Antti Kauppi

PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvien ominaisuuksien vaikutukset virheilmoitusten koettuun hyödyllisyyteen

Tietotekniikan pro gradu -tutkielma

1. heinäkuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Antti Kauppi

Yhteystiedot: antti.p.kauppi@student.jyu.fi

Ohjaaja: Toni Taipalus

Työn nimi: PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvien ominaisuuksien vaikutukset virheilmoitusten koettuun hyödyllisyyteen

Title in English: Effects of Features Related to The Usability of Error Messages on Perceived Usefulness of Error Messages in PostgreSQL Database Management System

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 42+1

Tiivistelmä: Ohjelmointikielten kääntäjien virheilmoituksia on tutkittu useiden vuosikymmenten ajan, mutta tietokannanhallintajärjestelmien virheilmoituksia on tutkittu toistaiseksi hyvin vähän. Tässä tutkimuksessa on tarkoituksena selvittää, mitkä PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoitusten koettuun hyödyllisyyteen aloittelijoiden näkökulmasta. Aiemmin tehdyissä tutkimuksissa virheilmoitusten on todettu olevan muun muassa riittämättömiä, turhauttavia, hyödyttömiä ja hämmentäviä. Virheilmoitusten laatu vaikuttaa merkittävästi muun muassa ohjelmoijien mielialoihin, ohjelmoinnin oppimiseen ja ohjelmistokehitysprojektien tehokkuuteen. Runsaasta tutkimustiedosta huolimatta virheilmoitusten laadussa on edelleen paljon parannettavaa.

Avainsanat: virheilmoitus, virheilmoitusten suunnitteluohjeet, syntaksivirhe, SQL-kieli, tietokannanhallintajärjestelmä

Abstract: Programming error messages have been studied for several decades. Instead, very little research has been done on error messages in database management systems. The purpose of this study is to determine which features related to the usability of error messages in the PostgreSQL database management system affect the perceived usefulness of error messages from the novices' point of view. In previous studies, programming error messages have been described as, for example, inadequate, frustrating, useless and confusing. The quality of error messages has a significant impact on programmers' moods, learning programming and the effectiveness of software development projects. Despite extensive research on programming error messages, there is still much room for improvement in their quality.

Keywords: error message, design guidelines, syntax error, SQL, database management system

Kuviot

Taulukot

Sisältö

1 Johdanto

Ohjelmointikielten kääntäjien virheilmoitusten laadussa on paljon parannettavaa, vaikka virheilmoituksia on tutkittu jo yli puolen vuosisadan ajan. Kääntäjien kehityksessä on panostettu enemmän esimerkiksi suorituskyvyn ja uusien ominaisuuksien lisäämiseen kuin virheilmoitusten laadun parantamiseen. Huonosti suunnitellut virheilmoitukset hankaloittavat ohjelmoijien työtä, ja sitä kautta ne vaikuttavat koko ohjelmistokehitysprojektiin ja sen tuloksena syntyvään lopputuotteeseen eli ohjelmistoon. Virheilmoitukset ovat erityisen hankalia aloitteleville ohjelmoijille. Ohjelmoinnin oppiminen on haastavaa, eivätkä vaikeaselkoiset virheilmoitukset helpota tätä oppimisprosessia. (Becker ym. 2019; Traver 2010)

Tietokannanhallintajärjestelmien virheilmoituksia on tutkittu toistaiseksi hyvin vähän. Tässä tutkimuksessa on tarkoituksena selvittää, mitkä PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoitusten koettuun hyödyllisyyteen. Käytettävyyteen liittyviä ominaisuuksia ovat esimerkiksi positiivisuus ja täsmällisyys. Näiden ominaisuuksien tarkoituksena on muun muassa auttaa käyttäjää virheiden korjaamisessa ja vähentää samojen virheiden tekemistä tulevaisuudessa.

Ohjelmointikielten osalta kirjallisuudessa on esitetty erilaisia ohjenuoria tehokkaiden virheilmoitusten suunnitteluun ja kehittämiseen. Tässä tutkimuksessa on tarkoituksena soveltaa näihin ohjenuoriin sisältyviä ominaisuuksia PostgreSQL-tietokannanhallintajärjestelmän virheilmoituksiin. Tutkimuksessa selvitetään, mitkä virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoitusten koettuun hyödyllisyyteen aloittelijoiden näkökulmasta. Aloittelijat ovat tässä tapauksessa Jyväskylän yliopiston informaatioteknologian tiedekunnan *Tietokannat ja tiedonhallinta* -kurssin opiskelijoita.

Luvussa 2 esitellään määritelmiä tutkimuksen aihealueeseen olennaisesti liittyville käsitteille. Luvussa 3 esitellään virheilmoituksia ja virheiden tekemistä koskevaa aiempaa tutkimusta. Luvussa 4 kuvataan tutkimusasetelma. Tutkimustulokset esitellään luvussa 5, ja tuloksiin liittyvä pohdinta esitetään luvussa 6. Tutkimuksen yhteenveto on tiivistetty lukuun 7.

2 Käsitteet

Luvussa esitellään määritelmiä tutkimuksen aihealueeseen olennaisesti liittyville käsitteille.

2.1 Tietokanta

Tietokannat liittyvät olennaisesti lähes kaikkien yritysten toimintaan nykypäivänä. Suoritamme päivittäin lukuisia toimintoja, joiden seurauksena tietokantaan tallennettua dataa haetaan, päivitetään, lisätään ja poistetaan. Tällaisia toimintoja ovat esimerkiksi rahan tallettaminen ja nostaminen pankissa, lentojen varaaminen sekä ostosten tekeminen verkossa. (Elmasri & Navathe 2017, s. 33–34; Silberschatz ym. 2006, s. 1–2)

Tietokanta on kokoelma toisiinsa liittyvää dataa, ja se kuvastaa reaalimaailmaa tietystä näkökulmasta. Muutokset reaalimaailmassa heijastuvat tietokantaan. Tietokantaan tallennetulla datalla on jokin luontainen merkitys, ja tietokanta on suunniteltu ja toteutettu tiettyä tarkoitusta, käyttäjäryhmää ja sovellusohjelmaa varten. Tietokanta voi olla kooltaan ja monimutkaisuudeltaan millainen tahansa. (Elmasri & Navathe 2017, s. 34–35)

Connollyn & Beggin (2005, s. 15) mukaan tietokanta on jaettu kokoelma loogisesti toisiinsa liittyvää dataa, jota voidaan hyödyntää samanaikaisesti useiden käyttäjien toimesta. Organisaation informaatiotarpeita analysoitaessa pyritään tunnistamaan ne reaalimaailman kohteet ominaisuuksineen, joista halutaan tallentaa dataa tietokantaan. Elmasri & Navathe (2017, s. 36–37) mainitsevat esimerkkinä yliopiston tietojärjestelmän tietokannan, johon tallennetaan tiedot muun muassa opiskelijoista, kursseista ja arvosanoista. Kaikista näistä reaalimaailman kohteista ja niiden välisistä suhteista muodostuu loogisesti toisiinsa liittyvän datan kokoelma.

Darwen (2010, s. 14) määrittelee tietokannan järjestetyksi ja koneellisesti luettavaksi symbolikokoelmaksi. Tietokanta on myös koneellisesti päivitettävissä, ja sitä hyödyntävien käyttäjien tarpeet voivat mahdollisesti vaihdella. Tietokannan voidaan tulkita kuvaavan yrityksen tilaa tietyllä ajanhetkellä, mutta se voi sisältää myös virheitä.

2.2 Tietomalli

Tietokanta noudattaa yhtä tai useampaa tietomallia. Tietomalli määrittää datan rakenteen, operaatiot datan hallintaan ja rajoitteet datan eheyden varmistamiseksi. Tietomallit jaetaan niiden abstraktiotason mukaan korkean tason käsitteellisiin malleihin, loogisen tason esitysmalleihin ja matalan tason fyysisiin malleihin. (Elmasri & Navathe 2017, s. 62–63; Connolly & Begg 2005, s. 43–44)

Käsitteellinen mallintaminen on olennainen vaihe tietokannan suunnittelussa. Sen avulla pyritään löytämään ne reaalimaailman kohteet ominaisuuksineen ja keskinäisine suhteineen, joista halutaan tallentaa dataa tietokantaan. Chenin (1976) esittelemä ER-malli (engl. *Entity-Relationship Model*) on suosittu käsitteellisen tason tietomalli. (Elmasri & Navathe 2017, s. 89–92)

