

SQL-KIELEN PERUSTEET

© JOUNI HUOTARI 1999-2011

KALVOT PERUSTUVAT PÄÄOSIN ARI HOVIN
SQL-OPPAASEEN (DOCENDO 2004)

1. VERSIO: TAPANI ÄIJÄNEN



1. JOHDANTO

HUOM. RELAATIOMALLI YM. TIETOKANTOJEN
PERUSTEET ON ESITETTY ERILLISILLÄ KALVOILLA



SQL - STRUCTURED QUERY LANGUAGE

- SQL on tietokantojen käsittelyyn kehitetty kieli
- Yleisimmät kielellä hoidettavat toiminnot:
 - Tietokannan rakenteen määrittely ja muuttaminen
 - Kyselyt tietokantaan ja laskenta tietokannan datalla
 - Tietojen ylläpito: lisäykset, muutokset ja poistot
 - Valtuuksien käsittely
 - Tapahtumankäsittelyn ohjaaminen
- Lisäksi SQL:n käyttö ohjelmoinnissa
 - ”Upotettu” SQL; SQL:n käyttö muiden kielten sisältä
 - API-rajapinnat ohjelmointikieliin



SQL

- SQL on luonteeltaan ei-proseduraalinen kieli, jossa kerrotaan **mitä** halutaan tehdä (ei miten)
- SQL perustuu relaatioalgebraan ja joukko-oppiin
- SQL:stä useita versioita, mm. SQL-92 eli SQL2, SQL:1999 eli SQL3, SQL:2003 ja SQL:2006
- Kielestä on eri valmistajilla toisistaan poikkeavia versioita (murteita)
- SQL-kieltä voidaan käyttää useissa SQL-tuotteissa vuorovaikutteisesti antamalla SQL-käskyjä omaan ikkunaan, jolloin tulokset saadaan välittömästi (toiseen ikkunaan)

SQL-KIELEN HISTORIAA

- Alun perin SEQUEL (Structured English QUery Language)
- 1. versio kehitettiin IBM:n tutkimuskeskuksessa 1970-luvulla (E.F. Coddin relaatiomalliin pohjautuen)
- 1. kaupallinen versio esiteltiin v. 1979 (Oracle)
- SQL:n ANSI-standardi julkistettiin v. 1986
- 1989 lisättiin viite-eheysmäärittelyt
- 1992 liitossyntaksi, tapahtumanhallinta ym.
- 1999 mm. herättimet, talletetut proseduurit ja oliolaajennuksia + CLI (Call Level Interface)
- 2003 ja 2006 lisää olio-ominaisuuksia, XML ym.

NÄISSÄ KALVOISSA ESIINTYVÄT SQL-KÄSKYT

- Noudattavat pääosin SQL2-standardia
- Suurin osa on testattu Oraclella, OCELOTilla, MS SQL Serverillä ja MS Accessilla; näistä vain OCELOT on täysin SQL-standardin (SQL2 ja osin SQL3) mukainen
- Kalvoissa on mainittu, jos käskystä on jokin standardista poikkeava tuotekohtainen ratkaisu
- Rakenne (käskyjen esitysjärjestys) on Ari Hovin kirjaa mukaileva

DDL JA DML

- Data Definition Language (DDL): tietokannan rakenteen määrittely
 - Perus-SQL:ssä CREATE, ALTER ja DROP TABLE, CREATE ja DROP VIEW, CREATE ja DROP INDEX
 - Käyttökelpoisia esimerkiksi silloin kun on tarve luoda uudestaan taulut ja sarakkeet
 - Lisäksi mm. auktorisointikäskyt GRANT ja REVOKE
 - Data Manipulation Language (DML): tietokannan käsittely
 - Kyselyihin SELECT, koostefunktiot (mm. SUM ja COUNT) ym.
 - Tietojen ylläpitoon UPDATE, DELETE ja INSERT
 - Muutosten vahvistus ja peruutus (COMMIT ja ROLLBACK)
 - Lisäksi eri RDBMS:t sis. ohjauskäskyjä mm. transaktioille, sessiolle ja järjestelmälle, esim. ANALYZE, AUDIT ja COMMENT
- => ks. esimerkkejä tuotekohtaisista eroista: <http://troels.arvin.dk/db/rdbms/>



SQL OHJELMOINNISSA

- ”Upotettu” SQL: SQL-käskyt muiden kielten sisältä
 - SQL-käskyt voidaan laittaa ohjelmiin sellaisenaan
 - ohjelma linkitetään relaatiokantaan API-rajapinnoilla (Application Programming Interface), esim. ODBC ja JDBC
- Dynaaminen SQL: (jonkin muuttujan arvoksi sijoitettu) SQL-käsky käännetään ja suoritetaan ajoaikana

2. PERUSKYSELYT

SELECT-KÄSKYN ERI MUODOT:

- JOHDANTO KYSELYIHIN
- FUNKTIOT
- LIITOKSET
- YHDISTE
- ALIKYSELYT



SELECT-KÄSKY

- SQL-kielen keskeinen toiminto on kyselyiden tekeminen tietokantaan
- Kyselyissä SQL-kielen perusrakenne käsittää kolme avainsanaa, joista SELECT ja FROM ovat pakollisia:
- SELECT <attribuuttilista> valitsee listan mukaiset sarakkeet (attribuutit) kyselyyn
- FROM <taululista> määrittelee haettavat taulut
- WHERE <ehdot> määrittelee ehdot haettaville riveille (monikoille)
 - SELECT * hakee kaikki sarakkeet
 - SELECT ilman WHERE-ehtoa hakee kaikki rivit

SQL-KÄSKYN ESITYSTAPA JA RAKENNE

- SQL-kyselyn yleinen muoto:

```
SELECT c1 [, c2, ... cn]  -- valitut sarakkeet
FROM t1 [, t2, ... tm]    -- valitut taulut
WHERE [e1]                -- kyselyn ehto
GROUP BY [cx]             -- ryhmittelyehto
HAVING [e2]              -- ryhmittelyn hakuehto
ORDER BY [cy]            -- lajitteluehto
```

-- Kommentit alkavat yleensä kahdella viivalla

/* Tällä tavoin voidaan merkitä pidempi kommentti monissa tuotteissa (mm. MySQL, Oracle, SQL Server) */

- Monissa tuotteissa käskyn lopussa on puolipiste, mikä kertoo tuotteelle käskyn päättymisestä



SQL-ESIMERKKEJÄ

- Hae kaikki henkilöt, jotka ovat syntyneet v. 1975:

```
SELECT *  
FROM henkilo  
WHERE syntvuosi = 1975
```

- DISTINCT, mikäli mahdolliset toistot halutaan poistaa:

```
SELECT DISTINCT myyja FROM tuote
```

- SELECT-lause voi sisältää sarakkeisiin kohdistuvia operaatioita (laskentaa) ja sarakkeelle voidaan antaa nk. Aliasnimi (AS-sidesanalla), esim.

```
– SELECT hinta, hinta * 1.23 AS HintaSisALV, TuoteNimi  
FROM tuote
```

