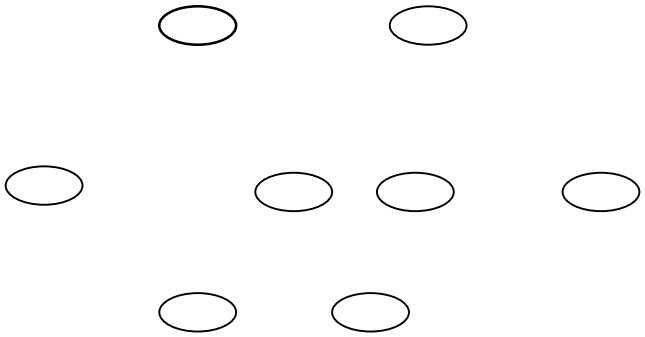
DB2-tietokantasuunnittelu



DB2-TIETOKANTASUUNNITTELU

KANTASUUNNITTELUPROSESSI

Looginen suunnittelu

Tietokannan looginen suunnittelu alkaa systeemityön määrittelyvaiheessa. Tietokannan suunnittelu voi olla tieto- tai toimintokeskeistä.

Määrittely tuottaa toimintoanalyysin tuloksena toiminnan kuvaukset. Kuvaukset voivat olla esimerkiksi tapaustaulukko, toimintohierarkia, tietovirtakaavio ja toimintomatriisi.

Määrittelyn tietoanalyysi tuottaa tietojen kuvaukset, esimerkiksi elinkaarimalli, kohdekaavio ja jokaisen yksittäisen tiedon nimen, muodon, kuvauksen ja tiedon lähteen.

Kohdemallin tarkastus

Kohdemallin pitää olla kolmannessa normaalimuodossa. Normalisointia voidaan jatkaa pidemmällekin, mutta se harvoin kannattaa. Tavallisesta poikkeava sovellus, vaikkapa tietohakemisto tms. metadatasovellus, voi vaatia hienommalle tasolle viedyn normalisoinnin. 3NM-kohdemallissa ei yleensä esiinny 1/1-suhteita.

Denormalisointia ei tässä vaiheessa tehdä.

Kohdemallin tarkastamiseksi pitää tarkastajan tuntea sovellusaluetta jonkin verran ja mukana pitää olla sovellusalueen asiantuntija ja tekijöitä. Kohdemalli tarkastetaan esikatselmuksessa, joka päättää määrittelyvaiheen. Kohdemallista pitää tarkastaa, että

- jokainen kohde jota tarvitaan, on mukana
- jokaisella kohteella on kuvaus
- mukana ei ole tarpeettomia kohteita tai esimerkiksi johdettuja tietoja
- kohdemalli on kolmannessa normaalimuodossa
- kaikki toimintoanalyysin tuottamien toimintojen tiedot löytyvät kohdemallista
- kaikkia kohteita käsitellään jossain toiminnossa (= tieto lisätään, päivitetään, poistetaan ja haetaan ainakin yhdessä toiminnossa)
- yksilöivä avain on todella yksilöivä eikä avaimesta voi poistaa yhtään tietoa ilman että yksilöivyys kärsii
- kohteet ovat riittävän yleisiä. Esimerkiksi taulut KUORMA_AUTO, LINJA_AUTO ja HENKILO_AUTO voidaan yhdistää tauluksi AJONEUVO.

Rakennetaan 0-tason relaatiomalli (relaatiokaavio), joka sisältää yksikäsitteiset avaimet, vierasavaimet ja kohteiden ja suhteiden lukumäärätiedot.