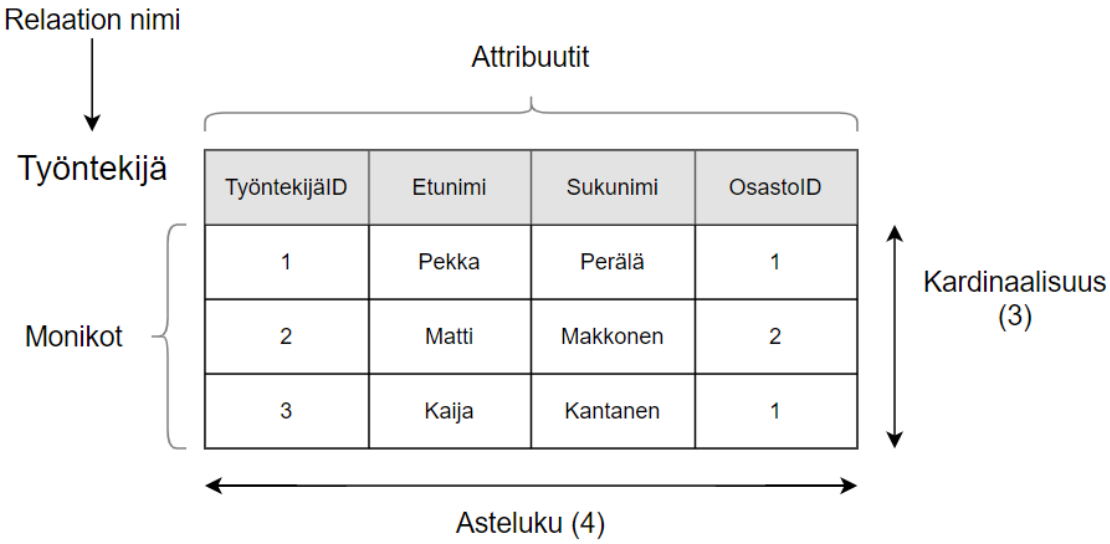
Loogisen tason tietomallit kuvaavat tietokannan sisältämän datan helposti ymmärrettävässä muodossa, mutta ne ovat riippumattomia tietokannan fyysisestä toteutuksesta. Relaatiomalli on eniten käytetty loogisen tason tietomalli, ja sitä käsitellään tarkemmin seuraavassa alaluvussa. Muita loogisen tason tietomalleja ovat esimerkiksi verkko- ja puurakennemallit. (Elmasri & Navathe 2017, s. 63; Foster & Godbole 2016, s. 37)

Fyysisillä tietomalleilla kuvataan tietokannan toteutuksen yksityiskohdat laitteistoläheisellä tasolla. Näitä yksityiskohtia ovat esimerkiksi datan varastointi tietokoneen muistissa levylohkoille ja tietokantahakujen tehostaminen indeksoinnin avulla. (Elmasri & Navathe 2017, s. 63–64)

2.3 Relaatiomalli

Codd (1970) esitteli ensimmäisenä relaatiomallin, jonka teoria perustuu joukko-oppiin. Relaatiomalli mahdollistaa datan riippumattomuuden sovellusohjelmista siten, etteivät muutokset datan järjestämisessä laitteistotasolla vaikuta sovellusohjelmien toimintaan. Relaatiomallilla pyritään torjumaan datan johdonmukaisuuteen ja toisteisuuteen liittyviä haasteita, ja se tarjoaa perustan korkean tason kyselykielelle.

Relaatiomallin mukaan tietokanta koostuu **relaatioista**, jotka esitetään monesti kaksiulotteisina taulukoina. Taulukkomuodossa esitetty relaatio koostuu riveistä eli *monikoista* (engl. *tuples*) ja sarakkeista eli *attribuuteista* (engl. *attributes*). Jokainen monikko kuvaa tyypillisesti tiettyä reaalimaailman kohdetta (esim. työntekijä), ja attribuutit ovat kohteen ominaisuuksia (esim. etunimi ja sukunimi). Relaation *asteluku* (engl. *degree*) määräytyy sen sisältämien attribuuttien määrän mukaan, ja monikkojen määrää kutsutaan puolestaan relaation *kardinaalisuudeksi* (engl. *cardinality*). Relaation nimi ja attribuuttien nimet kuvaavat relaation sisältämää dataa. Kuvio 1 havainnollistaa relaation rakennetta. (Connolly & Begg 2005, s. 72–74)



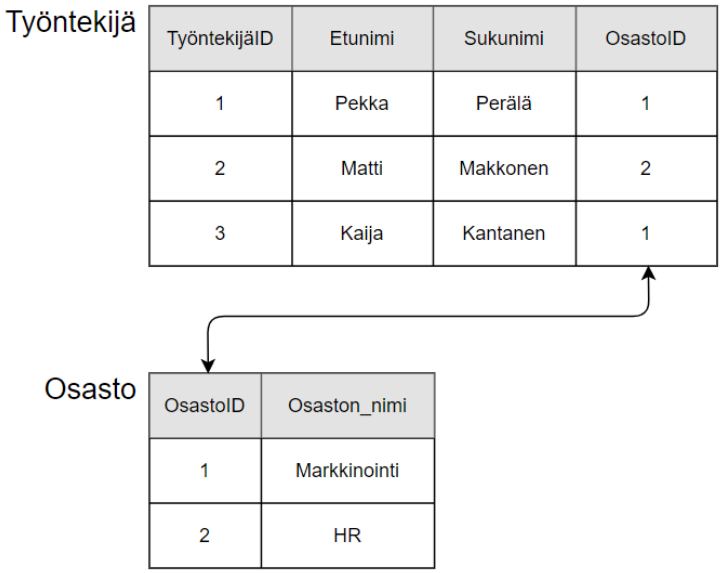
Kuvio 1. Relaatio taulukkona

Coddin (1970) mukaan taulukkona esitetyllä, asteluvultaan *N* relaatiolla on seuraavat ominaisuudet:

- Jokainen rivi edustaa relaation *N*-monikkoa.
- Rivien järjestyksellä ei ole merkitystä.
- Kaikki rivit ovat keskenään erilaisia.
- Sarakkeiden järjestyksellä ei ole merkitystä, mutta attribuutin nimen ja sitä vastaavan arvon yhteys tulee olla yksiselitteisesti tunnistettavissa.
- Sarakkeen merkitys voidaan ainakin osittain päätellä sen nimestä.

Relaatiossa on määritettävä attribuuttijoukko (sisältää 1-N attribuuttia), joka yksiselitteisesti yksilöi kunkin monikon. Tällaista attribuuttijoukkoa kutsutaan relaation **perusavaimeksi** (engl. *primary key*). Esimerkiksi kuviossa 1 relaation perusavain on *TyöntekijäID* eli jokaisella työntekijällä on uniikki työntekijätunnus. Relaatio voi sisältää useamman kuin yhden yksilöivän attribuuttijoukon, ja näitä attribuuttijoukkoja kutsutaan *avainehdokkaiksi* (engl. *candidate key*). Perusavain valitaan avainehdokkaiden joukosta. (Connolly & Begg 2005, s. 78–79)

Viiteavaimeksi (engl. *foreign key*) kutsutaan attribuuttijoukkoa, joka viittaa jonkin toisen relaation avainehdokkaaseen. Yleensä viiteavain viittaa toisen relaation perusavaimeen. Tietyn attribuutin tai attribuuttijoukon esiintyminen useammassa kuin yhdessä relaatiossa kuvaa yleensä näiden relaatioiden monikoiden välistä suhdetta. Kuviossa 2 *Työntekijä*-relaation attribuutti *OsastoID* on viiteavain, joka viittaa *Osasto*-relaation perusavaimeen *OsastoID*. (Connolly & Begg 2005, s. 79)



Kuvio 2. Viiteavain relaatiossa

Relaatiotietokanta koostuu normalisoiduista relaatioista, joilla on toisistaan poikkeavat nimet. **Normalisoinnilla** tarkoitetaan tekniikkaa, jossa pyritään tunnistamaan soveltuva joukko relaatioita tukemaan yrityksen informaatiotarpeita. Normalisoinnin tuloksena relaatiot sisältävät vain tarpeelliset attribuutit, ja loogisesti toisiinsa liittyvät attribuutit

löytyvät samasta relaatiosta. Normalisoinnilla pyritään vähentämään datan toisteisuutta, mutta viiteavainten osalta sitä voi luonnollisesti esiintyä relaatioiden välisten liitosten mahdollistamiseksi. Normalisoinnin myötä tietokannan ylläpito helpottuu ja fyysisen tallennustilan tarve vähenee. Normalisointiprosessi etenee asteittain tarkastelemalla eri *normaalimuotoja*, joita ei tässä luvussa käsitellä tarkemmin. (Date 1995, s. 98; Connolly & Begg 2005, s. 74, 388)

Relaatioalgebra koostuu operaatioista, joissa operaation syötteeksi annetuista relaatioista muodostetaan tulosrelaatio käyttäjän tietotarpeiden mukaan. Relaatiot ovat joukkoja, joten kaikki yleisimmät joukko-operaatiot ovat niihin sovellettavissa. Operaation tuloksena muodostuvaa tulosrelaatiota voidaan edelleen käsitellä relaatioalgebran operaatioiden mukaisesti. Operaatio voi ottaa syötteekseen yhden tai kaksi relaatiota. Yhteen relaatioon kohdistuvia perusoperaatioita ovat *valinta* ja *projektio*. Kahteen relaatioon kohdistuvia perusoperaatioita ovat *yhdiste*, *ristitulo*, *erotus*, *liitos*, *leikkaus* ja *jako*. (Codd 1970; Elmasri & Navathe 2017, s. 269–298; Silberschatz ym. 2006, s. 45–70)

Relaatiokalkyyli (engl. *relational calculus*) on relaatioalgebran ohella toinen relaatiomalliin liittyvä kyselykieli. Siinä määritetään deklaratiivisilla ilmauksilla, mitä tietoa kyselyllä halutaan palauttaa. Relaatiokalkyyli ei niinkään ota kantaa siihen, miten tieto palautetaan. Relaatioalgebra ja relaatiokalkyyli muodostavat perustan korkean tason käyttäjäläheisille kyselykielille, joista SQL on eniten käytetty. SQL-kieltä käsitellään tarkemmin seuraavassa alaluvussa. (Elmasri & Navathe 2017, s. 298)

