# Tietokanta ja SQL perusteet

## Mitä on tietokanta

**IT-alan ulkopuolella** sana *tietokanta* voi tarkoittaa yksinkertaisesti kokoelmaa tietoa, joka on järjestetty niin, että sitä voidaan käyttää ja hakea helposti. Se ei välttämättä viittaa digitaaliseen järjestelmään. Esimerkiksi:

- **Kirjaston kortisto**: Kirjojen tiedot (tekijä, nimi, sijainti) järjestettynä.
- **Yrityksen paperiarkisto**: Asiakastiedot kansioissa.
- **Tutkimusaineisto**: Excel-taulukko, jossa on mittaustuloksia.

Eli yleisessä merkityksessä tietokanta = **järjestetty tietokokoelma**, ei välttämättä tekninen tai digitaalinen.

Toisaalta IT-alalla tietokanta on järjestelmä, joka tallentaa, hallinnoi ja hakee tietoa tehokkaasti. Se on kuin digitaalinen arkisto, jossa dataa säilytetään rakenteellisesti niin, että sitä voidaan käyttää moniin tarkoituksiin.

**Keskeiset piirteet tietokannasta:**

- **Rakenteellinen tallennus:** Data on yleensä taulukoissa (rivit ja sarakkeet), mutta voi olla myös dokumentteina, avain-arvo -pareina tai graafeina.
- **Tietokantajärjestelmä tai tietokantahallintajärjestelmä (DBMS):** Ohjelmisto, joka hallitsee tietokantaa (esim. MySQL, PostgreSQL, Oracle).
- **Kyselykieli:** Usein käytetään SQL:ää (Structured Query Language) tietojen hakemiseen ja muokkaamiseen.
- **Hyödyt:** Nopea haku, tietojen eheys, samanaikainen käyttö, varmuuskopiointi.

Tietokantoja käytetään, koska ne tarjoavat tehokkaan ja luotettavan tavan hallita suuria määriä tietoa.

- **Keskitetty tiedonhallinta**: Kaikki data on yhdessä paikassa, mikä vähentää päällekkäisyyksiä ja virheitä.
- **Nopea ja helppo haku**: Voit hakea tietoa sekunneissa SQL-kyselyillä, sen sijaan että selaisit paperiarkistoja tai satunnaisia tiedostoja.

- **Tietojen eheys ja tarkkuus**: Tietokannat varmistavat, että data pysyy oikeassa muodossa (esim. syntaksisäännöt, viite-eheys).
- **Samanaikainen käyttö**: Useat käyttäjät voivat käyttää ja päivittää tietoja yhtä aikaa ilman ristiriitoja.
- **Turvallisuus**: Käyttöoikeudet ja salaus suojaavat tietoja luvattomalta käytöltä.
- **Skaalautuvuus**: Kun dataa tulee lisää, tietokanta pystyy käsittelemään sen ilman suuria muutoksia rakenteeseen.
- **Varmuuskopiointi ja palautus**: Tietokannat tukevat automaattisia varmuuskopioita ja palautusta virhetilanteissa.

# Tietokantahallintajärjestelmä

Tietokantahallintajärjestelmä (DBMS, **Database Management System**) on ohjelmisto, joka hallitsee tietokantaa ja sen käyttöä. Se toimii välikätenä käyttäjän (tai sovelluksen) ja tietokannan välillä.

## Keskeiset tehtävät DBMS:ssä

1. **Tietojen tallennus ja haku**
   - Mahdollistaa datan lisäämisen, päivittämisen, poistamisen ja hakemisen.
2. **Tietojen eheys ja johdonmukaisuus**
   - Varmistaa, että data pysyy oikeassa muodossa ja säännöt (esim. viite-eheys) toteutuvat.
3. **Samanaikainen käyttö**
   - Useat käyttäjät voivat käyttää tietokantaa yhtä aikaa ilman ristiriitoja.
4. **Tietoturva**
   - Käyttöoikeuksien hallinta, salaus ja autentikointi.
5. **Varmuuskopiointi ja palautus**
   - Suojaa tietoja virhetilanteissa.
6. **Käyttää kyselykieltä**
   - Kyselykieli on kieli, jota käytetään tietokannan kanssa kommunikointiin – eli tietojen hakemiseen, lisäämiseen, päivittämiseen ja poistamiseen. Se on kuin "ohjeet", joilla kerrotaan tietokantajärjestelmälle, mitä halutaan tehdä. Usein SQL, jolla käyttäjä voi hakea ja muokata tietoja.