Tätä kaaviota vastaan tarkastetaan kaikki tietotarpeet (=toiminnot):

- tarvittavat hakemistot lisätään
- taulujen järjestysvaatimusten mukaan päätetään mikä on järjestystoivomus (tämä on päätettävissä lopullisesti vasta teknisessä suunnittelussa)
- lasketaan QUBE (vastausajan pikaennuste) ; jos asetettuun vastausaikavaatimukseen ei päästä, voidaan vastausaika laskea tarkemmalla kaavalla ja päättää voidaanko tapahtuman auttamiseksi tehdä jotain
    - parempi indeksi, Index Screening tai Index Only -haku
    - taulu toiseen järjestykseen
    - tietotarpeen toteuttaminen toisella tavalla (= uudelleensuunnittelu)
    - tiedon pakkaaminen I/On vähentämiseksi
    - denormalisointi

Näistä tiedon pakkaaminen ja denormalisointipäätökset kuuluvat vasta tekniseen suunnitteluun, mutta jos tarve jätetään dokumentoimatta ja välittämättä tekniselle suunnittelulle, ei sitä kukaan arvaa ajatella. Teknisessä suunnittelussa voidaan käyttää paljon muitakin tapoja nopeuttaa tapahtumaa tai eräajoa (puskurien kasvatus, hiperpoolit, levy I/On nopeutus).

Normalisointi

Kolmas normaalimuoto tarkoittaa, että

- Toistuvat tietoryhmät on erotettu omaksi kohteekseen tai monistettu riveiksi
- Kohteen tiedot riippuvat koko avaimesta ja vain avaimesta
- N:M -yhteydet on purettu 1:N -yhteydeksi

Taulu ei siis voi sisältää toista taulua, tai kahta tai useampaa kertaa samaa tietoa. Onko esimerkiksi kotipuhelin ja työpuhelin sama tieto eli oma kohteensa vai ei ? Periaatteessa se on oma kohteensa PUHELINNUMERO, jolloin ei ole väliä onko asiakkaalla yksi vai kymmenen numeroa.

Kohteen tiedot riippuvat avaimesta, koko avaimesta ja vain avaimesta tarkoittaa siis, että tieto ei saa olla riippuvainen avaimen osasta tai avaimeen kuulumattomista tiedoista. (Huomaa, että tilakoodi-tyyppiset tiedot eivät voi olla normalisoidun kohdemallin tietoja, koska tieto on johdettua eikä riipu avaimesta lainkaan).

Tähän toiseen sääntöön tuo poikkeamismahdollisuuden TABLE CHECK CONSTRAINT. Tämän eheyssäännön avulla voidaan tehdä arvoaluetarkistuksia (domain) ja tarkistuksia saman taulun sarakkeiden välille. Esimerkiksi palkkio ei voi olla suurempi kuin palkka tai veroprosentti ei voi olla suurempi kuin lisäveroprosentti.

N:M -yhteydet puretaan teknisellä kohteella, jossa on vain kohteet yhdistävät avaimet. Joskus tämä tekninen kohde voi muodostua aidoksi kohteeksi, jolloin se saa omia tietoja. Esimerkiksi MYYJÄ- ja TUOTE-kohteen välille tuleva kohde KAUPPA.

1:1 -suhde voi olla myös merkki normalisointivirheestä. Varsinkin siinä tapauksessa, että molemmilla kohteilla on sama yksilöivä avain ja kenties samat suhteetkin. Mieti tarkkaan onko kyseessä kaksi kohdetta vai yhden kohteen ominaisuudet jaettuna kahteen eri joukkoon.

Denormalisointi

Denormalisointi voidaan jakaa karkeasti kolmeen tyyppiin

- Tiedon monistaminen moneen tauluun
- Summatun tiedon käyttö
- Kohteiden yhdistäminen

Tiedon monistaminen on syytä turvata triggereillä heti kun ne ovat tiedonhallintaohjelmiston piirteenä mukana. Tätä ennen on tehtävä ohjelma, joka tarkastaa tiedon eheyden (esimerkiksi asiakkaan nimen muutos pitää muistaa päivittää kaikkiin tauluihin joihin tieto on monistettu). Vaikka triggerit ovatkin käytössä, voidaan triggereillä suojattu tiedon eheys saada rikkoutumaan esimerkiksi LOAD-ajolla tai tekemällä taululle RECOVER TO COPY tai RECOVER TO RBA.

Summatun tiedon käyttämisessä on ehdottoman tärkeää, että lähdetiedot ovat jossain tallessa, peräkkäistiedostossa ellei taulussa. On tehtävä ohjelma joka tarkastaa, että summatut tiedot ovat oikein (huomaa peruutukset). Tämänkin denormalisoinnin käyttö vaatii oikeastaan triggereiden käyttöä. Mutta mikään ei pelasta siltä, että lähdetietoa ei saa hukata.