2.4 SQL-kieli

SQL (engl. *Structured Query Language*) on korkean tason kyselykieli, joka mahdollistaa relaatiotietokannan sisältämän datan hakemisen lisäksi muun muassa datan muokkaamisen ja rakenteen määrittämisen. SQL noudattaa monilta osin relaatioalgebran ja relaatiokalkyylin periaatteita, mutta se on huomattavasti käytännöllisempi ja käyttäjäläheisempi verrattuna jälkimmäisinä mainittuihin kyselykieliin. SQL-kieltä pidetään yhtenä merkittävimmistä tekijöistä relaatiotietokantojen suureen suosioon. (Elmasri & Navathe 2017, s. 207–208; Silberschatz ym. 2006, s. 75)

SQL on standardoitu kieli relaatiotietokannoille, ja standardoinnista ovat vastanneet American National Standards Institute (ANSI) ja International Standards Organization (ISO). SQL-standardin ensimmäinen versio SQL-86 julkaistiin vuonna 1986, ja tämän jälkeen standardia on päivitetty useita kertoja. Esimerkiksi standardin versioon SQL:1999 lisättiin muun muassa oliorelaationaalisia tietokannanhallintajärjestelmiä tukevia ominaisuuksia. Standardissa SQL:2003 päivitettiin edellä mainittua versiota lisäämällä muun muassa XML-yhteensopivia ominaisuuksia. (Eisenberg & Melton 1999; Eisenberg ym. 2004)

Silberschatzin ym. (2006, s. 75–162) sekä Connollyn & Beggin (2005, s. 112–195) mukaan SQL voidaan jakaa SQL-standardiin perustuen neljään alikieleen seuraavasti:

- tietokannan rakenteen määrittely (engl. *Data Definition Language, DDL*),
- datan hallinta (engl. *Data Manipulation Language, DML*),
- tapahtumanhallinta (engl. *Transaction Control Language, TCL*) ja
- valtuuttaminen eli oikeuksien hallinta (engl. *Data Control Language, DCL*).

DML muodostaa merkittävän osan SQL-kielestä. Se sisältää komennot datan hakemiseen (SELECT), lisäämiseen (INSERT), päivittämiseen (UPDATE) ja poistamiseen (DELETE). Tässä luvussa käsitellään tarkemmin ainoastaan SELECT-lauseen rakennetta, koska tutkimuksen empiirisessä osassa tutkittiin nimenomaan virheellisten SELECT-lauseiden aiheuttamia virheilmoituksia. Seuraavat esimerkit perustuvat Silberschatzin ym. (2006, s. 80–103), Connollyn & Beggin (2005, s. 117–155) sekä Elmasrin & Navathen (2017, s. 217–268) esityksiin.

SELECT-lauseella haetaan dataa yhdestä tai useammasta relaatiotietokannan relaatiosta. Relaatiomallin mukaisista termeistä *relaatio*, *monikko* ja *attribuutti* käytetään SQL-kielessä termejä *taulu*, *rivi* ja *sarake*. SELECT-lauseen perusrakenne on seuraava:

```
SELECT <sarakkeet>
  FROM <taulut>
 WHERE <ehdot>;
```

SELECT-lauseen kannalta olennaisimmat relaatioalgebran operaatiot ovat *valinta*, *projektio*, *ristitulo* ja *liitos*. SELECT-osassa määritetään projektioon perustuen ne taulun tai taulujen sarakkeet, joista halutaan dataa tulostauluun. FROM-osassa esitellään ristituloon perustuen taulut, joista dataa haetaan. WHERE-osa on lauseessa valinnainen, ja siinä määritetään valintaan perustuen ehdot, jotka tulostaulun rivien tulee täyttää. Esimerkiksi kuvion 2 *Työntekijä*-relaatiosta haetaan kaikki tiedot kaikista niistä työntekijöistä, joiden etunimi on Pekka, seuraavalla SQL-lauseella:

```
SELECT *
  FROM Työntekijä
 WHERE Etunimi = 'Pekka';
```

SELECT-osassa `*`-merkillä valitaan kaikki FROM-osassa esitellyn taulun sarakkeet tulostauluun. Edellä esitetty SQL-lause palauttaa tulostaulun, joka sisältää yhden rivin Pekka Perälän tiedoilla. Edelleen voitaisiin olla kiinnostuneita esimerkiksi niiden työntekijöiden etu- ja sukunimistä, jotka työskentelevät *Markkinointi*-osastolla. SQL-lause kirjoitetaan tällöin seuraavasti:

```
SELECT Etunimi, Sukunimi
  FROM Työntekijä
 WHERE OsastoID IN
      (SELECT OsastoID
         FROM Osasto
        WHERE Osaston_nimi = 'Markkinointi');
```

Edellä esitetyssä kyselyssä tehdään liitos *Työntekijä*- ja *Osasto*-relaatioiden välillä pääkyselyn WHERE-osassa IN-predikaattia käyttäen. Alikyselyssä haetaan *Markkinointi*-osaston *OsastoID*, jotta tulostauluun saadaan ainoastaan kyseisen osaston työntekijöiden etu- ja sukunimet. Taulujen välisen liitoksen tekemiseen on muitakin tapoja. Liitos voidaan tehdä EXISTS- tai JOIN-predikaatteja käyttäen tai niin sanotusti yksitasoisena vertailuoperaattoria käyttäen. Kyselyn rakenne poikkeaa näissä tavoissa hieman edellä esitetystä, mutta näitä tapoja ei käsitellä tässä luvussa tarkemmin.

SELECT-lauseella haettu data voidaan järjestää tulostaulussa tietyn sarakkeen tai sarakkeiden mukaan ORDER BY -avainsanalla. Tulokset voidaan järjestää joko nousevaan (engl. *ascending*) tai laskevaan (engl. *descending*) järjestykseen ASC- ja DESC-lisämääreillä sarakkeen nimen perässä. Tulokset järjestetään nousevaan järjestykseen, mikäli järjestämistapaa ei määritetä eksplisiittisesti.

GROUP BY -avainsanalla tulokset voidaan ryhmitellä jonkin sarakkeen tai sarakkeiden mukaan. Ryhmittelyä tarvitaan silloin, kun jokin tulostaulun sarake on muodostettu koostefunktiota käyttäen. Esimerkiksi tilanteessa, jossa haluttaisiin tietää yrityksen eri osastoilla työskentelevien työntekijöiden keskiarvopalkat, voidaan käyttää ryhmittelyä. HAVING-avainsanalla tulostaulusta voidaan rajata pois sellaiset rivit, jotka eivät täytä tiettyä ryhmittelyehtoa. Seuraava esimerkki havainnollistaa SQL-kyselyn rakenteen kaikkine osineen. Hakasuluissa esitetyt osat ovat valinnaisia, ja kyselyn osat kirjoitetaan seuraavassa järjestyksessä:

```
   SELECT <sarakkeet>
     FROM <taulut>
   [WHERE <ehdot>]
[GROUP BY <ryhmittelysarakkeet>]
  [HAVING <ryhmittelyehto>]
[ORDER BY <sarakkeet>];
```

Koostefunktioita (engl. *aggregate functions*) käytetään laskutoimitusten suorittamiseen. Niiden avulla voidaan laskea taulun rivien määrä (COUNT), sarakkeen arvojen summa (SUM), maksimi- ja minimiarvo (MAX, MIN) sekä keskiarvo (AVG). Koostefunktioita käytetään ainoastaan kyselyn SELECT- tai HAVING-osassa. AS-predikaatilla voidaan nimetä tulostaulun sarake halutulla tavalla, ja sen käyttäminen voi olla hyödyllistä etenkin koostefunktioita käytettäessä. Oletetaan esimerkiksi, että tietokantaan on tallennettu *Työntekijä*-tauluun tiedot työntekijöiden palkoista sarakkeeseen *Palkka*. Palkkojen keskiarvo haettaisiin seuraavasti:

```
SELECT AVG(Palkka) AS Palkkojen_keskiarvo
  FROM Työntekijä;
```

2.5 Tietokannanhallintajärjestelmä

Tietokannanhallintajärjestelmä (engl. *database management system, DBMS*) on ohjelmisto, jolla nimensä mukaan hallitaan tietokantaa. Tietokanta ja tietokannanhallintajärjestelmä noudattavat aina yhtä tai useampaa tietomallia, ja esimerkiksi relaatiomallia noudattavaa tietokannanhallintajärjestelmää kutsutaan relaatiotietokannanhallintajärjestelmäksi (engl. *relational database management system, RDBMS*). Relaatiotietokannanhallintajärjestelmä käsittelee käyttäjän tai sovellusohjelman lähettämiä SQL-komentoja, ja se palauttaa vastauksena komennosta riippuen esimerkiksi dataa tietokannasta sekä ilmoituksen komennon onnistumisesta. (Foster & Godbole 2016, s. 21; Darwen 2010, s. 22)

Fosterin & Godbolen (2016, s. 21) sekä Harringtonin (2009, s. 9–10) mukaan tietokannanhallintajärjestelmän tulee tarjota toiminnallisuudet muun muassa:

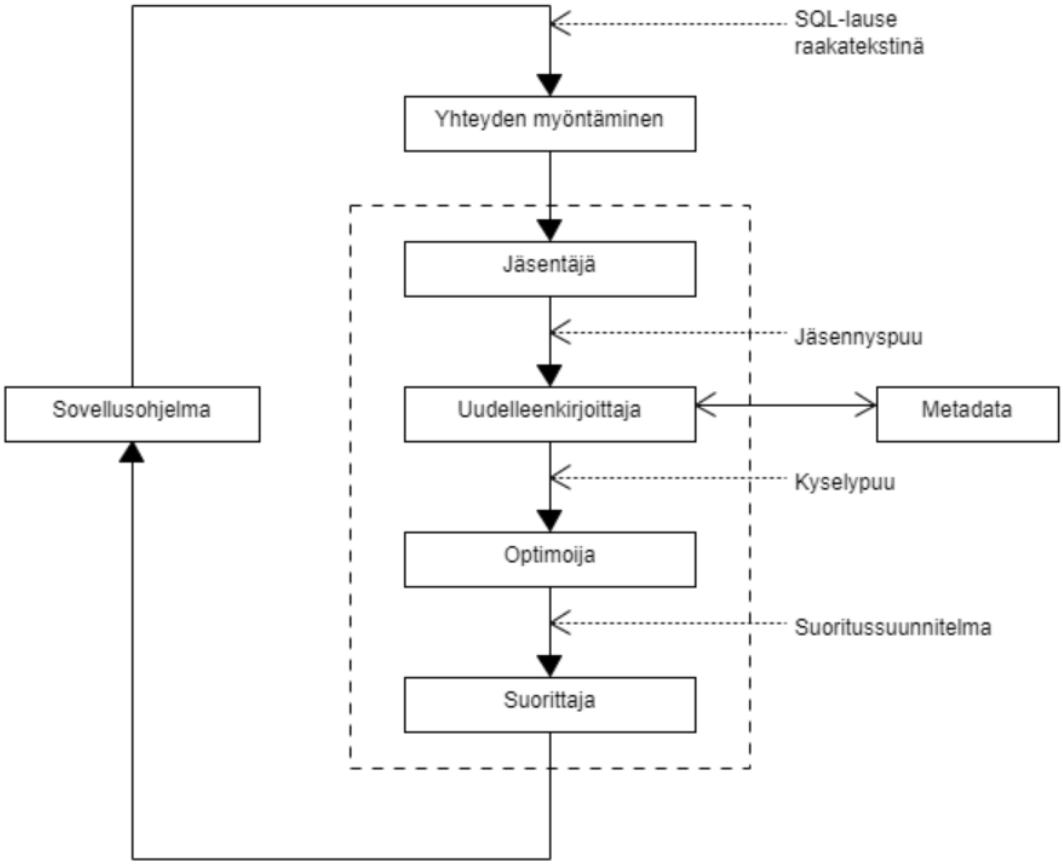
- tietokannan rakenteen määrittämiseen,
- datan hakemiseen, lisäämiseen, muokkaamiseen ja poistamiseen,
- datan turvallisuus- ja eheystarkistusten suorittamiseen,
- datan käytön hallintaan,
- tapahtumien hallintaan ja
- ohjelmointikielten tukemiseen.

PostgreSQL on avoimen lähdekoodin suosittu tietokannanhallintajärjestelmä, ja se noudattaa pääasiallisena tietomallinaan relaatiomallia. Se tukee kuitenkin myös muita tietomalleja. PostgreSQL:n suosio perustuu muun muassa luotettavuuteen ja laajennettavuuteen, ja se tukee valtaosaa SQL-standardissa määritellyistä ominaisuuksista. (PostgreSQL 2021a) Taulukossa 1 on esitetty DB-Enginesin (2021) listaamat kymmenen suosituinta DBMS:ää pääasiallisine tietomalleineen huhtikuussa 2021.

Sijoitus	DBMS	Tietomalli
1.	Oracle	Relaatiomalli
2.	MySQL	Relaatiomalli
3.	Microsoft SQL Server	Relaatiomalli
4.	PostgreSQL	Relaatiomalli
5.	MongoDB	Dokumenttimalli
6.	IBM DB2	Relaatiomalli
7.	Redis	Avain-arvoparimalli
8.	Elasticsearch	Hakukonemalli
9.	SQLite	Relaatiomalli
10.	Microsoft Access	Relaatiomalli

Taulukko 1. Suosituimmat tietokannanhallintajärjestelmät

Kuviossa 3 havainnollistetaan tiivistetysti PostgreSQL:n tapaa käsitellä sovellusohjelman lähettämiä SQL-lauseita. Ensimmäinen vaihe on yhteyden myöntäminen sovellusohjelman ja tietokantapalvelimen välille. Yhteyden myöntämisen jälkeen *jäsentäjä* (engl. *parser*) tarkistaa raakatekstinä vastaanottamansa SQL-lauseen syntaksin ja muodostaa siitä *jäsennyspuun* (engl. *parse tree*), mikäli lause on syntaksiltaan oikeellinen. Tietokannanhallintajärjestelmä palauttaa virheilmoituksen, jos SQL-lause sisältää syntaksivirheen. Syntaksivirheitä ja virheilmoituksia käsitellään tarkemmin seuraavassa pääluvussa. *Uudelleenkirjoittaja* (engl. *rewrite system*) tarkistaa tietokannan metadatasta tietokannan rakenteeseen mahdollisesti määritettyjä sääntöjä, joita soveltamalla se muodostaa *kyselypuun* (engl. *query tree*). *Optimoija* (engl. *optimizer* tai *planner*) laatii kyselypuun perusteella *suoritussuunnitelman*, jonka se siirtää *suorittajalle* (engl. *executor*). Optimoijan laatima suoritussuunnitelma perustuu tehokkaimpaan tapaan palauttaa sovellusohjelman tarvitsema tieto. Lopulta suorittaja palauttaa halutut tiedot sovellusohjelmalle. (PostgreSQL 2021b; PostgreSQL 2021c)



Kuvio 3. SQL-lauseiden käsittely PostgreSQL:ssä

3 Aiemmin tehty tutkimus

Luvussa esitellään ohjelmointikielten kääntäjien virheilmoituksiin liittyvää aiempaa tutkimusta sekä SQL-kyselyissä tapahtuvia virheitä ja virheiden mahdollisia syitä.

3.1 Yleistä virheilmoituksista

Ohjelmointikielten kääntäjien virheilmoituksia on tutkittu jo yli puolen vuosisadan ajan. Tästä huolimatta virheilmoitusten laadussa on edelleen paljon parannettavaa. Kryptiset virheilmoitukset ovat erityisen hankalia aloitteleville ohjelmoijille, ja ne vaikuttavat merkittävästi muun muassa ohjelmoijien itseluottamukseen, tyytyväisyyteen ja sitä kautta koko ohjelmistokehitysprojektiin. Virheilmoitukset toimivat rajapintana ihmisen ja kääntäjän välillä, joten niiden laatu vaikuttaa merkittävästi vuorovaikutuksen tehokkuuteen. (Becker ym. 2019; Traver 2010)

Ohjelmointikielen spesifikaatiossa määritetään syntaksi eli kielioppi sille, miten ohjelmia tulee kirjoittaa. Ohjelman lähdekoodin osa, joka ei noudata kielen syntaksia, aiheuttaa *syntaksivirheen* ohjelmaa käännettäessä. Esimerkiksi puutteellisesti kirjoitetut ohjelmointikielen avainsanat voivat aiheuttaa syntaksivirheen. Kääntäjä palauttaa soveltuvan *virheilmoituksen* virheen havaittuaan. (Becker ym. 2019)

Ohjelmistokehitysympäristöt (engl. *integrated development environments, IDEs*) voivat tuottaa virheilmoituksia eri tavoin. Perinteisiä tekstimuotoisia virheilmoituksia käytetään erityisesti komentorivityökalujen yhteydessä. Niiden lisäksi IDE voi tarjota esimerkiksi kuvakepohjaisia virheilmoituksia, tai virheellinen lähdekoodin osa voidaan osoittaa alleviivaamalla tai korostamalla se muulla tavalla. Tässä tutkimuksessa tutkitaan tekstimuotoisia SQL-virheilmoituksia, joten teoreettinen tarkastelu rajataan koskemaan ainoastaan tekstimuotoisia virheilmoituksia. (Becker ym. 2019)

Ohjelmointikielten kääntäjien virheilmoituksia on tutkittu paljon eri näkökulmista. Tutkimuskohteita ovat olleet muun muassa virheilmoitusten tehostaminen (engl. *enhancement*), tehostamiseen liittyvät tekniset haasteet sekä virheilmoitusten vaikutus ohjelmoinnin

opiskelussa. Virheilmoitusten tehostamisen vaikutuksia on tutkittu erityisesti aloittelevien ohjelmoijien näkökulmasta. Virheilmoitusten tehostamista käsitellään tarkemmin luvussa 3.3.

3.2 Virheilmoitusten vaikutukset HCI-näkökulmasta

Virheilmoitusten voidaan ajatella toimivan yhtenä rajapintana ihmisen ja tietokoneen välisessä vuorovaikutuksessa (engl. *human-computer interaction, HCI*). Virheilmoitusten suunnittelussa HCI-näkökulmaa ei ole kuitenkaan juuri huomioitu. Puutteelliset virheilmoitukset turhauttavat ja lannistavat ohjelmoinnin opiskelijoita, ja ne aiheuttavat merkittäviä esteitä oppimisessa. (Becker ym. 2019; Becker 2016) Ohjelmoinnin oppiminen on ylipäänsä haastavaa (mm. Luxton-Reilly ym. 2018), joten puutteelliset ja kryptiset virheilmoitukset eivät helpota jo ennestään haastavaa oppimisprosessia.