- Monissa tuotteissa AS-sidesana voidaan jättää pois



SQL-HAKUESIMERKKKEJÄ; LAJITTELU

- Haetaan kaikki tuotetiedot, lajittelu nimen mukaan:
`SELECT *`
`FROM tuote`
`ORDER BY TuoteNimi` -- oletus = nouseva eli ASC (ASCENDING)
 -- laskeva = DESC (DESCENDING)
- Kysely, joka lajittelee ensisijaisesti sukunimen mukaan ja toissijaisesti etunimen mukaan, mutta näyttää kuitenkin ensin etunimen; kolmantena lajitteluperusteena palkka (suurin palkka ensin):
 - `SELECT etunimi, sukunimi, kunta, tutkinto, palkka`
`FROM henkilo`
`ORDER BY sukunimi, etunimi, palkka DESC`



RIVIEN VALINTA: WHERE-EHTO

- WHERE-ehto voi sisältää
 - vertailuoperaattoreita <, >, <>, <=, >=, !=, !>, !<)
 - BETWEEN -määreen
 - IN -määreen
 - LIKE -määreen
 - AND, OR ja NOT -operaattoreita
 - IS NULL tai IS NOT NULL -ehdon
- Merkkijonohaut:
 - % = mikä tahansa merkkijono (Accessissa *)
 - _ = joku merkki (Accessissa ?)
 - Kirjainkoolla on väliä (paitsi MS:n tuotteissa ja MySQL:ssä)



WHERE-ESIMERKKEJÄ

```
SELECT tuote_nimi, hinta  
FROM tuote  
WHERE hinta BETWEEN 100 AND 1000
```

-- kaikki tuotenimet niistä tuoteryhmistä, joissa esiintyy sana "levy":

```
SELECT tuote_nimi FROM tuote  
WHERE tuote_ryhma LIKE '%levy%'
```

-- ehtona on, ettei tuoteryhmä kuulu tulostimiin tai skannereihin:

```
SELECT tuote_nimi, hinta FROM tuote  
WHERE tuote_ryhma NOT IN ('tulostimet', 'skannerit')
```



NULL-ARVOT

- Puuttuva, tuntematon arvo (ei siis välilyönti tms.)
- Haussa käytettävä IS-määreitä:
`WHERE sarake IS [NOT] NULL`
- Esimerkiksi jos haetaan kaikki ne tuotteet (nimi ja hinta), joilla on jokin nimi:
`SELECT tuote_nimi, hinta`
`FROM tuote`
`WHERE tuote_nimi IS NOT NULL`
- COALESCE-funktiolla voidaan testata, onko sarakkeen arvo NULL ja korvata NULL jollain tekstillä tai nolalla, esim.
`COALESCE(sarake, 0)`

FUNKTIOT



KOOSTEFUNKTIOT

- Viisi standardifunktiota:
 - AVG -- keskiarvo
 - MIN -- pienin arvo
 - MAX -- suurin arvo
 - SUM -- summa
 - COUNT -- lukumäärä
- Esim. montako henkilöä on henkilö-taulussa:
 - `SELECT Count(*) AS Lkm
FROM henkilö`

KOOSTEFUNKTIOIDEN KÄYTTÖ

- Koostefunktioita käytetään SELECT-lauseen sarakeluettelossa ja HAVING-lauseessa

```
SELECT MAX(hinta), MIN(hinta)      -- suurin ja pienin hinta
FROM tuote
```

- WHERE-lauseessa koostefunktioita ei voi käyttää (vaan ne on laitettava HAVING-lauseeseen), esim. Haetaan myyjät, joiden myymien tuotteiden hinta yhteensä on yli 5000:

```
SELECT myyja, SUM(hinta)           -- tuotteiden hintojen summa
FROM tuote
GROUP BY myyja                    -- ryhmittely myyjän mukaan
HAVING SUM(hinta) > 5000
```

- DISTINCT-rajaus (toimii muissa tuotteissa paitsi Accessissa):

```
SELECT COUNT(DISTINCT city)        -- monestako kaupungista
FROM authors                       -- kirjailijoita
```

RYHMITTELY - GROUP BY

- GROUP BY -ehdolla tiedot ryhmitellään jonkin sarakkeen tai sarakkeiden mukaan ryhmiin
 - GROUP BY -ehdon ryhmiin kohdistetaan yleensä jokin summafunktio
 - kaikki SELECT-listan lauseet, joissa ei ole funktiota, on lueteltava GROUP BY –lauseessa!
- Esim. kunkin myyjän tuotteiden hintojen keskiarvo:
`SELECT myyja, AVG(hinta)`
`FROM tuote`
`GROUP BY myyja`
- Kunkin myyjän myymien tuotteiden määrä
`SELECT myyja, COUNT(myyja)`
`FROM tuote`
`GROUP BY myyja`

RYHMITTELY - HAVING

- Sekä HAVING- että WHERE-lauseella voidaan ryhmittelyn tulosta rajata:

```
SELECT myyja, COUNT(*)      -- myös NULL kelpaa
FROM tuote
WHERE tuoteHinta > 100
GROUP BY myyja              -- hae myyjät joilla
HAVING COUNT(*) > 2         -- vähintään 3 tuotetta
```

- **Tulosjoukkoa voidaan siis rajata eri tasoilla:**
 - rajataan ensiksi ryhmiteltävä joukko WHERE-lausekkeella
 - tästä syntynyt tulos ryhmitellään GROUP BY -määrittelyllä
 - lopuksi ryhmiteltyä joukkoa voidaan vielä rajata HAVING-lauseella



MERKKIJONOFUNKTIOITA

- Merkkitiedon käsittely

- standardissa merkkijono saadaan funktiolla SUBSTRING(mjono FROM mista FOR pituus)

```
SELECT sukunimi, ' ', SUBSTRING(etunimi FROM 1 FOR 1), ' '  
FROM henkilo
```

- monissa tuotteissa on kuitenkin SUBSTR(mjono,mista,mihin)
-funktio (Accessissa Mid-funktio), esim.:

```
SELECT sukunimi, ' ', SUBSTR(etunimi,1,1), ' '  
FROM henkilo
```

- Tuloksena lista: Aaltonen, M. ...