Kohteiden yhdistämistä ei pidä tehdä tapahtumakantaan. Kohteita yhdistämällä menetetään kaikki SQL:n keinot käsitellä tietoa. Esimerkiksi taulusta, jossa sarakkeet ovat MATKUSTAJA1, MATKUSTAJA2, …, MATKUSTAJAN, on tietyn matkustajan hakeminen SQL:llä erittäin hankalaa. I/On määrän vähennys ei ole syy kohteiden yhdistelyyn: I/Ota voidaan vähentää monella muullakin tavalla, teknisen suunnittelun keinoin.

Jos kaikista varoituksista huolimatta kuitenkin päädyt tähän ratkaisuun, tee ensimmäiseksi suunnitelma ratkaisun purkamiseksi. Jos tällaista oikein toimivaa purkuratkaisua ei pystytä tekemään, ei pidä missään tapauksessa myöskään denormalisoida.

Tietokanta elää ikuisesti. Sovellus sen päällä voi muuttua tai vaihtua, mutta jos tietokanta on suunniteltu oikein, se ei muutu. Uusia kohteita voi tulla ja uusia tietoja, mutta perusrakenne säilyy. Teknisiä ratkaisuja voidaan muuttaa, mutta malli on "ikuinen".

Viite-eheydet

Viite-eheys voidaan määritellä aina kun yhteys kahden kohteen välillä on 1:N tai 1:NC (ehdollinen yhteys). Jos yhteys on ehdollinen molempiin suuntiin 1C:NC, on ainoa mahdollinen poistosääntö SET NULL.

Viite-eheys kannattaa määritellä aina kun se on mahdollinen. Lapsitauluun määritellään viiteavain (foreign key) ja viiteavain hakemisto, joka on samanlainen, tai jonka alkuosa on samanlainen, kuin isä-taulun avain (parent key).

Viite-eheyttä ei kuitenkaan pidä määritellä parametri-tyyppisen taulun ja datataulun välille, koska RECOVERY-kokonaisuuksiin ei ole syytä ottaa mukaan turhia tauluja, esimerkiksi parametri-taulun muita lapsitauluja.

Standardit

Käytä nimeämisessä standardia. Varsinkin tietojen nimien, muotojen ja pituuksien pitää olla samanlaiset joka paikassa. Ellei ympäristössäsi ole käytössä tietohakemistoa tai standardeja, pidä huoli että ainakin sovelluksen sisällä käytetään samoja määrittelyjä.

Jokainen tieto on potentiaalinen yleinen tieto (= sitä käytetään useammassa kuin yhdessä sovelluksessa) ja siksi kannattaa miettiä sille kuvaava nimi sekä sellainen muoto ja pituus, jotka sopivat sen kaikkiin mahdollisiin käyttötapoihin. Tiedonhallintajärjestelmissä on tavallisesti käytössä hyvinkin pitkät nimet (DB2 18 merkkiä), jolloin on turha typistää nimiä 8 merkin lyhennyksiksi, joita kukaan ei ymmärrä (esimerkiksi TPPTKKD, onko se joku koodi ja jos niin mikä?)

Tietojen samanmuotoisuus takaa yhdistettävyyden. Jos asiakastunnus on toisissa kohteissa CHAR 11 ja toisissa kohteissa CHAR 15 ja vielä jossain DEC 11, 0, niin millä nämä tiedot voi yhdistää. Näiden tietojen yhdistäminen voi olla vaikeaa.

Näiden standardien kehittäminen ja käyttäminen vaatii vaivannäköä, mutta maksaa itsensä nopeasti takaisin kehitystyössä.

Tietokannan fyysinen suunnittelu

Perusmäärittelyt

DATABASE

- Määrittele sovelluksen yhteenkuuluvat taulutilat samaan tietokantaan.