"Many databases start as a list in a word-processing program or spreadsheet. As the list grows bigger, redundancies and inconsistencies begin to appear in the data. The data becomes hard to understand in list form, and there are limited ways of searching or pulling subsets of data out for review. Once these

problems start to appear, it's a good idea to transfer the data to a database created by a database management system (DBMS)"

Lähde: Microsoft Database Basics

## Esimerkkejä DBMS-ohjelmistoista

IT alalla käytetään useita eri tietokantahallintajärjestelmiä:

| Järjestelmä | Tyyppi | Lisenssi | Käyttötarkoitus | Vahvuudet |
|---|---|---|---|---|
| **Oracle Database** | Relaatiotietokanta | Kaupallinen | Suuret yritykset, kriittiset sovellukset | Erittäin skaalautuva, korkea tietoturva, monimallinen tuki (SQL + JSON), AI-pohjainen automaatio |
| **MySQL** | Relaatiotietokanta | Avoin lähdekoodi | Web-sovellukset, CMS | Helppokäyttöinen, suosittu web-sovelluksissa, laaja yhteisötuki |
| **PostgreSQL** | Relaatiotietokanta | Avoin lähdekoodi | Kehittäjät, analytiikka | Hyvä integraatio Microsoft-ekosysteemiin, vahva tietoturva, hybridipilvi |
| **Microsoft SQL Server** | Relaatiotietokanta | Kaupallinen | Yritykset, BI ja raportointi | ACID-tuki, laajennettavuus, JSON-tuki, PostGIS paikkatietolaajennus |
| **MongoDB** | NoSQL (dokumentti) | Avoin lähdekoodi | Big Data, joustavat rakenteet | Joustava, skaalautuva, sopii suurille datamäärille |

| Järjestelmä | Tyyppi | Lisenssi | Käyttötarkoitus | Vahvuudet |
|---|---|---|---|---|
| | | | | ja pilvipohjaisiin ratkaisuihin |
| **Redis** | NoSQL (avain-arvo) | Avoin lähdekoodi | Välimuisti, reaaliaikaiset sovellukset | Erittäin nopea, sopii välimuistiin ja reaaliaikaisiin sovelluksiin |
| **Snowflake** | Pilvipohjainen | Kaupallinen | Data-analytiikka, datavarastointi | Erinomainen datavarastointiin ja analytiikkaan, skaalautuu hyvin pilvessä |
| **Elasticsearch** | NoSQL (hakumoottori) | Avoin lähdekoodi | Hakupalvelut, logien analyysi | Tehokas haku ja analytiikka, käytetään usein lokien ja tekstin analysointiin |
| **IBM Db2** | Relaatiotietokanta | Kaupallinen | Enterprise-järjestelmät | Vahva enterprise-tuki, korkea tietoturva |
| **SQLite** | Relaatiotietokanta | Avoin lähdekoodi | Mobiilisovellukset, sulautetut järjestelmät | Ei vaadi palvelinta, erittäin kevyt |

**Huomio:** PostgreSQL on tällä hetkellä kehittäjien keskuudessa suosituin avoimen lähdekoodin DBMS, kun taas Oracle hallitsee enterprise-markkinoita. Pilvipohjaiset ratkaisut kuten Snowflake kasvavat nopeasti.

Lähde: The 10 best databases of 2025: features and latest changes

# Relaatiotietokanta

**Relaatiotietokanta** on tietokantatyyppi, jossa data tallennetaan **taulukoihin**, ja taulut usein liittyvät toisiinsa suhteiden avulla. **Relaatio** nimi tulee matemaattisesta relaatiosta, joka tarkoittaa joukkoa järjestettyjä monikkoa. Relaatio esitetään tyypillisesti taulukkona, jossa rivit edustavat tietueita ja sarakkeet attribuutteja. Eli matemaattisesti relaatio on taulukko itsestään, mutta usein seköitetään termi siitä, että tässä mallissa taulut voivat liittyä toisiinsa.

- **Taulukkorakenne:** Data on riveissä (tietueet) ja sarakkeissa (attribuutit).
- **Suhdet:** Taulut voivat viitata toisiinsa avainten avulla (esim. asiakas → tilaukset).
- **SQL-kieli:** Relaatiotietokantoja hallitaan yleensä SQL:llä.
- **ACID-periaatteet:** Takaa tietojen eheys ja luotettavuus (Atomicity, Consistency, Isolation, Durability).

Esimerkki:

- Taulu **Asiakkaat**: (ID, Nimi, Kaupunki)
- Taulu **Tilaukset**: (TilausID, AsiakasID, Tuote)
  → **AsiakasID** yhdistää tilauksen oikeaan asiakkaaseen.

Relaatiotietokantamalli on edelleen laajimmin käytetty tietokantamalli, erityisesti liiketoimintasovelluksissa, koska se tarjoaa tehokkaan tavan järjestää ja hakea dataa monimutkaisissa suhteissa.

# NoSQL-tietokanta (ei relaatiotietokanta)

**NoSQL-tietokanta** on tietokantatyyppi, joka ei käytä perinteistä relaatiotietokantamallia. NoSQL tarkoittaa "Not Only SQL", mikä viittaa siihen, että nämä tietokannat voivat käyttää muita tietomalleja kuin taulukoita ja sarakkeita. NoSQL-tietokannat on suunniteltu käsittelemään suuria määriä hajautettua dataa ja tarjoamaan joustavuutta skaalautuvuuden ja suorituskyvyn suhteen.

Keskeiset piirteet:

- **Ei taulukkorakennetta:** Data tallennetaan esim. dokumentteina, avain-arvo -pareina, sarakkeina tai graafeina.
- **Joustava skeema:** Ei tarvitse ennalta määriteltyä rakennetta, mikä sopii muuttuvaan dataan.
- **Skaalautuvuus:** Suunniteltu käsittelemään suuria datamääriä ja hajautettuja järjestelmiä.