- Muita merkkijonofunktioita:

- | | |
|------------------|---------------------------------------|
| – CHAR_LENGTH | -- merkkien määrä (Oraclessa LENGTH) |
| – UPPER(nimi) | -- merkit isoiksi (Accessissa UCase) |
| – LOWER(nimi) | -- merkit pieniksi (Accessissa LCase) |
| – LEFT(nimi, n) | -- merkkijono vasemmalta (n merkkiä) |
| – RIGHT(nimi, n) | -- merkkijono oikealta (n merkkiä) |

VIELÄ MERKKIJONOJEN KÄSITTELYSTÄ

- Merkkijonojen yhdistäminen (konkatenointi)
 - standardia noudattavat Oracle ja DB2, jossa merkkijonot yhdistetään kahdella pystyviivalla, esim.:
`SELECT sukunimi || ', ' || etunimi AS Nimi`
`FROM henkilo`
 - MS:n tuotteissa käytetään + -merkkiä ja MySQL:ssä CONCAT-funktiota
- Tietotyypin muuntaminen:
 - eksplisiittisesti numeerisen ja merkkimuotoisen välillä, esim. ANSI-standardin CAST-funktiolla tai Oraclessa TO_CHAR (n, [format [,lang]]); muutoin implisiittisesti

MUITA FUNKTIOITA

- Numerofunktioita ovat mm. desimaalien pyöristysfunktiot ROUND ja FLOOR; ROUND-funktiolla saadaan luku pyöristettyä kokonaisluvuksi:
 - `SELECT ROUND(palkka*1.1,0) AS palkankorotus`
`FROM henkilo`
- NULL-arvon muuttamiseen voidaan käyttää esim. COALESCE-funktiota, jolla NULL muutetaan esim. nolaksi: `COALESCE(sarake, 0)`
- Päivämääräfunktiot, ks. Hovi s. 56 – 65
 - Tuotekohtaiset maa-asetukset huomioitava
 - ANSI-standardissa CURRENT_DATE: tämä päivä



SQL:2003-STANDARDIN UUSIA FUNKTIOITA

- Nk. skalaarifunktiot: LN (), EXP (), POWER (), SQRT (), FLOOR (), CEIL[ING] (), WIDTH_BUCKET()
- Koostefunktiot: STDDEV_POP (), STDDEV_SAMP (), VAR_POP (), VAR_SAMP () ym.
- Nk. taulukkokofunktiot: RANK () OVER ..., DENSE_RANK () OVER ..., PERCENT_RANK () OVER ..., CUME_DIST () OVER ..., ROW_NUMBER () OVER ...
- Muitakin on ...
- Tuotekohtaisista eroista on hyvä yhteenveto:
<http://troels.arvin.dk/db/rdbms/#functions>

CASE-LAUSE

- Tarvitaan kun sarakkeen otsikko riippuu sarakkeen arvosta
- Kaksi syntaksia (sarakenimen kanssa tai ilman)
- Esimerkiksi haetaan tuotteita ja luokituksia:

```
SELECT tuote_nimi,  
CASE  
    WHEN hinta < 100 THEN 'edullinen'  
    ELSE 'laatutavara'  
END AS luokitus  
FROM tuote
```

LIITOKSET



LIITOKSET - HAUT USEAAN TAULUUN

- Tavallisin liitos on valintaliitos
 - ehtona =, !=, <, <=, >=, >
 - yhtäläisyysliitos (=) yleisin (vrt. Inner join)
 - taulut liitetään toisiinsa yleensä avaimilla (perus- ja viiteavain)
- Esim.
`SELECT AsiakasNimi, tilausnumero`
`FROM asiakas INNER JOIN tilaus`
`ON asiakas.asiakastunnus = tilaus.asiakastunnus;`
- Kvalifiointi: taulun nimi sarakenimen eteen
 - pakollista, jos saman niminen sarake kahdessa taulussa

PERINTEINEN LIITOS

- Liitosehto WHERE-lauseessa
- Esim. hae yrityksen nimi, ko. yrityksen henkilöiden nimet ja palkat yli 2500 tienaavista:
 - **SELECT** yritys.yrtun, Nimi, -- Hae yrityksen tunnus, nimi
 - Sukunimi, Etunimi, Palkka -- ja henkilön nimi + palkka
 - **FROM** yritys, henkilo -- näistä tauluista
 - **WHERE** yritys.yrtun = henkilo.yrtun -- Liitosehto!
 - **AND** Palkka > 2500 -- Hakuehto palkalle
 - **ORDER BY** Nimi -- Lajittelu



UUSI LIITOSSYNTAKSI (SUOSITELTAVAMPI VAIHTOEHTO)

- Liitosehto FROM-lauseessa
- Hae yrityksen nimi, ko. yrityksen henkilöiden nimet ja palkat yli 2500 tienaavista:
 - **SELECT** yritys.Yrtun, Nimi, -- Hae yrityksen tunnus, nimi
 - Sukunimi, Etunimi, Palkka -- henkilön suku- ja etunimi + palkka
 - **FROM** yritys -- YRITYS –taulusta, joka
 - **INNER JOIN** henkilo -- liitetään HENKILO-tauluun
 - **ON** yritys.Yrtun = henkilo.Yrtun -- Liitosehto
 - **WHERE** Palkka > 2500 -- Hakuehto palkalle
 - **ORDER BY** Nimi -- Lajittelu
- Standardin mukaan voidaan myös käyttää NATURAL JOIN -liitosta, jos liitossarakkeet ovat samannimiset



MONEN TAULUN LIITOS

- Kun liität kolme tai useampia tauluja toisiinsa, on syytä olla huolellinen JOIN-määrittelyn kanssa
- Eräs ratkaisu on käyttää sellaista työkalua taulujen liittämiseksi, jolla SQL-käsky saadaan edelleen muokattavaksi, esim. MS Access:
 - SELECT OPINTOJAKSO.OpintojaksonNimi,
OPISK_OJAKSO.Kehitysehdotukset,
OPISKELIJA.OpiskNimi
 - FROM OPISKELIJA INNER JOIN (OPINTOJAKSO INNER
JOIN OPISK_OJAKSO ON OPINTOJAKSO.OJtunnus =
OPISK_OJAKSO.OJtunnus) ON OPISKELIJA.OpiskNro =
OPISK_OJAKSO.OpiskNro



LIITOKSET: ULKOLIITOS (OUTER JOIN)

- Outer Join ottaa mukaan myös ne rivit, jotka eivät täytä join-ehtoa; esim.:
 - haetaan myös ne asiakkaat, joille ei löydy tilauksia:
`SELECT AsiakasNimi, tilausnumero
FROM asiakas
LEFT OUTER JOIN tilaus
ON asiakas.asiakastunnus = tilaus.asiakastunnus`
 - vanha syntaksi:
`SELECT AsiakasNimi, tilausnumero
FROM asiakas, tilaus
WHERE asiakas.asiakastunnus (+) = tilaus.asiakastunnus`
- OUTER-sana voidaan jättää pois SQL-standardissa

LIITOS ITSEENSÄ

- Liitos itseensä: tarvitaan korrelaationimeä (alias)
- Esim. haetaan JAMKin eri yksiköt (org.hierarkiasta):

```
SELECT a.nimi  
FROM asiakas a, asiakas b  
WHERE a.emoyritystunnus = b.asiakastunnus  
AND b.asiakastunnus = 'JAMK'
```
- Joissakin tuotteissa pitää käyttää AS-sidesanaa korrelaationimen edessä, joissakin pitää olla välilyönti
- Kun korrelaationimi on otettu käyttöön, ei alkuperäistä nimeä saa käyttää sarakkeen kvalifioinnissa (esim. edellisessä käskyssä SELECT asiakas.nimi antaisi virheilmoituksen)

YHDISTE (UNION)

- Kaksi taulua (tai useampia) yhdistetään samantyyppisen tiedon avulla
- Esim. opettajien ja opiskelijoiden puhelinnumerot:

```
SELECT OpeNimi AS Nimi, PuhNro, 'opettaja'
```

```
FROM opettaja
```

```
UNION
```

```
SELECT OpiskNimi AS Nimi, PuhNro, 'opiskelija'
```

```
FROM opiskelija
```

```
ORDER BY 1
```

