* internet sivut:
http://www.openlinksw.com/info/docs/odbcwhp/tableof.htm (25.10.2007)

[12] Oracle (2003) *Oracle® Database Application Developer's Guide - Large Objects*
internet sivut: http://www.itk.ilstu.edu/docs/oracle/appdev.101/b10796.pdf (17.03.2008)

[13] Oracle (2004) *Oracle® Database JDBC Developer's Guide and Reference* internet sivut:
http://stanford.edu/dept/itss/docs/oracle/10g/java.101/b10979/toc.htm (14.04.2008)

[14] Oracle (2007) *Oracle® Multimedia User's Guide11g Release 1 (11.1*) internet sivut:
http://download-uk.oracle.com/docs/cd/B28359_01/appdev.111/b28415/toc.htm (26.5.2008)

[15] Oracle (2003) *Oracle® Database SQL Reference 10g Release 1 (10.1)* internet sivut :
http://www.softics.ru/docs/oracle10r2/server.101/b10759/toc.htm (12.5.2008)

[16] Oracle (2007) *Thread: Microsoft ODBC driver for BLOB data type* internet sivu:
http://forums.oracle.com/forums/thread.jspa?messageID=2216458 (26.3.3008)

[17] Oracle Technology Network (2007) *Oracle Multimedia* internet sivut:
http://www.oracle.com/technology/products/intermedia/index.html (14.4.2008)

[18] Oracle Wiki (2008) *ODBC* internet sivu : http://wiki.oracle.com/page/ODBC?t=anon
(14.4.2008)

[19] Oracle (2002) *Oracle9i OLAP* internet sivut:
http://www.rdbprime.com/Oracle/Oracle_Docs/Oracle9iDB_Server/olap.920/a95297.pdf
(21.4.2008)

[20] PHP manual (2008) *odbc_connect* internet sivu:
http://fi.php.net/manual/en/function.odbc-connect.php (5.5.2008)

[21] PowerBuilder Journal (2004) *Open Database Connectivity* internet sivu: http://pbdj.sys-con.com/read/106993.htm (3.12.2007)

[22] Practical PostgreSQL (2001) *Understanding SQL* internet sivu:
http://www.faqs.org/docs/ppbook/c1164.htm (15.5.2007)

[23] Rantanen Jaakko (2007) *Tiedonhallinnan_kehityslinjoja* internet sivu:
http://stud.hamk.fi/julkaisu/jaska/th00/1-1Tiedonhallinnan_kehityslinjoja.pdf (15.5.2007)

[24] Roth Consulting (2006) *Welcome to Win32::ODBC* internet sivu:
http://www.roth.net/perl/odbc/faq/ (26.3.2008)

[25] Shore (2007) *Shore - A High-Performance, Scalable, Persistent Object Repository*
internet sivut: : http://www.cs.wisc.edu/shore/ (11.3.2008)

[26] Sun Microsystems Sun Developer Network (2008) *JDBC Overview* internet sivut:
http://java.sun.com/products/jdbc/overview.html (12.03.2008)

[27] Teachtarget Holm Paul (2008) *BLOBs help you manage files from Web apps*
(http://search400.techtarget.com/tip/0,289483,sid3_gci1096413,00.html

[28] W3schools (2007) *Introduction to SQL* internet sivu:
http://www.w3schools.com/sql/sql_intro.asp (22.11.2007)

[29] W3scools (2008) *PHP Database ODBC* internet sivu
http://www.w3schools.com/php/php_db_odbc.asp (8.3.2008)

[30] Weber Keith T. (2008) *Introduction to IBM DB2 UDB (12.5.2008)*
http://giscenter.isu.edu/training/ppt/it4gis/week11-Introduction%20to%20IBM%20DB2%20UDB.ppt

[31] Webopedia (2008) *JDBC* internet sivut:
http://www.webopedia.com/TERM/J/JDBC.htm (21.4.2008)

[32] Wikipedia (2007) *Database management system* internet sivu:
http://en.wikipedia.org/wiki/Database_management_system (7.11.2007)

[33] Wikipedia (2008) *ActiveX Data Objects* internet sivu:
http://en.wikipedia.org/wiki/ActiveX_Data_Objects (5.5.2008)

[34] Wikipedia (2008) *Database connection* internet-sivu
http://en.wikipedia.org/wiki/Database_Connection (15.10.2007)

[35] Wikipedia (2008)*OLE DB* internet sivu: http://en.wikipedia.org/wiki/OLE_DB
(5.5.2008)