Ohjelmointikielen kääntäjän antama palaute eli virheilmoitus on yksi merkittävimmistä opiskelijan oppimiseen ja motivaatioon vaikuttavista tekijöistä. Virheilmoitukset aiheuttavat hankaluuksia erityisesti aloitteleville ohjelmoijille, mutta niiden heikko laatu vaikuttaa myös kokeneempiin ohjelmoijiin. Kokeneempikin ohjelmoija voidaan itse asiassa nähdä aloittelijana esimerkiksi tilanteessa, jossa tämä joutuu opiskelemaan uusia ohjelmointikieliä tai kääntäjien toimintaa. Heikkolaatuiset virheilmoitukset voivat ymmärrettävästi laskea opiskelijoiden itseluottamusta ja mielenkiintoa ohjelmointiin. (Becker ym. 2019; Watson ym. 2012)

Turhautuminen puutteellisiin virheilmoituksiin voi näkyä eri tavoin ohjelmoijan toiminnassa. Traverin (2010) mukaan ohjelmoija saattaa tehdä umpimähkäisesti korjauksia ohjelman lähdekoodiin päästäkseen yli virheilmoituksen aiheuttaneesta virheestä. Virheilmoitusta ei tällöin edes yritetä ymmärtää kunnolla. Toisaalta samaa jo virheelliseksi havaittua lähdekoodia voidaan yrittää kääntää uudelleen siinä toivossa, että kääntäjä antaisi eri tuloksen. Myös Pettit'n ym. (2017) mukaan turhautuminen voi näkyä saman kääntäjävirheen ilmenemisessä toistuvasti, kun virheilmoitusta ei ymmärretä kunnolla.

Ohjelmointikielet, niiden dokumentaatiot ja kääntäjien virheilmoitukset ovat useimmiten englanninkielisiä, mutta valtaosa ohjelmoinnin opiskelijoista ei puhu äidinkielenään englantia. Tämä aiheuttaa luonnollisesti lisähaasteita oppimiseen ja virheilmoitusten ymmärtämiseen. Virheilmoitukset ovat usein epätarkkoja ja epätäsmällisiä, mikä myös hankaloittaa niiden ymmärtämistä. Yhteyttä virheilmoituksen ja sen aiheuttaneen virheellisen ohjelmakoodin osan välillä voi olla hankalaa hahmottaa. Tietty virhe voi kontekstista riippuen tuottaa erilaisia virheilmoituksia, ja toisaalta tietyn virheilmoituksen voi tuottaa täysin toisistaan poikkeavat ja erilliset virheet. (Becker 2019; McCall & Kölling 2014)

3.3 Virheilmoitusten suunnittelu ja tehostaminen

Beckerin ym. (2019) mukaan tekstimuotoisen virheilmoituksen tehostamisella pyritään lisäämään sen hyödyllisyyttä virheen korjaamisen kannalta. Tehostaminen tapahtuu muokkaamalla virheilmoitusta eri tavoin. Virheilmoitusten tehostamisen vaikutuksia on selvitetty useissa tutkimuksissa, joiden tulokset ovat olleet osin ristiriitaisia keskenään.

Beckerin (2016) tutkimuksessa selvitettiin Java-ohjelmointikielen Javac-kääntäjän virheilmoitusten tehostamisen vaikutuksia ohjelmoinnin opiskelussa. Tutkimuksessa hyödynnettiin tätä tarkoitusta varten toteutettua Decaf-editoria, jossa opiskelijoille näytettiin sekä alkuperäinen että tehostettu virheilmoitus. Tulokset osoittivat, että virheilmoitusten tehostaminen vaikutti laskevasti tehtyjen virheiden kokonaismäärään, toistuvien virheiden määrään ja opiskelijakohtaisten virheiden määrään.

Pettit'n ym. (2017) tutkimuksessa selvitettiin C++-ohjelmointikielen kääntäjän virheilmoitusten tehostamisen vaikutuksia opiskelijoiden tekemiin virheisiin. Tehostamisesta huolimatta toistuvien virheiden tekemisen todennäköisyyden ei havaittu laskevan. Denny ym. (2014) eivät myöskään havainneet opiskelijoiden hyötyneen merkittävästi virheilmoitusten tehostamisesta. Tutkimusten yksityiskohtia tarkastelemalla voidaan kuitenkin havaita eroavaisuuksia, jotka mahdollisesti selittävät tulosten ristiriitaisuutta. Tuloksiin vaikuttavia tekijöitä voivat olla esimerkiksi tehostettujen virheilmoitusten määrä ja esitystapa sekä erot tutkimukseen osallistuvien henkilöiden tietotaidoissa.

Virheilmoitusten suunnittelun ja tehostamisen ohjenuoria on esitetty useissa tutkimuksissa. Seuraavaksi esiteltävät Beckerin ym. (2019) laatimat yleistetyt ohjenuorat perustuvat näihin eri tutkimuksissa esitettyihin ohjenuoriin. Lihavoinnilla on korostettu ohjenuoraan olennaisesti sisältyvä virheilmoituksen käytettävyyteen liittyvä ominaisuus.

Virheilmoituksen laatuun vaikuttaa olennaisesti sen **luettavuus**. Shneidermanin (1982) mukaan virheilmoituksen tulisi olla lyhyt, kattava ja täsmällinen. Se ei saisi sisältää kryptisiä eikä epämääräisiä ilmauksia, jotka eivät auta käyttäjää virheen korjaamisessa. McIver & Conway (1996) painottavat, että teknisen jargonin sijaan virheilmoituksissa tulisi käyttää enemmän luonnollista kieltä. Tämä auttaa etenkin aloittelevia ohjelmoijia virheilmoituksen ymmärtämisessä ja virheen korjaamisessa. Traverin (2010) mukaan virheilmoituksen tulee olla selkeä ja ytimekäs, koska liian pitkää virheilmoitusta ei välttämättä lueta kokonaan tai se tulkitaan väärin. Luettavuuden arviointiin ei ole toistaiseksi esitetty konkreettisia ohjeita, mutta luettavampien virheilmoitusten on todettu muun muassa vähentävän virheiden määrää ja ohjelmoijien turhautumista.

Virheilmoituksen sisältämän **kognitiivisen kuorman** vähentäminen on tärkeää, koska ihmisen kyky käsitellä informaatiota tehokkaasti on rajallinen (Sweller 1988). Tämä perustuu pitkälti työmuistin rajallisuuteen. Virheilmoituksen ytimekkyys on siis tärkeää tässäkin mielessä. Tutkimuksissa on todettu, että kognitiivisen kuorman määrä on epäsuorasti yhteydessä tehtyjen virheiden määrään (Ayres & Sweller 1990; Ayres 2001). Hundhausen ym. (2017) esittävät kognitiivisen kuorman vähentämiseen kolme keinoa, jotka ovat seuraavat:

- Virheilmoituksen tulee sisältää riittävästi informaatiota virheen ymmärtämiseksi, ja virheen sijainti ohjelmakoodissa tulee osoittaa virheilmoituksen välittömässä läheisyydessä.
- Toisteisuutta tulee vähentää eli jokainen virhetyyppi esitetään vain kertaalleen.
- Virheilmoitus voidaan esittää sekä visuaalisesti että auditiivisesti. Esimerkiksi virheen kuvaus voidaan antaa auditiivisesti samalla kun se osoitetaan visuaalisesti näytöllä.

Laadukkaassa virheilmoituksessa virheelle tarjotaan **konteksti**, mikä voi tarkoittaa esimerkiksi virheen sijainnin osoittamista ohjelmakoodissa. Konteksti voi sisältää myös tietoa virheen syistä, mikä helpottaa virheen ymmärtämistä ja korjaamista. Esimerkiksi puutteellisesti kirjoitettujen ohjelmointikielen avainsanojen osoittaminen selkeästi voi olla olennaista virheen korjaamisen kannalta (Becker ym. 2016). Virheen sijainnin määrittäminen tarkasti esimerkiksi näkyvän osoittimen avulla auttaa luonnollisesti sen korjaamisessa, mutta väärin paikannettuna se voi lisätä ohjelmoijan turhautumista (Traver 2010; Horning 1976). Virheilmoituksen aiheuttanut virhe tulee kuvata täsmällisesti (Shneiderman 1982). Epämääräisten ilmausten ja numeeristen virhekoodien esittämistä tulee siis välttää. Virheilmoituksessa voidaan hyödyntää esimerkiksi ohjelmakoodissa esiintyviä muuttujien nimiä, jotka ovat ohjelmoijalle tuttuja.

Positiivisessa ja **rakentavassa** sävyssä esitetty virheilmoitus auttaa ohjelmoijaa virheen korjaamisessa. Positiivisia virheilmoituksia on kuvailtu muun muassa hillityiksi, kohteliaiksi, ystävällisiksi ja rohkaiseviksi. Shneiderman (1982) huomauttaa, että sävyltään negatiiviset ja ohjelmoijia virheistä kritisoivat virheilmoitukset hämmentävät, järkyttävät ja lannistavat etenkin aloittelevia ohjelmoijia. Negatiivisten ilmausten, kuten *ILLEGAL, INVALID, INCORRECT* ja *ERROR*, käyttöä tulee siten välttää. Negatiivisten ja epämääräisten ilmausten sijaan virheilmoituksen tulisi sisältää riittävästi informaatiota virheen korjaamiseksi. Shneiderman (1982) antaa esimerkin, jossa virheilmoitus *INVALID PASSWORD* voitaisiin esittää positiivisemmin ja rakentavammin esimerkiksi muodossa *Your password did not match the stored password. Please try again*.