ALIKYSELYT



ALIKYSELY

- Voidaan käyttää SELECT, INSERT, UPDATE ja DELETE -komennoissa
- **Sisin kysely suoritetaan ensiksi**
- Voi palauttaa vain yhden sarakkeen ja sille yhden tai useamman arvon
 - vertailuoperaattori "=": alikysely voi palauttaa vain yhden arvon
 - IN-operaattori tai vertailuoperaattori ANY tai ALL -ehdon kanssa sallii alikyselyn palauttavan useita arvoja
- Oltava aina suluissa ja haettavana saa olla vain yksi sarake
- Käyttää automaattisesti distinct-määrettä

ALIKYSELYT – ESIMERKKI1: =

- Hae sen henkilön nimi, joka myy solmioita:

```
SELECT HenkNimi  
FROM henkilo  
WHERE HenkID =  
    (SELECT MyyjaID  
     FROM tuote  
     WHERE TuoteNimi = 'Solmio');
```
- Aika usein alikyselyssä etsitään arvoa koostefunktiolla (palauttaa aina yhden arvon)

ALIKYSELYT – ESIM. 2 (KOOSTEFUNKTION KÄYTTÖ)

- Tyypillistä on hakea alikyselyssä jokin koostefunktion palauttama arvo ja verrata sitä toiseen joukkoon

```
SELECT Tuotenimi, hinta, myyja
```

```
FROM tuote
```

```
WHERE hinta >
```

```
    (SELECT AVG(hinta)
```

```
    FROM tuote
```

```
    WHERE myyja = 'Pekka');
```

-- hakuehto:

-- ne myyjät

-- jotka myyvät **keskimäärin**

-- kalliimpia tuotteita

-- kuin Pekka

ALIKYSELYT – ESIMERKKI 3: IN

- Hae niiden tuotteiden nimet ja hinnat, joita myy laiteosastolla olevat henkilöt:

```
SELECT TuoteNimi, hinta
```

```
FROM tuote
```

```
WHERE myyjäID IN
```

```
(SELECT myyjäID
```

```
FROM myyja
```

```
WHERE osasto = 'laite');
```

- Myös NOT IN on käyttökelpoinen, kun halutaan sulkea pois alikyselyn palauttama tulosjoukko

ALIKYSELYT – ESIMERKKI 4: ANY JA ALL

- Jos käytetään vertailuoperaattoria $>$ tai $<$, niin IN ei käy
`SELECT nimi, hinta, myyja`
`FROM tuote` -- hakuehto:
`WHERE hinta > ANY` -- ne myyjät
`(SELECT hinta` -- jotka myyvät
`FROM tuote` -- kalliimpia tuotteita
`WHERE myyja = 'Pekka');` -- kuin Pekka
- ANY eli SOME: jos alikyselyn mikä tahansa arvo täyttää ehdon
- ALL: jos alikyselyn jokainen yksittäinen arvo täyttää ehdon

ALIKYSELYT – ESIMERKKI 5: EXISTS

- EXISTS
 - tosi, jos alikysely palauttaa ainakin yhden rivin
 - epätosi, jos alikysely ei palauta yhtään riviä
- Esimerkki: haetaan ne tuotteet, jotka ovat myyjän ainoita myymiä tuotteita:

```
SELECT tuote_nimi, hinta
FROM tuote t
WHERE NOT EXISTS
  (SELECT *
   FROM tuote
   WHERE myyja = t.myyja
   AND tuote_nimi <> t.tuote_nimi);
```

- Yllä on esimerkki korreloidusta (kytketystä) alikyselystä, jossa alikyselyyn liitetään arvo ylemmältä tasolta



3. TAULUJEN MÄÄRITTELY JA MUUTTAMINEN

DDL: TAULUJEN LUONTI, MUUTOS JA POISTO
DML: TAULUJEN TIETOJEN YLLÄPITO
TAPAHTUMIEN (TRANSAKTIOIDEN) HALLINTA
NÄKYMÄT, SYNONYymIT JA MUUT TIETOKANTAOBJEKTIT



TAULUJEN PERUSTAMINEN

- Taulun perustajalla täytyy olla valtuudet (GRANT) tietokantaan
- Kullekin taulun sarakkeelle annetaan nimi ja tietotyyppi (+ maksimipituus ja mahdolliset desimaalit)
- Tarvittaessa määritetään rajoitteita (NOT NULL, PRIMARY KEY, FOREIGN KEY, UNIQUE, DEFAULT ja CHECK)

SQL-ESIMERKKI: TAULUN LUONTI

- Taulu luodaan CREATE TABLE –käskyllä:

```
CREATE TABLE YRITYS (  
    YritysID                smallint NOT NULL,  
    YritysNimi              varchar(80),  
    Toimialakoodi           char(10),  
    YrLisaysPvm date DEFAULT Current_date,  
    CONSTRAINT pk_YritysID PRIMARY KEY (YritysID))
```

- Viite-eheys REFERENCES-määreellä, ks. Rajoitteet

```
CONSTRAINT fk_Yritys_Toimialakoodi FOREIGN KEY (Toimialakoodi)  
REFERENCES TOIMIALA (Toimialakoodi)
```

- Hyvä käytäntö: rajoittimen nimi kertoo, mikä rajoitin on kyseessä ja mihin tauluun ja sarakkeeseen rajoitin kohdistuu

MUUTAMIA SQL-92-STANDARDIN TIETOTYYPPEJÄ

- CHAR [(pit.), kiinteä, oletuspit. 1]
- VARCHAR [(pit.), vaihtuvanmitt. merkkijono]
 - VARCHAR on sama kuin CHAR VARYING
- NUMERIC [(koko pituus [,desimaaliosa])]
 - NUMERIC on sama kuin DECIMAL
- INTEGER [kokonaisluku, pit. tuotekoht.]
 - Voidaan lyhentää INT
- DATE [vuosi (0001-9999), kk ja päivä]
- TIME [(pit.), tunti, minuutti ja sekunti]
- BLOB [SQL-99, binary large object]



OCELOT SIS. SQL2:N JA OSAN SQL3:N TIETOTYYPEISTÄ

Create Table

Table name and column descriptors. "OK" to move SQL statement to main screen.

Schema Name

Table Name

Checks

Column Name

Data Type

- ☐ BIT
- ☐ BIT VARYING
- ☐ BLOB
- ☐ BOOLEAN
- ☒ CHAR
- ☐ CHAR VARYING
- ☐ CLOB
- ☐ DATE
- ☐ DECIMAL
- ☐ DOUBLE PRECISION
- ☐ FLOAT
- ☐ INTEGER
- ☐ INTERVAL
- ☐ NUMERIC
- ☐ REAL
- ☐ REF
- ☐ SMALLINT
- ☐ TIME
- ☐ TIME WITH TIME ZONE
- ☐ TIMESTAMP
- ☐ TIMESTAMP WITH TIME ZONE

Description

Length

Character Set

Collation

☐ Primary?

☐ Not Null?