TABLESPACE

- Määrittele taulutilaan yksi taulu, koska monet toiminnot tapahtuvat taulutilatasolla.
- Määrittele taulutila segmentoiduksi tai partitioiduksi.
- Älä määrittele simple-taulutiloja, koska niiden tekniset ominaisuudet eivät ole hyviä ja ne saattavat myöhemmissä versioissa poistua käytöstä.

TABLE

- Harkitse surrogaatti-avainten käyttöä, koska ne
  - vähentävät avaimen päivitystarvetta
  - lyhentävät avainta
- Määrittele perusavain (valvoo kohde-eheyttä)
- Huomioi rivin sarakkeiden määrittelyssä
  - kiinteämittaiset rivit
    - määrittele vierekkäin ne sarakkeet, joissa tieto on eniten muuttuvaa (lokille kirjoitus ensimmäisestä muutoksesta viimeiseen)
  - vaihtuvamittaiset rivit (tauluun määritelty VARCHAR-sarakkeita)
    - määrittele loppuun ne sarakkeet, joiden tieto on muuttuvaa
  - FIELDPROC-exitin käyttö
    - tarkista, että toimii tarkoitetulla tavalla (varsinkin lajittelu)
- Määrittele viiteyhteydet
  - älä määrittele yhteyksiä tietokannasta toiseen, jos et ole varma vaikutuksista
  - älä määrittele viiteyhteyksiä koodi- ja data-taulujen välille (koodien sallitut arvot kannattaa tarkistaa CHECK CONSTRAINT -säännöillä. Kooditauluilla ei myöskään yleensä ole RECOVERY-tarvetta)
- Määrittele CHECK-säännöt
  - sääntöjen ei ole todettu aiheuttavan merkittävää lisäkuormaa
  - mieti etukäteen sääntöjen muuttumisesta aiheutuvat vaikutukset ja suunnittele jatkotoimenpiteet
- Mieti tarvittavat leimasarakkeet
  - mikä on sarakkeiden käyttötarkoitus (esimerkiksi päivityksen valvonta, selvitys)
- Aseta taululle WITH RESTRICT ON DROP -vipu (myös testiympäristöissä), jotta vahingossa tapahtuvilta taulumääritysten tuhoamisilta vältytään.

COLUMN

- Mieti NULL-arvon tarpeellisuus kullekin sarakkeelle.
- Mieti päivämäärien alku- yms. arvot. Esimerkiksi arvot ’01.01.0001' ja ’31.12.9999' voivat osoittautua ongelmallisiksi replikoinnissa toiseen relaatiotietokantajärjestelmään (esimerkiksi SQL Server) ja joidenkin ohjelmointikielien (esimerkiksi Java) kanssa. Näiden päivämäärien sijaan tulisi käyttää NULL-arvoa.
- CHAR/VARCHAR-päätös riippuu myös siitä, missä ympäristössä ja millä kielellä saraketta käytetään. Käytä vaihtuvamittaisia sarakkeita silloin, kun vaihtuvamittaisuudella on merkitystä. Perinteisessä sovellusympäristössä perinteisillä kielillä (esimerkiksi Cobol) ohjelmoitaessa pituuden vaihtelun pitää olla suuri ennen kuin vaihtuvamittaisuuden hyödyt voittavat ylläpidon vaivat. Dynaamisen SQL:n ympäristöissä tiedon muokkaus on vaikeampaa ja niissä vaihtelevan mittaiset tiedot pitäisi myös määritellä vaihtuvamittaisiksi:

  Virtanen , Ville Velmeri
  Virtanen, Ville Velmeri

COMMENTS, LABELS

- Suosi kommentointia, jolloin DB2-katalogi palvelee paremmin myös tietohakemistotyyppisesti.