Huumorin ja antropomorfismin eli inhimillisten piirteiden lisäämisen vaikutuksia virheilmoitusten hyödyllisyyteen on tutkittu jonkin verran. Huumorin avulla virheilmoituksista saadaan hauskempia, mutta toisaalta huumorin vaikutus voi hiipua ajan kuluessa. Todennäköisesti huumorin lisääminen myös pidentää ja monimutkaistaa virheilmoitusta, ja väärin tulkittuna huumori jopa heikentää virheilmoituksen laatua. Virheilmoituksen elollistaminen antropomorfismin keinoin voi olla hyödyksi etenkin aloitteleville ohjelmoijille, koska kääntäjien standardivirheilmoitukset ovat monesti tylyjä ja tuomitsevia. Toisaalta liiallinen inhimillisyys voi saada ohjelmoijan virheellisesti ajattelemaan, että tietokone osaisi aistia ja ajatella.

Useissa tutkimuksissa **esimerkkien** näyttämisellä on todettu olevan positiivisia vaikutuksia virheilmoitusten hyödyllisyyteen. Virheestä riippuen virheilmoituksessa voidaan esimerkein opastaa, miten ohjelmointikielen ominaisuuksia tulee käyttää. Esimerkit voivat liittyä muun muassa tietotyyppien ja muuttujien määrittelyyn. Niiden näyttäminen pidentää virheilmoitusta ja samalla lisää kognitiivista kuormaa, mutta toisaalta ne myös tehostavat opiskelijoiden oppimista ja vähentävät toistuvien virheiden määrää. Haasteena voi olla muun muassa se, että ohjelmoija ei osaa yhdistää geneeristä esimerkkiä omassa ohjelmakoodissaan esiintyvään virheeseen. Toisaalta ohjelmoija saattaa etsiä esimerkkikoodin virhettä omasta ohjelmakoodistaan, mikä lisää hämmennystä.

Virheen raportoinnin lisäksi sen korjaamista helpottaa **ratkaisujen** tai **vihjeiden** esittäminen virheilmoituksessa. Erot esimerkkien ja ratkaisujen tai vihjeiden välillä voivat olla häilyviä. Ratkaisut ovat oikeastaan vain ehdotuksia, koska kääntäjän on mahdotonta tietää ohjelmoijan todellisia aikeita. Kantorowitz & Laor (1986) painottavatkin, että virheen ratkaisutapa tulee esittää virheilmoituksessa vain silloin, kun ratkaisu on suurella todennäköisyydellä oikeellinen. Jos virheen korjaamiseen on olemassa useampia ratkaisutapoja, tulee kaikki ratkaisutavat esittää virheilmoituksessa. Tällöin ohjelmoija voi saada myös vinkkejä ohjelmointikielen tarjoamista ominaisuuksista ja mahdollisuuksista.

Vuorovaikutuksellisuuden lisääminen on yksi keino virheilmoitusten tehostamisessa. Traver (2010) esittää virheilmoituksen jakamista kolmeen tasoon, jotka jokainen osaltaan auttavat virheen korjaamisessa. Ensimmäisenä ohjelmoijalle näytetään lyhyt virheilmoitus, joka auttanee useimmissa tapauksissa riittävästi virheen korjaamisessa. Seuraavalla tasolla annetaan hieman pidempi kuvaus virheestä ja mahdollisesti joitain esimerkkejä. Kolmannella tasolla voidaan esittää lista mahdollisista korjaustoimista, mikäli edellisten tasojen virheilmoitukset eivät ole olleet riittäviä. Näin ollen ohjelmoijalla on mahdollisuus saada lisäapua virheen korjaamiseen oman osaamistasonsa mukaan. McIver & Conway (1996) kutsuvat tällaista lisäavun asteittaista tarjoamista *tell-me-more* -toiminnallisuudeksi. Barik ym. (2018) esittävät, että lisäapua tarjottaisiin virheilmoitukseen sisällytettävän WWW-linkin kautta. Toinen vaihtoehto voisi olla esimerkiksi jonkin erillisen komennon (esim. `explain`) lähettäminen kääntäjälle, jolloin kääntäjä palauttaa standardivirheilmoitusta laajemman kuvauksen virheestä.

Ohjelmoinnin oppimista voidaan tehostaa tarjoamalla **kognitiivista tukea** (engl. *cognitive scaffolding*), jonka tarkoituksena on tiivistetysti tarjota opiskelijoille tukea tietotaitojen kehittämiseen. Tuen sisällyttäminen virheilmoituksiin tehostaa oppimista etenkin alkuvaiheessa, koska tällöin opiskelijat saavat parempaa palautetta tekemistään virheistä verrattuna kääntäjien standardivirheilmoituksiin. Flowersin ym. (2004) tutkimuksessa selvitettiin Gauntlet-työkalulla tehostettujen Java-kääntäjän virheilmoitusten vaikutuksia opiskelijoiden kokemuksiin ja opettajien työmäärään. Gauntlet tulkitsi yleisimpien ohjelmointivirheiden aiheuttamia virheilmoituksia luonnollisella kielellä helposti ymmärrettävässä muodossa. Gauntletin ansiosta opiskelijat kykenivät paremmin itsenäiseen ongelmanratkaisuun, ja opettajien työmäärä laski opiskelijoiden avun tarpeen vähentyessä. Opiskelijoiden tuottaman ohjelmakoodin laadun havaittiin myös parantuneen. Coull & Duncan (2011) ehdottavat vahvan tuen tarjoamista opiskelijoille oppimisprosessin alkuvaiheessa, ja tukea vähennetään asteittain opiskelijoiden tietotaitojen kehittyessä. Näin ollen myös lyhyemmät kääntäjien standardivirheilmoitukset voivat myöhemmin auttaa riittävästi virheiden korjaamisessa.

Viimeaikaisissa tutkimuksissa on tuotu esiin **loogisen argumentoinnin** merkitys osana laadukkaita virheilmoituksia. Barik ym. (2018) havaitsivat, että argumentoinnin rakenteeseen ja sisältöön panostamalla virheitä voidaan selittää huomattavasti tehokkaammin ja ihmisläheisesti. He sovelsivat tutkimuksessaan Toulminin argumentointimallia virheilmoitusten laadun arvioinnissa. Kyseisen mallin mukaan argumentti koostuu kolmesta perustavanlaatuisesta osasta, jotka ovat *väite* (engl. *claim*), *perusteet* (engl. *grounds*) ja *oikeutus* (engl. *warrant*). Virheilmoituksessa väite kuvaa virheen syyn, ja perusteet tukevat väitettä selittämällä tarkemmin mistä virhe johtuu. Oikeutuksella luodaan silta väitteen ja perusteiden välille esimerkiksi seuraavasti: *[väite] koska [perusteet]*. Rakenteeltaan oikea argumentti sisältää kaikki kolme edellä mainittua osaa, kun taas puutteellisesta argumentista joku osista puuttuu. Argumenttiin voi sisältyä myös muita osia edellä mainittujen lisäksi, jolloin puhutaan laajennetusta argumentista.

Viimeisenä ohjenuoranaan Becker ym. (2019) mainitsevat virheilmoitusten esittämisen **oikea-aikaisuuden**. Viime vuosina tehtyjen tutkimusten mukaan virheilmoitus tulisi esittää ohjelmoijalle niin pian kuin mahdollista staattisten analyysityökalujen avulla. Staattista analyysiä hyödynnetään paljon nykyaikaisissa ohjelmistokehitysympäristöissä. Sen avulla

virheellinen ohjelmakoodin osa voidaan osoittaa koodieditorissa esimerkiksi punaisella aaltoviivalla.

Beckerin ym. (2019) mukaan kääntäjällä on kaksi laajaa tehtävää, jotka sen tulee suorittaa tuottaakseen tehokkaita ja asianmukaisia virheilmoituksia. Ne ovat virheen havaitseminen ja tietojen kerääminen ohjelman käännösaikaisesta tilasta. Kirjallisuudessa on esitetty useita yleisiä ongelmia, jotka liittyvät tehokkaiden virheilmoitusten tuottamiseen. Ne ovat tiivistetysti seuraavat:

- **Täydellisyysongelma** (engl. *completeness problem*): Ohjelman virheellistä käyttäytymistä on käytännössä mahdotonta havaita kaikissa mahdollisissa tilanteissa ilman, että tehokkuus laskee merkittävästi. Kääntäjien kehityksessä keskitytäänkin tyypillisesti vain tietynlaisten virheiden havaitsemiseen.
- **Paikantamisongelma** (engl. *locality problem*): Virhe havaitaan usein kaukana todellisesta sijainnistaan, jolloin virheilmoitus ei viittaa täsmällisesti todelliseen virheeseen.
- **Kartoitusongelma** (engl. *mapping problem*): Jotkin ohjelmointikielet tukevat koodin transformointia, jossa ohjelmoijan kirjoittama ohjelmakoodi muutetaan toiseen muotoon ennen ohjelman suorittamista. Jos virhe havaitaan transformoidussa koodissa, virheilmoitus viittaa alkuperäisen koodin sijaan transformoituun koodiin.
- **Suunnitteluongelma** (engl. *engineering problem*): Tehokkaiden virheilmoitusten suunnittelu vaatii paljon panostuksia kääntäjien kehittäjiltä. Kehitystyössä panostetaan enemmän uusien ominaisuuksien lisäämiseen kuin virheilmoitusten laadun parantamiseen. Tehokkaampien virheilmoitusten kehittäminen voi laskea kääntäjien suorituskykyä ja lähdekoodin ylläpidettävyyttä.
- **Elävyysongelma** (engl. *liveness problem*): Nykyaikaisten ohjelmistokehitysympäristöjen analyysityökalut analysoivat lähdekoodia koko ajan sen ollessa muokattavana. Ohjelma ei ole täydellinen ollessaan muokattavana, mikä voi aiheuttaa haasteita sopivien virheilmoitusten tuottamiseen.