Default

CREATE TABLE koe(koe CHAR)

SQL:2003 – MUUTOKSET AIKAISEMPAAN STANDARDIIN

- Uudet tietotyypit:
 - BIGINT (tarkkuus tuotekohtainen, kuitenkin aina $\text{BIGINT} \geq \text{INTEGER} \geq \text{SMALLINT}$)
 - MULTiset (voidaan luoda ARRAY-tyyppinen kokoelma)
- Kaksi tietotyyppiä poistettu:
 - BIT
 - BIT VARYING
- Sequence
- CREATE TABLE LIKE ...
- Ym.

ORACLEN TIETOTYYPPEJÄ

- CHAR [255 tavua, kiinteäpituuksinen]
- VARCHAR2 [2 kt (4 kt Oracle 8:ssa)]
- NUMBER [numeerinen $1 \times 10^{-130} - 9,99 \times 10^{125}$]
- LONG [2 Gt]
- RAW [255 tavua, binaarinen]
- LONG RAW [2 Gt, binaarinen]
- DATE [pvm ja aika, kiinteäpituuksinen]
- ROWID [rivin ID:n muuttujatyyppi]
- NCHAR ja NVARCHAR2: kansallisen kielen tietotyyppi
- CLOB: yksitavuinen, NCLOB: yksi- tai monitavuinen, 4 Gt
- BFILE: osoitin tallennettuun ulkoiseen bin.tiedostoon
- Oraclen oma (esim. SDO_GEOMETRY for Oracle Spatial)



SQL SERVERIN (TRANSACT-SQL) TIETOTYYPPEJÄ

- `char [ ( n ) ] ; nchar [ ( n ) ]`
 - vakiomittainen merkkitieto, $n = 1 \dots 255$
- `varchar [ ( n ) ]`
`nvarchar [ ( n | max ) ]`
 - $n = 1 \dots 255 \dots 4000 \dots 8000$ (riippuen versiosta), $\max = 2^{31} - 1$
- `text, image`
 - suuret tietomäärät (maks 2 Gtavua)
- `binary [ ( n ) ] , varbinary [ ( n ) ]`
- `bit`
 - Saa arvot 1, 0 tai NULL
- `datetime, smalldatetime`
 - pvm-typit
- `timestamp`
 - muuttuu riviä päivitettäessä
- Kokonaisluvut:
 - `bigint`
 - `int (32 bit)`
 - `smallint (16 bittiä)`
 - `tinyint (0 ... 255)`
- Desimaaliluvut:
 - `decimal [ ( p, [ , s ] ) ]` **tai** `numeric [ ( p, [ , s ] ) ]`
 - p = tarkkuus ($1 \dots 38$), s = skaala, desimaaliosien määrä
 - `float [ ( n ) ]`
 - liukuluku, tarkkuus 15 numeroa
 - `real`
 - liukuluku, tarkkuus 7 numeroa
- Raha:
 - `money, smallmoney`

Tuotekohtaisista eroista: http://troels.arvin.dk/db/rdbms/#data_types

SQL-ESIMERKKI: TAULUN MUUTOS

- Taulu muutetaan ALTER TABLE –käskyllä, esim. sarakkeen lisäys (loppuun)

ALTER TABLE yritys

ADD COLUMN maa char (20)

- ALTER TABLE tuli osaksi SQL-standardia vasta SQL3:ssa
- Katso tuotekohtaisesti, voidaanko lisätä kerralla useita sarakkeita tai sarake johonkin väliin
- Sarakkeen muutos: ALTER COLUMN ja poisto: DROP COLUMN, ks. oma tuotteesi, esim.

ALTER TABLE yritys

DROP COLUMN maa CASCADE



EHEYSVAATIMUKSET

- Tietokantaan kohdistuu kolmen tyyppisiä eheysvaatimuksia:
 - Avaineheys: taululla tulee olla perus- eli pääavain (PK)
 - Viite-eheys: viiteavaimelle tulee löytyä arvo viitattavan taulun pääavaimesta
 - Attribuuttieheys: attribuuttien tulee kuulua sallittuun joukkoon
- Määritetään nk. rajoitteella (ks. seur. kalvo)
- Palvelin tarkistaa eheysvaatimusten toteutumisen

RAJOITTEET ELI PAKOTTEET (CONSTRAINTS)

- Määritellään CREATE TABLE tai ALTER TABLE -komennoissa
- Rajoitteita voidaan määritellä sarakkeille tai tauluille
- Rajoitteita ovat:
 - avaimet: PRIMARY KEY, FOREIGN KEY ja UNIQUE
 - oletusarvot ja raja-arvotarkistukset: DEFAULT ja CHECK
- esimerkkejä:

ALTER TABLE dept

ADD CONSTRAINT pk_deptno PRIMARY KEY (deptno),

ADD priority SMALLINT DEFAULT 1,

ADD CONSTRAINT chk_sukup CHECK (sukupuoli IN ('M','N'))



PRIMARY KEY, UNIQUE JA FOREIGN KEY

- Primary Key (**CONSTRAINT** pk_sarake **PRIMARY KEY** (sarakenimi1 [,sarakenimi2]))
 - Yksiköi taulun rivit
 - Sarakkeiden tulee olla Not Null -tyyppisiä
 - Käytetään hyväksi viite-eheyttä luotaessa
- Unique (**CONSTRAINT** sarakenimi **UNIQUE**)
 - Luo yksilöivän indeksin (oletus on ei klusteroitu)
 - Sarakkeet voivat sisältää Null-arvoja
 - Voi olla useita / taulu
- Foreign Key (**CONSTRAINT** fk_sarake **FOREIGN KEY** (sarakenimi) **REFERENCES** vanhempi (perusvain))
 - Luo viite-eheyden taulujen välille
 - Voi sisältää useita sarakkeita



VIITEAVAIMEN LISÄYS JA POISTOJEN/PÄIVITYSTEN VYÖRYTYS

- Viite-eheys voidaan määritellä taulua luotaessa tai jälkeenpäin ALTER TABLE -käskyllä
- Esim. Puhelin-taulu, jonka vanhempi on Henkilo; jos henkilö poistetaan niin myös henkilön puhelimet poistetaan (ON DELETE CASCADE)

```
ALTER TABLE Puhelin  
  ADD CONSTRAINT fk_HenkiloID  
  FOREIGN KEY (HenkiloID)  
  REFERENCES Henkilo (HenkiloID)  
  ON DELETE CASCADE
```

- Viisi vaihtoehtoa: CASCADE, SET DEFAULT, SET NULL, NO ACTION, RESTRICT, joista kaksi viimeistä estävät poiston, jos vanhemmalla on lapsia
- Myös päivitykset voidaan vyöryttää: ON UPDATE CASCADE



LASKURI: AUTOMAATTISESTI KASVAVA SARAKE

- Taululle voidaan määritellä sarake, joka on automaattisesti kasvava numero
- Määrittelyssä annetaan alkuarvo ja lisäys
- Standardin mukaan laskuri luodaan joko IDENTITY-määreellä tai SEQUENCE-käskyllä (MySQL:ssä AUTO_INCREMENT-määreellä)
- Ominaisuus voidaan luoda myös CREATE TABLE, ALTER TABLE tai SELECT INTO –komennon yhteydessä
- Laskuri toimii usein perusavaimena