- **Korkea suorituskyky:** Hyvä valinta reaaliaikaisiin sovelluksiin ja Big Dataan.

NoSQL-tietokannat voidaan jakaa useisiin tyyppeihin, joista yleisimmät ovat:

- **Dokumenttipohjaiset** (esim. MongoDB, CouchDB): Tallentavat dataa dokumentteina, yleensä JSON- tai BSON-muodossa.
- **Avain-arvo -tietokannat** (esim. Redis, DynamoDB): Tallentavat dataa avain-arvo -pareina, mikä mahdollistaa nopean haun.
- **Saraketietokannat** (esim. Cassandra): Hyvä analytiikkaan ja suurille datamäärille.
- **Graafitietokannat** (esim. Neo4j): Suunniteltu verkostoille ja suhteille.

Milloin käytetään NoSQL:ia?

- Kun data on **epäsäännöllistä** tai muuttuu usein.
- Kun tarvitaan **suuri skaalautuvuus** (esim. miljoonia käyttäjiä).
- Kun halutaan **korkea suorituskyky** ja hajautettu arkkitehtuuri.

# SQL

**SQL** (Structured Query Language) on **kyselykieli**, jota käytetään relaatiotietokantojen hallintaan ja käsittelyyn. Se on standardoitu kieli, jonka avulla voidaan:

- **Hakea tietoa**: `SELECT`
- **Lisätä tietoa**: `INSERT`
- **Päivittää tietoa**: `UPDATE`
- **Poistaa tietoa**: `DELETE`
- **Luoda ja muokata rakenteita**: `CREATE`, `ALTER`, `DROP`

SQL:n keskeiset ominaisuudet:

- **Deklaratiivinen kieli**: Kerrot mitä haluat, ei miten se tehdään.
- **Standardi**: Käytössä lähes kaikissa relaatiotietokannoissa (Oracle, MySQL, PostgreSQL, SQL Server), mutta joka DBMS käyttää oma versiotaan, jossa on pieniä eroja muiden kanssa.
- **Laajennettavuus**: Monet tietokannat lisäävät omia laajennuksia SQL:ään.

Esimerkki SQL-kyselystä:

```sql
SELECT nimi, osoite FROM asiakkaat WHERE kaupunki = 'Helsinki';
```

Tämä hakee kaikkien Helsingin asiakkaiden nimet ja osoitteet.

# SQLite

**SQLite** on kevyt ja erittäin suosittu **relaatiotietokantahallintajärjestelmä**, joka eroaa monista muista DBMS-järjestelmistä siinä, että se **ei vaadi erillistä palvelinta**. Se toimii kirjastona, joka tallentaa koko tietokannan **yhteen tiedostoon** (tai vain muistiin).

Keskeiset ominaisuudet:

- **Palvelimeton**: Ei tarvita erillistä tietokantapalvelinta – sovellus käyttää tietokantaa suoraan.
- **Kevyt ja nopea**: Sopii pieniin ja keskisuuriin sovelluksiin.
- **Yksi tiedosto**: Koko tietokanta tallentuu yhteen `.sqlite`- tai `.db`-tiedostoon.
- **Avoin lähdekoodi**: Vapaa käyttää ja jakaa.
- **SQL-tuki**: Käyttää standardia SQL-kieltä.

Missä käytetään?

- Mobiilisovellukset (Android, iOS)
- Sulautetut järjestelmät
- Pienet desktop-sovellukset
- Prototyypit ja testaus

Hyödyt:

- Helppo ottaa käyttöön (ei asennusta, ei konfigurointia)
- Erittäin kevyt ja nopea
- Luotettava ja vakaa

Rajoitukset:

- Ei sovellu hyvin erittäin suuriin tai hajautettuihin järjestelmiin
- Vähemmän ominaisuuksia kuin enterprise-tietokannoissa (esim. Oracle)
- SQLite ei pakota tyyppirajoituksia, mikä voi johtaa tietojen epäjohdonmukaisuuksiin, jos sovellus ei huolehdi siitä itse.

SQLite voidaan lataa ilmaiseksi sen virallisilta sivuilta, mutta useimmat ohjelmointikielet, kuten Python, sisältävät SQLite-tuen valmiiksi. Myös sellaiset työkalut kuin DB Browser for SQLite sisältävät SQLite:n osana asennusta. Eli ei välttämättä tarvitse erikseen ladata sitä.

## SQLite:n CLI (Command Line Interface)

SQL-komentojen lisäksi SQLite-CLI hyväksyy omia komentoja, jotka ovat tarkoitettu hallitsemaan ohjelman ominaisuuksia. Esimerkiksi:

- Apua: `.help`
- Avata tietokantatiedostoa: `.open`
- Listata taulua: `.tables`
- Tulosta taulun järjestely: `.schema taulu`
- Aseta otsikot päälle: `.headers on`
- Vaihtaa tulosten muotoa: `.mode box` | `table` | `json` | `csv` | …
- Pakottaa vierasavaimien tarkistaminen: `PRAGMA foreign_keys=1`
- Poistua SQLitestä: `.quit` tai `.exit`

SQLite:n asetuksen tiedosto on `~/.sqliterc`, johon voidaan laittaa valmiina haluttuja asetuksia.