INDEX

- Päätä C-hakemisto
  - huomioi partitiointitarve (partitioiva avain aina C-avain)
  - huomioi eräkäsittelyn järjestystarve (eräkäsittelyn oltava peräkkäiskäsittelyä)
  - huomioi keskeisin selausjärjestys
  - huomioi, että partitioivan hakemiston muutokset (sarakkeen lisäys, Unique-määritys yms.) vaativat taulutilan droppaamisen
- Määrittele hakemisto aina Unique-määreellä, jos hakemistorivit ovat yksikäsitteisiä.
- Muista UNIQUE WHERE NOT NULL -mahdollisuus silloin kun hakemistosarakkeelle on NULL-arvo mahdollinen.
- Suunnittele hakemistosarakkeiden järjestys hakutarpeiden pohjalta
  - ASC, DESC (esimerkiksi Pvm, Jno -tiedoista yleensä uusimmat kiinnostavimpia)
- Muista aina määritellä viiteavainhakemistot, koska ilman sitä isärivin poistaminen aiheuttaa aina TS Scanin lapsitauluun.
- Vältä VARCHAR-saraketta hakemistosarakkeena.
  - VARCHAR-sarake on hakemistossa aina maksimipituisena
  - älä määrittele VARCHAR-saraketta Unique-hakemistoon, koska syntyy helposti tupla-arvoja (ns. space-lipsahdukset)
- Vältä loppuun kasvavaa hakemistoa, vaikka indeksi ei enää pidäkään lukkoja eikä varaa pelkkiä puolikkaita sivuja.
- Huomioi NULL-arvojen käyttäytyminen ASC- ja DESC-indekseissä. ASC-järjestyksessä NULL-arvo on viimeisenä ja DESC-järjestyksessä ensimmäisenä.

VIEW

- Suosi moduulikohtaisia näkymiä tietoriippumattomuuden ja ylläpidettävyyden takia.
- Luettele näkymän sarakkeet käytön mukaiseen (käyttäjän haluamaan) järjestykseen (teknisen suunnittelun tuloksena sarakkeet ovat taulussa siinä järjestyksessä kuin fyysisesti on parasta).
- Laita näkymän WHERE-osaan vain rajoittavat ehdot, jolloin muutos ei välttämättä vaikuta ohjelmaan.
- Käytä WITH CHECK - optionia niiden näkymien kanssa, joiden kautta voi päivittää.

Tekninen suunnittelu

Lukitukset

- Lukituskoko
  - ANY on DB2:n oletus, mutta DB2 valitsee tavallisesti PAGE-lukitustason. Jos lukkoja tulee paljon niin syntyy lukituksen eskalointi, jolloin lukitus nousee TABLESPACE-tasoiseksi.
  - PAGE-lukituksessa eskalointia ei tapahdu. Jos lukkoja tulee liikaa niin suoritus keskeytyy.
  - TABLESPACE/TABLE-lukituksessa kyseiseen kohteeseen ei ole yhtäaikaista pääsyä
  - ROW-rivilukitus kasvattaa yhtäaikaisten lukkojen määrää moninkertaiseksi. Rivilukitus ei ole ratkaisu deadlock-ongelmiin, vaan ne on ratkaistava ohjelmansuunnittelulla (päivitysjärjestys). Rivilukituksella samanaikaisuus lisääntyy ja timeoutit vähenevät.
- Käsittely pitäisi pystyä hajauttamaan tasaisesti avaimen mukaan, jotta kuumia sivuja ei pääse syntymään.

Partitiointi

- Valitse partitioiva tekijä huolellisesti, koska C-hakemisto määritellään tämän mukaan ja partitioivan tekijän muuttaminen on hankalaa.
- Partitioitujen taulutilojen partitioimattomat hakemistot saattavat kasvaa yllättävän suuriksi
  - isojen hakemistojen (tyypillisesti partitioidun taulutilan partitioimaton hakemisto) luonnissa työtila saattaa loppua. Luo hakemisto tällöin DEFER YES ja REBUILD INDEX -menettelyllä.
- Huomioi tilasuunnittelu (mitä pystytään muuttamaan, jos joku partitio alkaa kasvaa odottamattomasti)
  - versiossa 6 on mahdollista muuttaa partitiorajoja (Limit Key) ALTER-komennolla
  - huomioi, että uusia partitioita ei saa määriteltyä lisää (harkitse tyhjien partitioiden määrittelyä väliin)
  - Huomioi, että samanaikaisesti aukiolevien tiedostojen määrä kasvaa.