LASKURIN LUONTI, 2 VAIHTOEHTOA

- Esimerkki 1:

```
CREATE TABLE tuote (  
  tuoteID INTEGER  
  GENERATED ALWAYS AS  
    IDENTITY  
  -- ja lisäksi esim.  
  START WITH 100  
  INCREMENT 1  
  MINVALUE 100  
  NO MAXVALUE  
  NO CYCLE,  
  ... )
```

- Esimerkki 2:

```
CREATE SEQUENCE tuoteID AS  
  INTEGER  
  START WITH 1  
  INCREMENT BY 1  
  MAXVALUE 100000  
  MINVALUE 1
```


SQL:2003 - GENERATED ALWAYS

- Esimerkki:

```
CREATE TABLE employees (  
  EMP_ID INTEGER,  
  SALARY DECIMAL(7,2),  
  BONUS DECIMAL(7,2),  
  TOTAL_COMP GENERATED ALWAYS AS (  
    SALARY + BONUS ),  
  HR_CLERK GENERATED ALWAYS AS (  
    CURRENT_USER )  
)
```

TAULUN POISTO

- Taulun poisto tapahtuu DROP TABLE -käskyllä, esim.

DROP TABLE temp

- Tällöin vyörytetään poistot, jos CASCADE DELETE on lapsitauluun määritetty, muutoin poisto estetään (jos viite-eheys on asetettu)
- Samalla poistuu taulussa olevat tiedot (olihan siis backup tallessa tai ROLLBACK-käsky hanskassa?)

DML (DATA MANIPULATION LANGUAGE)

TIETOJEN LISÄYS, MUUTOS JA POISTO
TAPAHTUMIEN ELI TRANSAKTIOIDEN HALLINTA



TIETOJEN LISÄÄMINEN

- INSERT INTO taulu VALUES (arvot)

- Esim.

```
INSERT INTO tuote(tuoteNimi, hinta)  
VALUES ('SQL-opas', 20)
```

- Sarakkeiden nimiä ei tarvitse luetella, jos kaikille annetaan arvo tai niillä on oletusarvo tai ne sallivat Null-arvon; on kuitenkin suositeltavaa luetella ne!

- Taulusta toiseen:

```
INSERT INTO taulu1 SELECT s1, s2 FROM taulu2
```

- Useimmissa tuotteissa (kuten MySQL:ssä, SQL Server 2008:ssa ja IBM DB2:ssa) voidaan antaa useita lisättäviä rivejä (Multirow inserts) yhdessä käskyssä, ks. esim. wikistä: [http://en.wikipedia.org/wiki/Insert \(SQL\)](http://en.wikipedia.org/wiki/Insert_(SQL))

TIETOJEN MUUTTAMINEN ELI PÄIVITTÄMINEN

UPDATE taulu SET arvo = uusiarvo [WHERE ehto]

- Olemassa olevien tietojen päivittäminen

UPDATE tuote SET hinta = 120

WHERE tuoteID = 123 -- tuotteen 123 hinnaksi 120

Tai esim. 10 %:n hinnan lisäys:

UPDATE tuotteet SET hinta = hinta * 1.1

- Päivitys, jossa 2 taulua on liitetty toisiinsa

UPDATE tuote SET varastosaldo = 10

FROM tuote INNER JOIN myyja

ON tuote.myyjaID = myyja.myyjaID

TIETOJEN POISTAMINEN

- Rivien poistaminen taulusta
 - DELETE {FROM} [taulun nimi] WHERE [ehto poistettaville riveille]:

DELETE FROM tuote WHERE tuoteNimi = 'auto'

- Poisto alikyselyn avulla

DELETE FROM asiakas WHERE asiakasID
NOT IN (SELECT asiakasID FROM TILAUS)

TAPAHTUMIEN ELI TRANSAKTIOIDEN HALLINTA

- Tapahtuma on yleensä useamman käskyn kokonaisuus, joka suoritetaan joko kokonaan tai ei ollenkaan
- Esim. maalitilaston päivitys (joukkueID:t ja maalien määrä välitetään parametreina):

BEGIN TRANSACTION

```
INSERT INTO ottelu (joukkue1ID, joukkue2ID, kotimaalit,  
    vierasmaalit) VALUES (:j1id, :j2id, :kotimaalit, :vierasmaalit);
```

```
UPDATE joukkue  
    SET maalisaldo = maalisaldo + :kotimaalit - :vierasmaalit  
    WHERE joukkueID = :j1id;
```

```
UPDATE joukkue  
    SET maalisaldo = maalisaldo + :vierasmaalit - :kotimaalit  
    WHERE joukkueID = :j2id;
```

COMMIT;

- **COMMIT** hyväksyy, **ROLLBACK** peruuttaa tapahtuman

SQL-92 ERISTYVYYSTASOT

- Eristyvyystaso ilmaisee transaktion eristyvyyttä eli riippumattomuutta muista samanaikaisista transaktioista
- Tavoitteena ehkäistä samanaikaisuuden ongelmia:
 - 1 Hukatun päivityksen ongelma (lost update): päällekirjoitus
=> estettävä aina!
 - 2 Keskenäisen käsittelyn ongelma (dirty read)
 - a) luetaan peruutettava
 - b) päivitetään peruutettava
 - 3 Ristiriitaisen tulkinnan ongelma
 - a) eritahtinen päivitys (nonrepeatable read)
 - b) uusien rivien ongelma (phantom rows)
- Katso esimerkit erillisestä kalvosetistä (tekijä J. Rantanen):
<http://homes.jamk.fi/~huojo/opetus/IIZO3030/eristyvyys.pps>

alkuper. versio: <http://data.hamk.fi/materiaalit/tiedonhallinta2/ccr/eristyvyys.pps>



ERISTYVYYSTASOT

Komennolla

SET TRANSACTION ISOLATION LEVEL

voidaan asettaa neljä eri eristyvyystasoa:

	Dirty Read	Nonrepeatable Read	Phantoms
--	------------	--------------------	----------