## DB Browser for SQLite

DB Browser for SQLite on graafinen työkalu, jolla voidaan hallinnoida SQLiten tietokantoja. Voidaan sitä tehdä graafisella tavalla tai voidaankin myös siellä suorittaa SQL kyselyä.

Ohjelma löytyy myös *Portable*-appina.

# SQL perusteita

## Kommentit

SQL:ssä kommentit voidaan lisätä kahdella tavalla:

- Yhden rivin kommentti alkaa `--` merkeillä. Kaikki merkit rivillä `--` jälkeen ovat kommentteja.

```
-- Tämä on yhden rivin kommentti
SELECT * FROM clients; -- Tämä hakee kaikki asiakkaat
```

- Monirivinen kommentti aloitetaan `/*` merkeillä ja lopetetaan `*/` merkeillä.

```
/* Tämä on
monirivinen kommentti */
SELECT * FROM clients;
```

# SELECT

Kuten on jo mainittu, SQL-kielellä voidaan soveltaa eri toimenpiteettä, mutta yleisin käytäntö on saada tietoa tietokannasta, eli "lukea" tietokantaa. Kyseessä on siis `SELECT` -komento.

`SELECT` perusmuoto on:

```
SELECT expression
```

Eli voidaan tehdä jotain kuin

```
SELECT 2+2
```

mutta tavallisesti halutaan hakea tietoa tietokannasta, joten tehdään:

```
SELECT sarake1, sarake2, ...
FROM taulun_nimi;
```

SQL lauseet päätetään `;`-merkkillä ja lauseet voivat käyttää useita riveja. Sisennystä ei oteta myöskään huomioon.

Eli yleisesti, komennon perusmuoto on:

```
SELECT (mitä haetaan) FROM (mistä taulusta);
```

Esim:

```
SELECT first_name, last_name
FROM clients;
```

Jos halutaan saada kaikki mahdolliset sarakkeet taulusta, käytetään `*` SELECTin yhteydessä.

```
SELECT *
FROM clients;
```

## Sarakkeen uudelleennimeäminen kyselyssä

Sarakkeille voidaan antaa eri nimeä kyselyissä käyttämällä `AS` -avainsanaa. Esim:

```
SELECT first_name AS etunimi, last_name AS sukunimi
FROM clients;
```

`AS` -osa on kuitenkin välinnäinen:

```
SELECT first_name etunimi, last_name sukunimi
FROM clients;
```

SQL:ssä **literal-teksti** tarkoittaa kiinteää arvoa, kuten merkkijonoa, numeroa tai päivämäärää, joka kirjoitetaan suoraan kyselyyn. Niihin käytetään **yksinkertaisia heittomerkkejä** `''`. Numerot kirjoitetaan ilman heittomerkkejä. **Tuplamerkit** `" "` SQL-standardissa käytetään **sarakkeiden tai taulujen nimien ympärillä** (tunnisteet/identifiers), jos niissä on välilyöntejä tai varattuja sanoja. Joissakin tietokannoissa (esim. MySQL) voi käyttää **backtick**-merkkejä (``) tunnisteiden ympärillä.

Esim:

```
SELECT "first_name" 'Etunimi', "last_name" 'Sukunimi', "last_login" 'Viimeinen kirjautuminen'
FROM "clients";
```

On myös mahdollista yhdistää tekstejä `||` operaattorilla (concatenaatio). Esim:

```sql
SELECT first_name || ' ' || last_name AS full_name
FROM clients;
```

## WHERE

`WHERE` -ehtolauseke suodattaa rivejä, jotka täyttävät tietyt ehdot. Se lisätään `SELECT` -lauseeseen `FROM` -osan jälkeen määrittämään, mitkä rivit palautetaan.

```sql
SELECT first_name, last_name
FROM clients
WHERE last_name = 'Finnberg';
```

Ehtolausekkeella voidaan käyttää eri operaattoria:

| Toimenpide | MariaDB | SQLite | PostgreSQL | Huomiottava | Esimerkki (SQL) |
|---|---|---|---|---|---|
| **Yhtä suuri kuin** | `=` | `=` | `=` | Same in all | `ikä = 30` |
| **Eri suuri kuin** | `!=` or `<>` | `!=` or `<>` | `!=` or `<>` | All support both forms | `ikä <> 30` |
| **Suurempi kuin** | Same in all | | | | `ikä > 30` |
| **Pienempi kuin** | `<` | `<` | `<` | Same in all | `ikä < 30` |
| **Suurempi tai yhtä suuri kuin** | `>=` | `>=` | `>=` | Same in all | `ikä >= 30` |
| **Pienempi tai yhtä suuri kuin** | `<=` | `<=` | `<=` | Same in all | `ikä <= 30` |
| **Null-turvallinen vertailu** | `<=>` | `IS` | `IS NOT DISTINCT FROM` | PostgreSQL uses `IS NOT DISTINCT FROM` for null-safe equality | `ikä IS NULL` |