Käytettävyys 24 tuntia

Yleistä

24-tunnin ympäristössä on muistettava, että puhumme aina noin 24-tunnin ympäristöstä. Rajoituksia syntyy koneen huoltokatkoista ja DB2:n katkoista. Sovellusten 24-tunnin käytettävyyteen liittyy myös muiden järjestelmien katkot, kuten CICS. SYSPLEX- ja Data Sharing -ympäristöissä toimittaessa katkojen tarve vähenee entisestään. Mahdolliset RECOVERY-tilanteet aiheuttavat aina katkon. LOAD-ajot ovat käytännössä mahdottomia 24-tunnin ympäristössä. Kannat on pyrittävä suunnittelemaan siten, että katkot kohdistuvat osaan kannasta tai taulutilasta. Voimme siis pyrkiä lähes 24-tunnin/7-päivää viikossa käytettävyyteen.
Mietittäessä 24-tunnin ympäristöjen rakentamista, on myös pidettävä mielessä, mitä lähempänä 24-tunnin ympäristöä olemme, sitä kalliimpia ja monimutkaisempia järjestelmiä rakennamme.
Nämä tosiasiat on tehtävä selviksi sovellusten tilaajalle.

Looginen suunnittelu

24-tunnin ympäristö on otettava huomioon heti suunnittelun alkuvaiheessa, jälkeenpäin sen toteuttaminen on työlästä.

Yleensä 24-tunnin vaatimuksessa on vain pieni osa dataa, joka on todella tarpeen olla käytettävissä vuorokauden ympäri. Kantoja suunniteltaessa olisi hyvä lähtökohta erottaa toisistaan todellinen 24-tunnin data ja muu data. Pyritään pitämään 24-tunnin tietoja sisältävät taulut , taulujen lukumäärä mahdollisimman pieninä. Joissain tilanteissa voidaan muu data siirtää vastaaviin historiatauluihin, joita voidaan vapaasti käsitellä ja siivota yöaikaan. Historiataulujen käyttö on suositeltavaa, jos dataan ei kohdistu enää varsinaista käsittelyä. Historiointiin voi käyttää arkistointia, sillä vastausaikavaatimus historiatiedoille ei yleensä ole heti.

Koko kanta kaikkine yksityiskohtaisine tietoineen ei aina tarvitse olla 24-tuntia käytössä. Voidaan rakentaa yön tarpeita varten oma versionsa sovelluksesta.

Tekninen suunnittelu

Kantojen kopiointi hoidetaan aina shrlevel change muodossa ja perään ajetaan QUIESCE . Kopiointi hoidetaan sovelluksen kannalta mahdollisimman hiljaiseen aikaan. RUNSTATS-ajot voidaan myös ajaa shrlevel change optiolla, jolloin samanaikainen käsittely on mahdollista. 24:n tunnin taulujen suunnittelussa partitiointi on oleellinen asia, jotta pystytään käyttämään hyväksi partitioiden rinnakkaiskäsittelyä. ONLINE REORG voi lyhentää kannan huoltokatkoja.

Voidaan tehdä aputauluja, joiden avulla tietyt taulut voidaan ottaa pois 24-tunnin ympäristöstä ja hoitaa eräajoina yöllä. Kannattaa hyödyntää huoltokatkoaikoja erätarpeisiin. Nämä ratkaisut monimutkaistavat jonkin verran sovellusta, mutta helpottavat huomattavasti 24-tunnin vaatimuksen toteuttamista

Tilasuunnittelu

- Vapaan tilan määrä
  - taulutila ei saa kasvaa loppuun (lukitukset kasaantuvat; lukko-odotukset, läpimenoajat ja deadlockien määrä kasvavat)
  - lisäysten ja päivitysten hajauduttava tasaisesti (organisointitarve vähenee)
  - käytä FREEPAGE=0 (jos et käytä niin muista, että ellei prefetchin trigger-sivulla käydä, joudutaan I/O:ta odottamaan; jos trigger-sivu on tyhjä, sillä ei koskaan käydä) ja satsaa enemmän PCTFREE-arvojen suunnitteluun
- Tiivistys (Compress)
  - käytä HW-pakkausta
  - voit harkita tiivistämättä jättämistä, jos lisäyksiä/päivityksiä paljon
  - tarkkaile tiivistysprosenttia ja FARINDREF sekä NEARINDREF -arvojen nousua, koska tiivistyshakemisto (Dictionary) menee helposti huonoon kuntoon
  - jos tieto aiotaan tiivistää, ei VARCHAR-tietomuotoa kannata käyttää tilansäästämiseksi