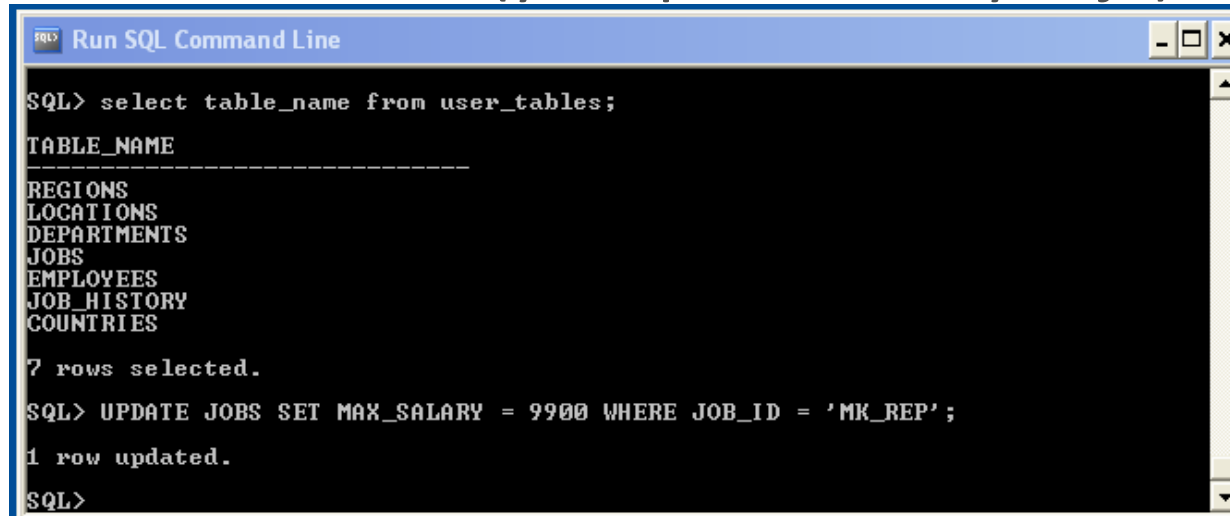
- | | | | |
|---------------------------|----------------|----------------|----------------|
| • READ UNCOMMITTED | mahdollinen | mahdollinen | mahdollinen |
| • READ COMMITTED | Ei mahdollinen | mahdollinen | mahdollinen |
| • REPEATABLE READ | Ei mahdollinen | Ei mahdollinen | mahdollinen |
| • SERIALIZABLE | Ei mahdollinen | Ei mahdollinen | Ei mahdollinen |

Esim. **SET TRANSACTION ISOLATION LEVEL SERIALIZABLE**



ESIMERKKI, ORACLE

- Kaksi käyttäjää, toinen päivittää taulua, toinen lukee sitä; alkutilanne (ylempi on hr-käyttäjä):



```
SQL> select table_name from user_tables;

TABLE_NAME
-----
REGIONS
LOCATIONS
DEPARTMENTS
JOBS
EMPLOYEES
JOB_HISTORY
COUNTRIES

7 rows selected.

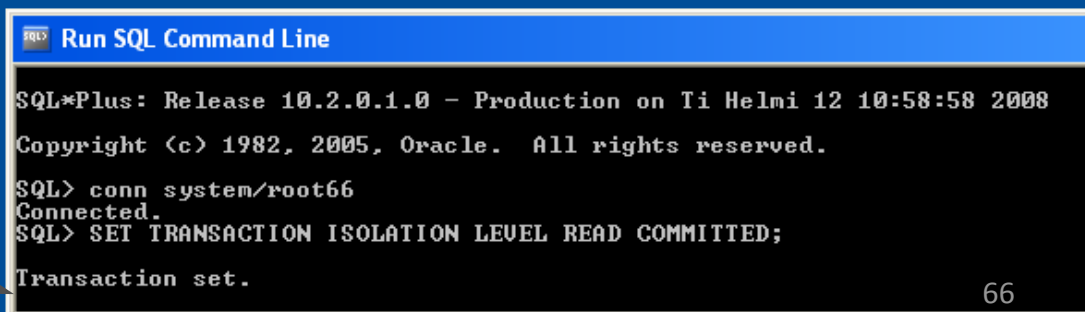
SQL> UPDATE JOBS SET MAX_SALARY = 9900 WHERE JOB_ID = 'MK_REP';

1 row updated.

SQL>
```

Oracle
Database 10...

Kun system-käyttäjä antaa
käskyn `SELECT * FROM
hr.jobs`, mitä tapahtuu?



```
SQL*Plus: Release 10.2.0.1.0 - Production on Ti Helmi 12 10:58:58 2008
Copyright (c) 1982, 2005, Oracle. All rights reserved.

SQL> conn system/root66
Connected.
SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;

Transaction set.
```

TAPAHTUMIEN HALLINTAA MUISSA TUOTTEISSA

- MySQL:
<http://www.databasejournal.com/features/mysql/article.php/3393161>
- OCELOT: asenna se uudelleen, ohje
<http://www.ocelot.ca/magazine.htm#CONFIGURE>
- SQL Server: ota yhteys virtuaalikoneeseen ja tee kyselyjä/päivityksiä kahdessa eri ikkunassa (ajaa käskyt rivi kerrallaan: valitse käsky ja paina F5)

MUITA TIETOKANTAOBJEKTEJA

NÄKYMÄT, SYNONYIMIT, INDEKSOINTI,
VALTUUDET JA SYSTEEMIHAKEMISTO



NÄKYMÄT

- **Näkymä** (view) on looginen näyte tietokannan tauluista tai näkymistä
- Näkymä ei voi sisältää SELECT INTO, ORDER BY tai COMPUTE –ehtoja (useimmissa tuotteissa)
- Ei voi viitata väliaikaisiin tauluihin
- Tarvitaan SELECT-oikeus tauluihin
- Esim:

`CREATE VIEW Hinnasto`

`AS SELECT nimi, hinta FROM tuote`

- Käyttö:

`SELECT * FROM Hinnasto`



NÄKYMÄN KÄYTTÖ

- Näkymän kautta voidaan päivittää tai lisätä tietoja, mutta
 - yleensä vain yhteen tauluun
 - näkymässä ei saa olla DISTINCT-määrettä, HAVING-lausetta eikä alikyselyä
- Näkymä poistetaan **DROP VIEW** -komennolla
- Jos näkymässä on laskettuja sarakkeita, view-määrittelyn nimen jäljessä täytyy luetella sarakkeiden nimet:

```
CREATE VIEW ALVhinnasto(nimi, ALVhint)  
AS SELECT nimi, hinta*1.22  
FROM TUOTE
```



ESIMERKKI LASKENTAOSASTO-NÄKYMÄSTÄ

```
CREATE VIEW Accounting AS  
  SELECT dname, loc, empno, ename, job  
  FROM dept INNER JOIN emp  
  ON dept.deptno = emp.deptno  
  WHERE dept.deptno = 10;  -- 10: ACCOUNTING-osasto
```

```
SELECT * FROM Accounting;
```

```
DROP VIEW Accounting;
```

VINKKEJÄ NÄKYMIEEN LUONTIIN

- Testaa ensin kysely ja luo vasta sitten näkymä
- Jos taulun nimi muuttuu, luo kysely jossa on alkuperäisen taulun sarakkeet ja anna sen nimeksi alkuper. taulun nimi; tällä tavoin sinun ei tarvitse muuttaa olemassa olevia SELECT- ym. käskyjä
- Näkymillä voit suojata tietoja käyttäjäryhmittäin
- Voit rakentaa näkymän toisen näkymän päälle, mutta se voi vaikeuttaa ymmärtämistä
- Joissakin tuotteissa nk. Materialized view (eli indexed view), jossa näkymä tallennettu levyille (ikään kuin tauluksi)



SYNONYymIT (EI KUULU STANDARDIIN)