| Toimenpide | MariaDB | SQLite | PostgreSQL | Huomiottava | Esimerkki (SQL) |
|---|---|---|---|---|---|
| **Onko arvo NULL** | `IS NULL` | `IS NULL` | `IS NULL` | Same in all | `ikä IS NULL` |
| **Arvo ei ole NULL** | `IS NOT NULL` | `IS NOT NULL` | `IS NOT NULL` | Same in all | `ikä IS NOT NULL` |
| **Arvo tietyllä välillä** | `BETWEEN ... AND ...` | `BETWEEN ... AND ...` | `BETWEEN ... AND ...` | Same in all | `ikä BETWEEN 30 AND 40` |
| **Arvo kuuluu joukkoon** | `IN (...)` | `IN (...)` | `IN (...)` | Same in all | `ikä in (30, 42, 57)` |
| **Merkkijonon osittainen vastaavuus** (`%` / `_`) | `LIKE` (case-**insensitive** by default) | `LIKE` (case-**insensitive** by default) | `LIKE` (case-**sensitive** by default) | PostgreSQL is case-**sensitive** unless using `ILIKE` | `nimi LIKE 'A%'` |
| **Merkkikokoista riippumaton merkkijonon osittainen vastaavuus** | ❌ Not supported | ❌ Not supported | `ILIKE` | PostgreSQL-only, case-**insensitive** LIKE | - |
| **Ei vastaa mallia** | `NOT LIKE` | `NOT LIKE` | `NOT LIKE` | Same in all | `nimi NOT LIKE 'A%'` |
| **Vastaa säännöllistä lauseketta** | `REGEXP` or `RLIKE` | `REGEXP` (user-defined) | `~` (POSIX regex) | PostgreSQL uses `~`, `~*`, `!~`, `!~*` for regex | - |

| Toimenpide | MariaDB | SQLite | PostgreSQL | Huomiottava | Esimerkki (SQL) |
|---|---|---|---|---|---|
| **Ei vastaa säännöllistä lauseketta** | `NOT REGEXP` | `NOT REGEXP` (if defined) | `!~` (case-sensitive), `!~*` (insensitive) | PostgreSQL has built-in regex negation | - |
| **Merkkijonon osittainen vastaavuus** (`*` / `?`) | ❌ Not supported | `GLOB` (case sensitive) | ❌ Not supported | SQLite-only | `nimi GLOB 'A*'` |

### Mikä on null-turvallinen vertailu?

Normaali vertailu `=` ei toimi `NULL`-arvojen kanssa:

```
SELECT * FROM users WHERE age = NULL; -- Ei palauta mitään!
```

**Null-turvallinen vertailu** ottaa huomioon myös `NULL`-arvot:

- **MariaDB**: `age <=> NULL`
- **PostgreSQL**: `age IS NOT DISTINCT FROM NULL`
- **SQLite**: `age IS NULL` (ei suoraa vastinetta `<=>`:lle)

MariaDB:n operaatorit: https://mariadb.com/kb/en/operators/
SQLiten operaatorit: https://www.tutorialspoint.com/sqlite/sqlite_operators.html
PostgreSQL:n operaatorit: https://www.postgresql.org/docs/current/functionsa.html

SQL-kielessä matemaattiset operaattorit ovat melko yksinkertaisia ja niitä käytetään aritmeettisiin laskutoimituksiin. Tässä on taulukko yleisimmistä matemaattisista operaattoreista:

| Operaattori | Kuvaus | Esimerkki |
|---|---|---|
| `+` | Yhteenlasku | `SELECT 5 + 3;` → 8 |

| Operaattori | Kuvaus | Esimerkki |
|---|---|---|
| `-` | Vähennys | `SELECT 5 - 3;` → 2 |
| `*` | Kertolasku | `SELECT 5 * 3;` → 15 |
| `/` | Jakolasku | `SELECT 10 / 2;` → 5 |
| `%` | Jakojäännös (modulo) | `SELECT 10 % 3;` → 1 |

Näitä operaattoreita voi käyttää myös sarakkeiden arvojen kanssa, esim. `SELECT price * quantity FROM orders;`.
SQLitessa jakolaskun paluarvo riippuu siitä, onko käytetty kokonaislukuja vai liukulukuja.

```
SELECT 10 / 4;      -- Tulostaa 2 (kokonaisluku)
SELECT 10.0 / 4;    -- Tulostaa 2.5 (liukuluku)
```

## AND, OR, NOT

Useita ehtoja voidaan yhdistellä loogisilla operaattoreilla `AND`, `OR` ja `NOT`.

```
SELECT first_name, last_name
FROM clients
WHERE city = 'Helsinki' AND age > 30;
```

Kuten aina, kannattaa käyttää sulkuja selkeyttämään ehtoja monimutkaisemmissa lausekkeissa.

```
SELECT first_name, last_name
FROM clients
WHERE (city = 'Helsinki' OR city = 'Espoo') AND age > 30;
```

## DISTINCT

`DISTINCT` on avainsana, jota käytetään poistamaan duplikaatit tulosjoukosta. Kun käytätetään `DISTINCT`-lausetta, se varmistaa, että palautetut rivit ovat ainutlaatuisia valittujen sarakkeiden osalta.
Perussyntaksi on:

```sql
SELECT DISTINCT sarake
FROM table_name;
```