3.4 SQL-kyselyissä tapahtuvat virheet

Taipalus ym. (2018) esittävät SQL-standardiin perustuvan tietokannanhallintajärjestelmistä riippumattoman virheluokittelun opiskelijoiden SQL-kyselyissä (SELECT-lauseet) tekemistä virheistä. Virheluokittelu perustuu aiemmin tehtyihin tutkimuksiin (mm. Ahadi ym. 2016; Smelcer 1995; Welty 1985). Tietokannanhallintajärjestelmät poikkeavat toteutuksiltaan toisistaan, joten Taipaluksen ym. (2018) virheluokittelua soveltamalla eri tutkimusten tulokset ovat keskenään vertailukelpoisia. Taipaluksen ym. (2018) jaottelun mukaan virheluokat ovat syntaksivirheet, semanttiset virheet, loogiset virheet ja komplikaatiot. Tässä luvussa esitellään laajemmin vain tutkimuksen aihepiiriin sisältyviä syntaksivirheitä. Muut virheluokat esitellään tiivistetymmin.

Semanttisen virheen sisältävä SQL-kysely on syntaksiltaan oikeellinen, mutta kysely ei aina palauta haluttua tulostaulua. Esimerkiksi epäjohdonmukaisesta ehdosta kyselyn WHERE-osassa johtuen tulostaulu voi olla aina tyhjä, mikä osoittaa kyselyn virheellisyyden. Semanttinen virhe voidaan havaita tietämättä *tietotarvetta* (engl. *data demand*). Tietotarve kuvaa luonnollisella kielellä tiedot, jotka tietokannasta halutaan hakea. **Loogisen virheen** havaitsemiseksi tietotarpeen on puolestaan oltava tiedossa. Loogisen virheen sisältävä SQL-kysely on siis syntaksiltaan oikeellinen, mutta kyselyn palauttama tulostaulu ei vastaa tietotarvetta. (Brass & Goldberg 2005; Taipalus ym. 2018)

Komplikaatiolla (engl. *complication*) tarkoitetaan SQL-kyselyn tarpeetonta monimutkaisuutta. Kyseessä ei periaatteessa ole virhe, koska komplikaatiosta huolimatta tulostaulu on oikeellinen. Kyselystä saadaan kuitenkin tehokkaampi ja helpommin luettava poistamalla tarpeeton monimutkaisuus. Komplikaatiot ilmenevät tyypillisesti esimerkiksi tarpeettomina taulujen välisinä liitoksina tai taulujen esittelyinä, ja ne voidaan havaita tietämättä tietotarvetta. (Brass & Goldberg 2005; Taipalus ym. 2018)

Brassin & Goldbergin (2005) mukaan **syntaksivirheen** sisältävä SQL-kysely ei noudata SQL:n syntaksia, jolloin tietokannanhallintajärjestelmä ei voi suorittaa kyselyä. Tulostaulun sijaan tietokannanhallintajärjestelmä palauttaa tällöin virheilmoituksen. Taipalus ym. (2018) huomauttavat, että tietokannanhallintajärjestelmä voi korjata syntaksivirheen sisältäviä kyselyitä oman sietokykynsä mukaan. Näin ollen käytetystä tietokannanhallintajärjestelmästä

riippuen virheellinen kysely voi palauttaa joko tulostaulun tai virheilmoituksen. Taipalus ym. (2018) jakavat syntaksivirheet kuuteen kategoriaan seuraavasti:

- **Monitulkintainen tietokantaobjekti (SYN-1)**: Tietokantaobjektit (esim. taulut) muodostavat omat nimiavaruutensa, ja esimerkiksi eri tauluissa voi olla samannimisiä sarakkeita. Useampaan kuin yhteen tauluun kohdistuvissa kyselyissä tietokannanhallintajärjestelmä ei välttämättä osaa ratkaista, minkä taulun sarakkeeseen kyselyssä viitataan, jos taulut sisältävät samannimisiä sarakkeita eikä erillisiä tarkentimia ole käytetty.
- **Määrittelemätön tietokantaobjekti (SYN-2)**: Viittaukset määrittelemättömiin tietokantaobjekteihin aiheuttavat syntaksivirheen. Tällainen virhe voi johtua esimerkiksi kirjoitusvirheestä. Kyselyssä voidaan viitata esimerkiksi tauluun *työntekijät*, vaikka taulun oikea nimi on *työntekijä*.
- **Tietotyyppien ristiriita (SYN-3)**: Virhe voi ilmetä esimerkiksi siten, että tietyn sarakkeen arvoa tutkitaan sopimattomalla operaattorilla (esim. vertailuoperaattorin sijaan käytetty LIKE-operaattoria). Tyypillinen esimerkki on myös lainausmerkkien puuttuminen merkkijonosta sitä tutkittaessa.
- **Koostefunktion virheellinen sijoittaminen (SYN-4)**: Koostefunktioiden käyttäminen on sallittu vain lauseen SELECT- tai HAVING-osissa. Koostefunktion käyttäminen toisen koostefunktion parametrina aiheuttaa myös syntaksivirheen.
- **Virheellinen tai riittämätön ryhmittely (SYN-5)**: Vähintään yhden koostefunktion ja ryhmittelysarakkeen sisältävä pääkyselyn SELECT-osa edellyttää lauseelta myös GROUP BY -osaa. Ryhmittelysarakkeet tulee esittää sekä SELECT- että GROUP BY -osissa. Ylimääräiset tai puuttuvat ryhmittelysarakkeet sekä HAVING-osan esittely ilman GROUP BY -osaa aiheuttavat useimmissa tietokannanhallintajärjestelmissä syntaksivirheen.
- **Yleiset syntaksivirheet (SYN-6)**: Syntaksivirheet voivat aiheutua esimerkiksi puuttuvista puolipisteistä, ei-standardeista avainsanoista, lauseen dublikaattiosista (esim. WHERE-osa kahdesti) tai liian monesta sarakkeesta alikyselyn SELECT-osassa.

3.5 Virheiden mahdolliset syyt

Smelcerin (1995) mukaan virheet SQL-kyselyissä voivat johtua pohjimmiltaan neljästä syystä. Ihmisen työmuistin kapasiteetti on rajallinen, mikä näkyy etenkin käyttäjän unohduksena lisätä tiettyjä tulostaulun oikeellisuuden kannalta keskeisiä osia kyselyyn. Tämä näkyy kyselyssä esimerkiksi taulujen välisen tarpeellisen liitoksen puuttumisena. Tietotarpeen monimutkaistuminen esimerkiksi liitosten määrän lisääntymisen myötä voi aiheuttaa virheitä kyselyn muotoilussa, koska kaikki oleellinen tieto ei mahdu kerralla ihmisen työmuistiin. Toinen virheitä mahdollisesti aiheuttava syy on kyselyn muotoilua helpottavien eksplisiittisten vihjeiden puuttuminen tietotarpeesta. Kysely voi vaatia onnistuakseen esimerkiksi taulujen välisiä liitoksia, mutta suorasanaisia vihjeitä liitosten tekemiselle ei tietotarpeessa mainita. Kolmas mahdollinen syy on käyttäjien tottumus käyttää kyselyn muotoilussa tekniikoita, jotka eivät sovellu kyseiseen tilanteeseen. Esimerkiksi yhteen tauluun kohdistuvat kyselyt eroavat muotoilultaan useampaan tauluun kohdistuvista kyselyistä. Käyttäjien puutteelliset tietotaidot voivat myös olla syynä virheiden tekemiseen. Esimerkiksi ymmärryksen puute SQL-kielen toiminnasta tai relaatiomallista kasvattaa virheiden tekemisen todennäköisyyttä.

Taipalus (2020) esittää tutkimuksessaan näkemyksiä SQL-kyselyissä tapahtuvien yleisimpien pysyvien virheiden syistä Smelcerin (1995) esittämiin syihin perustuen. Pysyvällä virheellä tarkoitetaan virhettä, jota käyttäjä ei onnistu korjaamaan lopulliseen kyselyyn. Taipalus & Perälä (2019) havaitsivat, että loogiset virheet ja komplikaatiot ovat syntaksi- ja semanttisia virheitä pysyvämpiä. Taipaluksen (2020) näkemysten mukaan virheen syy riippuu virhetyypin lisäksi myös kyselyn luonteesta. Kyselyn luonteeseen vaikuttaa muun muassa se, kohdistuuko kysely yhteen vai useampaan tauluun ja käytetäänkö kyselyssä koostefunktioita. Puuttuvat ilmaukset (engl. *missing expression*), ylimääräiset tai puuttuvat ryhmittelysarakkeet (engl. *extraneous or omitted grouping column*) sekä puuttuvat liitokset (engl. *missing join*) lukeutuvat yleisimpien pysyvien virheiden joukkoon riippumatta kyselyn luonteesta. Taipaluksen (2020) mukaan esimerkiksi ilmauksen puuttuminen voi johtua useampaa taulua koskevissa kyselyissä työmuistin ylikuormittumisesta. Koostefunktioita sisältävässä kyselyssä ilmauksen puuttuminen voi puolestaan johtua eksplisiittisen vihjeen puuttumisesta tietotarpeessa.

4 Tutkimusasetelma

Luvussa kuvataan tutkimusasetelma ja -kysymykset sekä käytetty tutkimusmenetelmä.