- Helpottavat viittaamista tauluun antamalla tauluviittaukselle jokin kuvaava nimi
- Esim. jos toinen käyttäjä (Pekka) on antanut SELECT-oikeuden omaan tauluunsa (kurssi), voidaan tiedot hakea komennolla **SELECT * FROM Pekka.kurssi**
- Luomalla synonyymi (CREATE SYNONYM kurssit FOR Pekka.kurssi) voidaan em. kysely tehdä käskyllä **SELECT * FROM kurssit**
- Tarkista tuki synonyymille omasta tuotteestasi



VALTUUDET

- Tavoitteena tietojen suojaus
- Tiedot voidaan suojata käyttäjä- ja taulukohtaisesti
- Käyttäjiä luodaan useimmissa tuotteissa CREATE USER -käskyllä ja salasana määritetään samalla, esim. CREATE USER jouni IDENTIFIED BY jh1
- Joissain tuotteissa salasana luodaan CREATE PASSWORD -käskyllä
- Käyttöoikeudet voidaan niputtaa nk. rooliin (voi olla myös yksittäinen henkilö) ja luoda CREATE ROLE -käskyllä, esim. CREATE ROLE opettaja
- Valtuudet myönnetään GRANT-käskyllä, esim. GRANT SELECT ON kirjat TO jouni tai kaikki oikeudet GRANT ALL PRIVILEGES -käskyllä
- Valtuudet poistetaan REVOKE-käskyllä ja käyttäjä DROP USER tai DROP ROLE -käskyllä



INDEKSIT

- Tauluille voidaan perustaa sarakekohtaisia hakemistoja eli indeksejä, esim.
 - `CREATE INDEX i_postinro ON henkilo(postinro)`
- Indeksien käytön tarkoituksena on mm. nopeuttaa tietokantaan tapahtuvia hakuja
- Yksilöivä (unique) indeksi voi toimia myös perusavaimena
 - `CREATE UNIQUE INDEX i_sarake ON taulu(sarake)`

INDEKSIEN KÄYTTÖ

- Indeksoinnin etuja:
 - hakujen, liitosten, ryhmittelyjen ja lajittelujen nopeutuminen
 - pakottaa yksilöimään rivit
- Kannattaa indeksoida:
 - perusavaimet ja viiteavaimet (jokainen viiteavain erikseen!)
 - sarakkeet, joita haetaan tai järjestetään usein (ja joissa on paljon tietoa eli rivejä on tuhansia tai miljoonia)
- Indeksointi vie levytilaa ja aikaa luotaessa ja muutoksissa
- Ei kannata indeksoida:
 - harvoin haussa käytettäviä sarakkeita
 - sarakkeita, joiden selektiivisyys on huono



PEUKALOSÄÄNTÖ INDEKSIN LUOMISELLE (NK. 3 TÄHDEN INDEKSI)

1. Ota kaikki WHERE-kohdassa mainitut sarakkeet
2. Lisää ORDER BY –kohdassa mainitut sarakkeet
3. Laita loppuun SELECT-lauseessa olevat sarakenimet

Esim. sopiva indeksi ”**SELECT a, b, c FROM t WHERE d = 'x' ORDER BY b**” –käskylle olisi

CREATE INDEX i_dbac ON t(d,b,a,c)

- Jos kaikki SELECTissä mainitut sarakkeet on mukana indeksissä, kyse on paksusta indeksistä
- Indeksi poistetaan DROP INDEX -käskyllä



TIETOKANNAN JA TIETOKANTAKAAVAN ("SKEEMAN") PERUSTAMINEN + SYSTEEMIHAKEMISTO

- Tietokanta luodaan monissa tuotteissa CREATE DATABASE -käskyllä, esim. CREATE DATABASE kirjat
- Tietokantakaava (schema) on kokoelma tietokanta-objekteja (taulut, näkymät jne.), se vastaa siis käsitettä tietokanta; kaava luodaan käskyllä **CREATE SCHEMA**
- Tieto eri tietokantaobjekteista on nk. tieto- eli systeemihakemistossa (catalog, data dictionary, schemata); sen nimi standardin mukaan: **INFORMATION_SCHEMA**
- Tietokantakaavan omistaa aina yksi henkilö (rooli)
- Tietokanta otetaan käyttöön eri tuotteissa eri tavoilla:
 - USE-käskyllä (MS SQL-Server, MySQL)
 - Standardissa on käsky **SET SCHEMA**



HAUT SYSTEEMIHAKEMISTOSTA

- Käskyt vaihtelevat tuotteittain
- Lista (käyttäjän) tauluista saadaan eri tuotteissa seuraavilla käskyillä:
 - `SHOW TABLES` (MySQL)
 - `SELECT table_name from user_tables` (Oracle)
 - `SELECT tname FROM tab` (Oracle)
 - `SELECT table_name FROM information_schema.tables` (SQL Server)
- Lista tietohakemiston sisältämistä tietokannoista ja niiden omistajista (Ocelot):
 - `SELECT catalog_name, schema_name, schema_owner FROM information_schema.schemata`



TAULUN SARAKKEIDEN KUVAUS

- Useissa tuotteissa (Oracle, DB2, [MySQL](#)) on käsky DESCRIBE, joka kertoo tietoa sarakkeista ja niiden tietotyypeistä, esim.

```
mysql> DESCRIBE city;
```

Field	Type	Null	Key	Default	Extra
Id	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			

- Standardin mukaan:

```
SELECT column_name, data_type, column_default, is_nullable
FROM information_schema.tables AS t
JOIN information_schema.columns AS c ON
  t.table_catalog = c.table_catalog AND
  t.table_schema = c.table_schema AND
  t.table_name = c.table_name
WHERE
  t.table_name = 'TABLE-NAME'
```



LISÄTIETOA

- Relaatiotietokantasanasto: <http://www.cs.helsinki.fi/~laine/relaationsanasto/>
- Historiaa ym.: <http://en.wikipedia.org/wiki/SQL>
- Suomeksi: <http://fi.wikipedia.org/wiki/SQL>
- Yhteenvedo: <http://www.ocelot.ca/glossary.htm>
- SQL-kurssi: <http://sqlcourse.com> ja <http://sqlcourse2.com>
- SQL-standardeista: <http://www.jcc.com/sql.htm>
- SQL'92, '99 ja 2003: <http://savage.net.au/SQL/>
- Tuotteiden erosta: <http://troels.arvin.dk/db/rdbms/>

YHTEENVETO PERUS-SQL-KÄSKYISTÄ

- DML: tietojen ylläpitoon INSERT INTO, UPDATE ... SET ja DELETE ja hakuihin SELECT ... FROM
- DDL: CREATE, ALTER JA DROP
- Valtuuksien käsittelyyn GRANT ja REVOKE
- Tapahtumankäsittelyn ohjaamiseen BEGIN [TRANSACTION] ja END, SET TRANSACTION ISOLATION LEVEL, COMMIT ja ROLLBACK
- Tietokantarajapinnat (toteutus pitkälti tuotekoht.)

MITÄ SEURAAVAKSI?

- SQL-kielen ohjelmointikäyttö
- Proseduurit, herättimet (eli triggerit) ja omat funktiot
- API-liittymät
- Muut SQL:n erityispiirteet kuten XML:n käyttö SQL-kannoissa
- Ks. Ari Hovin kirja, luku 5