Esimerkki:

```sql
SELECT DISTINCT city
FROM clients;
```

**Huomioita:**

Voidaan käyttää `DISTINCT` useamman sarakkeen kanssa, tällöin sarakkeiden yhdistelmä on ainutlaatuinen.

```sql
SELECT DISTINCT city, age
FROM clients;
/*
city     age
-------- ---
Helsinki  32
Helsinki  25
Espoo     25
Espoo     37
```

## Sisäänrakennetut SQL-funktiot

Tietokantahallintajärjestelmiä sisältävät funktiota helppottamaan eri tehtäviä. Mitkä funktiota ovat saatavilla ja miten niitä käytetään riippuu TKHJ:sta.

- SQLite **perusfunktiot (Scalar Functions)**
  Nämä palauttavat yhden arvon syötteiden perusteella:
  - `abs(X)` – Palauttaa arvon itseisarvon
  - `coalesce(X, Y, ...)` – Palauttaa ensimmäisen ei-NULL-arvon
  - `ifnull(X, Y)` – Palauttaa X, ellei se ole NULL, muuten Y
  - `nullif(X, Y)` – Palauttaa NULL, jos X = Y, muuten X
  - `length(X)` – Merkkijonon pituus
  - `lower(X)`, `upper(X)` – Muuntaa pieniksi/suuriksi kirjaimiksi

  - `substr(X, Y, Z)` – Merkkijonon osajono
  - `replace(X, Y, Z)` – Korvaa Y:n Z:llä merkkijonossa X
  - `trim(X)`, `ltrim(X)`, `rtrim(X)` – Poistaa välilyöntejä tai merkkejä
  - `typeof(X)` – Palauttaa arvon tietotyypin
  - `quote(X)` – Palauttaa SQL-lainatun version arvosta
- SQLite **päivämäärä- ja aika-funktiot**
  - `date(...)`, `time(...)`, `datetime(...)`
  - `julianday(...)`, `strftime(...)`, `current_date`, `current_time`, `current_timestamp`
- SQLite **matemaattiset funktiot**
  - `round(X, Y)` – Pyöristää X:n Y desimaaliin
  - `random()` – Palauttaa satunnaisen kokonaisluvun
  - `randomblob(N)` – Palauttaa satunnaisen binääridatan
  - `sign(X)` – Palauttaa arvon etumerkin
- SQLite **merkkijonofunktiot**
  - `char(X1, X2, ...)` – Unicode-merkkejä koodipisteistä
  - `instr(X, Y)` – Palauttaa Y:n sijainnin X:ssä
  - `concat(X, ...)`, `concat_ws(SEP, X, ...)` – Merkkijonojen yhdistäminen
- SQLite **Aggregaattifunktiot**
  - `count(X)`, `sum(X)`, `avg(X)`, `min(X)`, `max(X)`
  - `total(X)` – Kuten `sum`, mutta palauttaa aina 0, ei NULL

Koko listaa virallisessa dokumentaatiossa: SQLite Built-in Functions

## ORDER BY

`ORDER BY` -lauseke järjestää tulokset tietyn sarakkeen tai sarakkeiden perusteella. Oletuksena järjestys on nouseva (ASC), mutta voidaan määrittää myös laskeva (DESC).

```
SELECT first_name, last_name
FROM clients
ORDER BY last_name ASC, first_name DESC;
```

## LIMIT

`LIMIT` -lauseke rajoittaa palautettavien rivien määrää.

```sql
SELECT first_name, last_name
FROM clients
LIMIT 10;
```

## OFFSET

`OFFSET` -lauseke ohittaa tietyn määrän rivejä ennen tulosten palauttamista.

```sql
SELECT first_name, last_name
FROM clients
LIMIT 10 OFFSET 5;
```

## GROUP BY

`GROUP BY` -lauseke ryhmittelee rivit tietyn sarakkeen tai sarakkeiden perusteella, usein yhdessä aggregaattifunktioiden kanssa.

```sql
SELECT city, COUNT(*) AS asiakas_maara
FROM clients
GROUP BY city;
```

## HAVING

`HAVING` -lauseke suodattaa ryhmiteltyjä tuloksia, toisin kuin `WHERE`, joka suodattaa ennen ryhmittelyä.

```sql
SELECT city, COUNT(*) AS asiakas_maara
FROM clients
GROUP BY city
HAVING asiakas_maara > 5;
```

## JOIN

`JOIN` -lauseke yhdistää rivejä kahdesta tai useammasta taulusta niiden välisen suhteen perusteella.

```sql
SELECT clients.first_name, orders.order_date
FROM clients JOIN orders
ON clients.client_id = orders.client_id;
```

### Dot notation

Dot notation tarkoittaa piste-merkintätapaa, jota käytetään SQL:ssä viittaamaan tietokannan objekteihin, kuten tauluihin ja sarakkeisiin, erityisesti kun käsitellään useita tauluja tai skeemoja. Piste ( `.` ) erottaa objektin nimen sen kontekstista.