Sovellussuunnittelu

- Tietotarpeen WHERE-ehdon ja ORDER BY -määreen tulisi kulkea saman hakemiston kautta (huomioi, että hakupolut voivat muuttua tilastotietojen muuttuessa!).
- Käsittelyjärjestyksellä pystyt välttämään deadlockeja. Viite-eheysmääritykset määräävät pitkälti päivitysjärjestyksen ja jos niitä ei käytetä niin käsittelyjärjestys on suunniteltava ja sovittava asia.
- Muista hakupolkujen tarkistusten (Explain) sekä CPU- ja vastausaikojen merkitys välttämättöminä tarkistuspisteinä.
- Näkymän (VIEW) monipuolista käyttöä suositellaan
- Tietojen samanlainen talletustapa eli mikäli tieto ei "täytä" koko kenttää, millä muu osa täytetään. Esimerkiksi tieto on määritelty CHAR(20) ja jossain tapauksessa tieto onkin vain kuusi merkkiä pitkä ; talletetaanko kuusi merkkiä ja 14 tyhjää vai päinvastoin ?

Tietokannan hoito

Hoitosuunnitelma

- Määrittele perusperiaatteet
  - vastuut henkilöitävä
  - varmistuskopiot
    - suunnittele RECOVERY-kokonaisuudet (esimerkiksi QUIESCE-pisteet)
    - kopioita pitää ottaa (koko RECOVERY-kokonaisuus kerralla tai ainakin QUIESCE)
    - määrittele varmistuskopioiden ottotiheys ja -laajuus (full/partial)
  - statistiikka
    - RUNSTATS-ajoja pitää ajaa
    - ajotiheys määriteltävä
    - muista monen eri sarakkeen käyttömahdollisuus (first + keycard)
  - LOAD- ja REORG-ajojen jälkeiset toimenpiteet
    - COPY- ja QUIESCE-ajot (tällöin kopio ei estä käyttöä, palautukset täytyy tehdä aina RBA:han, ei kopioon).
    - RUNSTATS- ja CHECK-ajot
- Organisointiajot
  - voi tehdä ONLINE REORG -ajona
  - huomioi levytilan tarve
  - aja organisointi hiljaisimpaan mahdolliseen aikaan (ONLINE REORG)
  - jos organisointi perustuu tilastotietoihin, muista ajaa RUNSTATS-ajo organisoinnin jälkeen
- Indeksien organisointi
  - huonossa järjestyksessä olevan indeksin peräkkäiskäsittely vie enemmän aikaa kuin hyvässä järjestyksessä olevan
  - lisäykset nopeutuvat, kun sivuilla on tyhjää tilaa (eikä tarvitse tehdä page splittejä)
- Tarkkaile indeksin tasojen lukumäärää tilankäyttö- ja I/O-mielessä

Palautussuunnitelmat

- "Normaali" palautussuunnitelma
  - määrittele toimenpiteet, jotka tehdään esimerkiksi levyrikon yhteydessä
- Eräajovirheen (ohjelma, syöttöaineisto) aiheuttama palautustarve
  - kannassa on oltava riittävästi tietoa, jonka avulla peruutus voidaan tehdä (rivin valintakriteeri ei saa muuttua ajossa tai muuttuneet rivit on löydettävä aukottomasti muun tiedon, esimerkiksi muutosaikaleiman, avulla)

Seuranta

- Mittarien kehittäminen seurattavia asioita varten
  - tilankäyttö
  - statistiikkatiedot
  - puuttuvat (viiteavain) hakemistot
  - puuttuvat varmistuskopiot (taulutilat, joita ei varmisteta)
  - saantipolut
  - lukitukset
  - vastaus-/CPU-ajat