Edellisessä esimerkissä se auttaa erottamaan `client_id` -sarakkeen, joka kuuluu `clients` -tauluun, ja `client_id` -sarakkeen, joka kuuluu `orders` -tauluun.

Jos ei ole epäselvyyttä, sarakkeen nimeä voidaan käyttää ilman taulun nimeä.

```sql
SELECT first_name, order_date
FROM clients JOIN orders
ON clients.client_id = orders.client_id;
```

### Aliakset tauluille

Tauluille voidaan antaa lyhyempi nimi (alias) helpottamaan viittauksia, erityisesti monimutkaisissa kyselyissä.

```sql
SELECT c.first_name, o.order_date
FROM clients AS c JOIN orders AS o
ON c.client_id = o.client_id;
```

Samaa kuiin aikasemmin voidaan tehdä ilman `AS` -avainsanaa:

```sql
SELECT c.first_name, o.order_date
FROM clients c JOIN orders o
ON c.client_id = o.client_id;
```

# CREATE DATABASE

`CREATE DATABASE` -lauseke luo uuden tietokannan.

SQLite:ssä tietokanta luodaan yksinkertaisesti avaamalla uusi tiedosto, joten `CREATE DATABASE` -komentoa ei käytetä.

```
CREATE DATABASE tietokanta_nimi;
```

# DROP DATABASE

`DROP DATABASE` -lauseke poistaa olemassa olevan tietokannan ja kaikki sen sisältämät tiedot.

SQLite:ssä tietokanta poistetaan yksinkertaisesti poistamalla tietokantatiedosto, joten `DROP DATABASE` -komentoa ei käytetä.

```
DROP DATABASE tietokanta_nimi;
```

# CREATE TABLE

```
CREATE TABLE [IF NOT EXIST] nimi (
    sarake1    tietotyyppi     [sarakkeen rajoitukset],
    sarake2    tietotyyppi     [sarakkeen rajoitukset],
    [taulun rajoitukset]
);
```

komennolla luodaan uusi (tyhjä) taulu. Komennon syntaksi voi vaihdella TKHJ:n mukaan

Tietotyypit vaihtelevat TKHJ:n mukaan:

- SQLite: https://www.sqlite.org/datatype3.html
- MariaDB: https://mariadb.com/kb/en/data-types/
- PostgreSQL: https://www.postgresql.org/docs/current/datatype.html

Yleisiä tietotyyppejä:

| Tietotyyppi | Kuvaus |
|---|---|
| `INTEGER` | Kokonaisluku |
| `REAL` | Liukuluku |
| `TEXT` | Merkkijono |
| `BLOB` | Binääridata |
| `DATE` | Päivämäärä |
| `DATETIME` | Päivämäärä ja aika |
| `BOOLEAN` | Totuusarvo (1 tai 0) |

## PRIMARY KEY

PRIMARY KEY määrittelee sarake tai sarakkeet, jotka toimivat taulun pääavaimena

```
CREATE TABLE taulun_nimi (
    sarake1 integer PRIMARY KEY,
    ...
);
```

```
CREATE TABLE taulun_nimi (
    sarake1 integer,
    sarake2 integer,
    ...
    PRIMARY KEY (sarake1, sarake2)
);
```

PRIMARY KEY sisältää periaatteessa NOT NULL ja UNIQUE rajoitukset.
Huom! SQLite:ssä pitää lisätä NOT NULL joka tapauksessa.

PRIMARY KEY hyväksy sellainen automaattisesti nouseva numero käsky (AUTOINCREMENT SQLite:ssä, AUTO_INCREMENT MariaDB:ssa)

```
CREATE TABLE taulun_nimi (
    sarake1 integer PRIMARY KEY AUTOINCREMENT,
    ...
);
```

Tällä tavalla ei tarvitse syöttää arvoa tälle sarakkeelle, koska se ottaa automaattisesti seuraava numeroarvo (aloittaa 1:stä). Kuitenkin SQLite:ssä AUTOINCREMENT on valinnainen, koska INTEGER PRIMARY KEY toimii samalla tavalla ilman sitäkin
SQLite:n taulut sisältävät automaattisesti `rowid` sarakkeen, joka toimii samalla tavalla kuin AUTOINCREMENT. Jos taululla on sarake, joka on määritelty `INTEGER PRIMARY KEY`, se toimii `rowid` -sarakkeen aliasena. Jos taululla ei ole `INTEGER PRIMARY KEY` saraketta, SQLite luo automaattisesti **piilotetun** `rowid` -sarakkeen.

## FOREIGN KEY (vain taululle)

FOREIGN KEY määrittelee, että sarake on pääavain toisessa taulussa

```
CREATE TABLE taulun_nimi (
    sarake1 integer PRIMARY KEY,
    sarake2 integer,
    ...
    FOREIGN KEY (sarake2) REFERENCES toinen_taulu(toinen_sarake)
    [ON DELETE action]
    [ON UPDATE action]
);
```

"*action*" voi olla esimerkiksi: `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION` tai `CASCADE`
FOREIGN KEY voidaan käyttää useita kertaa taulussa eri sarakkeille.

## UNIQUE

`UNIQUE` määrittää, että ko. sarakkeelle tai sarakkeille ei voi tulla samaa arvoa useita kertoja

```
CREATE TABLE taulun_nimi (
    sarake1 integer UNIQUE,
    ...
);
```

```
CREATE TABLE taulun_nimi (
    sarake1 integer,
    sarake2 integer,
    ...
    UNIQUE (sarake1, sarake2)
);
```

NULL arvot pidetään erillisenä niiden välillä, joten saa olla useita NULL arvoa sarakkeessa.

## NOT NULL (vain sarakkeelle)

NOT NULL määrittää, että ko. sarakkeessa ei saa olla NULL arvoa

```
CREATE TABLE taulun_nimi (
    sarake1 integer NOT NULL,
    ...
);
```

> **Huom!**
>
> SQL standardien mukaan, PRIMARY KEY pitää sisältää NOT NULL, mutta SQLite TKHJ:ssä ei seuraa sitä. Eli SQLitessä tarvitaan NOT NULL PRIMARY KEY yhteydessä.

## CHECK ehto

CHECK määrittää sarakkeelle tai sarakkeille ehtoa/ehtoja, jotka sarakkeen arvot pitää seurata

```
CREATE TABLE taulun_nimi (
    sarake1 integer CHECK (ehtolauseke),
    ...
);
```

```
CREATE TABLE taulun_nimi (
    sarake1 integer,
    sarake2 integer,
    ...
    CHECK (ehtolauseke)
);
```

### DEFAULT arvo (vain sarakkeelle)

DEFAULT määrittää sellainen arvo, joka annetaan sarakkeelle jos INSERT komento ei antaa mitään arvoa sille

```
CREATE TABLE taulun_nimi (
    sarake1 integer DEFAULT 100,
    ...
);
```

## INSERT INTO

INSERT INTO komennolla saadaan lisättyä uuden tietuerivin tauluun.

```
INSERT INTO taulu [(sarake1, sarake2, ...)]
VALUES (arvo1, arvo2, ...);
```

- Jos sarakkeita ei mainita, pitää antaa arvoa kaikille sarakkeille.
- Sarakkeet ei tarvitse olla samassa järjestyksessä, jolla ne on määritetty.
- Useita rivejä voidaan lisätä yhdellä komennolla:

  ```
  INSERT INTO taulu [(sarake1, sarake2, ...)]
  VALUES (arvo1, arvo2, ...), (arvo3, arvo4, ...), ...;
  ```

# UPDATE

UPDATE komennolla päivitetään olemassa olevan tietuerivin arvoja.

```
UPDATE kirjailijat
SET kirjailijat_sukunimi = 'Turunen'     -- Sarake johon muutos kohdistuu
WHERE kirjailijat_tekija_id = 204;       -- Ehto: rivi, mihin muutos kohdistuu
```

Taulu käydään läpi, ja kaikkien ehdon täyttävien tietuerivien arvo2 korvataan annetulla arvolla.

> ⚠ **HUOM!**
>
> Tosi tärkeä! **Älä koskaan unohda** laittaa WHERE –osaa! Muuten **päivität koko taulun** sisällön!

On mahdollista päivittää useita sarakkeita kerralla pilkulla erotettuna.

```
UPDATE opiskelija
SET etunimi = 'Matti',
    sukunimi = 'Meikäläinen'
WHERE opiskelija_id = 123;
```

Ja on myös mahdollista päivittää kenttiä maattemaattisin operaation perusteella.
Tämä voi olla esimerkiksi

- Juokseva numerointi (mahdollisesti ID)

  ```
  UPDATE opiskelija
  SET sarake1 = sarake1 + 1
  ```

- Uusi arvo, joka on johdettu muista arvoista

  ```
  UPDATE opiskelija
  SET ikä = (NOW() - syntymäaika)
  ```

# DELETE FROM

DELETE komennolla poistetaan taulusta ehdon täyttävät tietuerivit.

```
DELETE FROM taulu
WHERE ehto;
```

```
DELETE FROM opiskelijat
WHERE sukunimi='Aaltonen';
```

Taulu käydään läpi, ja kaikki ehdon täyttävät tietueet poistetaan.

> ⚠ **HUOM!**
>
> Tosi tärkeä! **Älä koskaan unohda** laittaa WHERE –osaa! Muuten **poistat koko taulun** sisällön!

# ALTER TABLE

Taulun rakennetta voidaan muuttaa `ALTER TABLE` -lausekkeella.

```
ALTER TABLE taulu_nimi
ADD COLUMN sarake_nimi tietotyyppi [sarakkeen rajoitukset];
```

Muita mahdollisia toimintoja:

- `DROP COLUMN sarake_nimi;` – Poistaa sarakkeen (ei tuettu kaikissa TKHJ:ssa)
- `RENAME TO uusi_taulu_nimi;` – Nimeää taulun uudelleen
- `RENAME COLUMN vanha_sarake_nimi TO uusi_sarake_nimi;` – Nimeää sarakkeen uudelleen (ei tuettu kaikissa TKHJ:ssa)

# DROP TABLE

`DROP TABLE` -lauseke poistaa olemassa olevan taulun ja kaikki sen sisältämät tiedot.

```
DROP TABLE [IF EXISTS] taulu_nimi;
```