4.1 Tutkimuksen tausta

Tutkimuksessa selvitettiin PostgreSQL-tietokannanhallintajärjestelmän (versio 12.1) virheilmoitusten käytettävyyteen liittyvien ominaisuuksien vaikutusta virheilmoitusten koettuun hyödyllisyyteen. PostgreSQL on maksuton ja suosittu relaatiotietokannanhallintajärjestelmä, joka tukee relaatiomallin lisäksi myös muita tietomalleja (DB-Engines 2021). Tutkittaviksi virheilmoituksiksi valittiin Taipaluksen ym. (2018) virheluokittelun 16 yleisimmän syntaksivirheen aiheuttamat virheilmoitukset. Taulukossa 2 on esitetty kuhunkin tehtävään sisältynyt syntaksivirheen luokka, ID ja virheen kuvaus edellä mainitun virheluokittelun mukaan.

Tehtävä	Luokka	ID	Virhe
1	SYN-1	2	Monitulkintainen sarake
2	SYN-2	11	Puuttuvat lainausmerkit merkkijonossa
3	SYN-6	35	IS-predikaatin käyttö väärässä yhteydessä
4	SYN-6	32	Virhe avainsanan käytön syntaksissa
5	SYN-6	31	Virhe avainsanan käytön logiikassa
6	SYN-6	26	Liian monta saraketta alikyselyssä
7	SYN-2	4	Saraketta ei ole olemassa
8	SYN-2	9	Kirjoitusvirheet
9	SYN-3	12	Sarakkeen nimen toistamatta jättäminen
10	SYN-4	14	Koostefunktion käyttäminen muualla kuin SELECT- tai HAVING-osassa
11	SYN-5	16	Ylimääräinen tai puuttuva ryhmittelysarake
12	SYN-6	37	Ei-standardit operaattorit
13	SYN-6	19	WHERE-avainsanan käyttö kahdesti
14	SYN-6	36	Ei-standardit avainsanat tai standardien avainsanojen käyttö väärässä yhteydessä
15	SYN-2	10	Synonyymit
16	SYN-6	34	Aalto-, haka- tai puuttuvat sulkeet

Taulukko 2. Tutkitut virheilmoitukset aiheuttaneet syntaksivirheet

Tutkimus toteutettiin verkkopohjaisena kyselytutkimuksena maaliskuussa 2021. Kyselylomakkeella (liite A) vastaajille esitettiin tietokannan skeema, tietotarve, syntaksivirheen sisältävä SQL-kysely ja tietokannanhallintajärjestelmän palauttama virheilmoitus. Vastaajat kirjoittivat korjatun SQL-kyselyn erilliseen tekstikenttään. Tämän jälkeen vastaajat vastasivat kolmeen Likert-asteikolliseen väittämään koskien virheilmoituksen hyödyllisyyttä virheen havaitsemisen, korjaamisen ja virheestä toipumisen näkökulmista. Vastaajat olivat Jyväskylän yliopiston informaatioteknologian tiedekunnan *Tietokannat ja tiedonhallinta* -kurssin opiskelijoita. Ennen kyselyyn vastaamista vastaajat olivat osallistuneet kyseisen kurssin opetukseen. Kurssilla käsiteltyihin asiasisältöihin kuuluivat tietokantoihin liittyvät yleiset käsitteet, käsitteellinen mallintaminen, relaatiomalli, relaatioalgebra- ja kalkyyli, transformointi, SQL-kieli, normalisointi sekä tietovarastointi. Vastaamalla kyselyyn vastaajilla oli mahdollisuus ansaita hyvityspiste kurssin tenttiin, mutta tutkimukseen osallistuminen oli vapaaehtoista. Kyselyyn vastasi 217 opiskelijaa, joista 190 osallistui tutkimukseen. Tutkimukseen osallistuneille vastaajille esitettiin ennen kyselyyn vastaamista tietosuojailmoitus, jossa tiedotettiin muun muassa tutkimuksen tarkoituksesta, henkilötietojen suojaamisesta ja tutkimusaineiston anonymisoinnista.

Tutkimusaineiston anonymisoinnin jälkeen virheilmoitukset koodattiin Shneidermanin (1982) heuristiikkojen mukaan siten, että kuhunkin virheilmoitukseen liitettiin tiedot sen sisältämistä käytettävyyteen liittyvistä ominaisuuksista. Ominaisuudet olivat *lyhyys, positiivisuus, rakentavuus, täsmällisyys, paikannettavuus* ja *ymmärrettävyys*. Shneiderman (1982) ei ole eksplisiittisesti maininnut paikannettavuutta ominaisuutena, mutta esimerkiksi Traver (2010) nostaa sen yhdeksi kriteeriksi tehokkaille virheilmoituksille.

4.2 Tutkimuskysymykset

Tutkimuksessa selvitettiin virheilmoitusten käytettävyyteen liittyvien ominaisuuksien vaikutusta virheilmoitusten koettuun hyödyllisyyteen kolmesta eri näkökulmasta. Tutkimuskysymykset olivat seuraavat:

- TK1: Mitkä tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoituksen koettuun hyödyllisyyteen virheen havaitsemisen näkökulmasta?

- TK2: Mitkä tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoituksen koettuun hyödyllisyyteen virheen korjaamisen näkökulmasta?
- TK3: Mitkä tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoituksen koettuun hyödyllisyyteen virhetilanteesta toipumisen näkökulmasta?

4.3 Tutkimusmenetelmä

Kaikkien tutkimuskysymysten koestamiseen tutkimusmenetelmäksi valittiin järjestysasteikollinen logistinen regressioanalyysi (engl. *ordinal logistic regression*), koska riippuva muuttuja oli kussakin tutkimuskysymyksessä järjestysasteikollinen, ja riippumattomat muuttujat olivat dikotomisia. Riippumattomien muuttujien luonteesta johtuen multikollineaarisuutta ei tarkasteltu. Kaikki virheilmoitukset koodattiin aineiston koodaamisvaiheessa lyhyiksi ja ymmärrettäviksi, joten nämä ominaisuudet jätettiin regressiomallin ulkopuolelle.

5 Tulokset

Luvussa esitetään tutkimustulokset yhteenvetona ja tutkimuskysymyksittäin eriteltyinä.

5.1 Yhteenveto

Taulukossa 3 on esitetty yhteenveto tutkimustuloksista. Plusmerkki (+) taulukon solussa osoittaa, että virheilmoituksen käytettävyyteen liittyvällä ominaisuudella on positiivinen vaikutus virheilmoituksen koettuun hyödyllisyyteen. Miinusmerkki (-) puolestaan osoittaa vaikutuksen olevan negatiivinen. Tyhjä solu osoittaa, ettei ominaisuudella ole vaikutusta virheilmoituksen koettuun hyödyllisyyteen. Esimerkiksi rakentavuus vaikuttaa virheilmoituksen koettuun hyödyllisyyteen ainoastaan virhetilanteesta toipumisen näkökulmasta (TK3), ja vaikutus on negatiivinen. Rakentava virheilmoitus siis laskee virheilmoituksen koettua hyödyllisyyttä virhetilanteesta toipumisen näkökulmasta verrattuna ei-rakentavaan virheilmoitukseen. Positiivisuus ja paikannettavuus vaikuttavat positiivisesti eli nostavasti virheilmoitusten koettuun hyödyllisyyteen kaikkien tutkimuskysymysten osalta.

Ominaisuus	**TK1**	**TK2**	**TK3**
Positiivisuus	+	+	+
Rakentavuus			-
Täsmällisyys	+	+	
Paikannettavuus	+	+	+

Taulukko 3. Yhteenveto tuloksista

5.2 Ominaisuuksien vaikutukset virheen havaitsemiseen

Järjestysasteikollista logistista regressioanalyysiä käytettiin selvittämään, vaikuttavatko tietokannanhallintajärjestelmän virheilmoituksen ominaisuudet (positiivisuus, rakentavuus, täsmällisyys ja paikannettavuus) virheilmoituksen koettuun hyödyllisyyteen virheen havaitsemisen näkökulmasta. Todennäköisyys sille, että positiiviset virheilmoitukset koettiin hyödyllisiksi virheen havaitsemisen kannalta, oli 4.890-kertainen (95 % luottamusväli, 3.268...7.317) ei-positiivisiin virheilmoituksiin verrattuna, $\chi^2(1) = 59.579$, $p < .001$.

Todennäköisyys sille, että rakentavat virheilmoitukset koettiin hyödyllisiksi virheen havaitsemisen kannalta, oli 0.900-kertainen (95 % luottamusväli, 0.755…1.074) ei rakentaviin virheilmoituksiin verrattuna $\chi^2(1) = 1.365$, $p = .243$. Todennäköisyys sille, että täsmälliset virheilmoitukset koettiin hyödyllisiksi virheen havaitsemisen kannalta, oli 1.361-kertainen (95 % luottamusväli, 1.181…1.568) ei-täsmällisiin virheilmoituksiin verrattuna, $\chi^2(1) = 18.079$, $p < .001$. Todennäköisyys sille, että paikannetut virheilmoitukset koettiin hyödyllisiksi virheen havaitsemisen kannalta, oli 2.366-kertainen (95 % luottamusväli, 1.990…2.813) ei-paikannettuihin virheilmoituksiin verrattuna, $\chi^2(1) = 95.057$, $p < .001$. Taulukossa 4 on esitetty virheilmoitusten käytettävyyteen liittyvien ominaisuuksien regressiokertoimet ja tilastolliset merkitsevyydet virheen havaitsemisen näkökulmasta.

Ominaisuus	**Regressiokerroin**	**Merkitsevyys**
Positiivisuus	4.890	$p < .001$
Rakentavuus	0.900	$p = .243$
Täsmällisyys	1.361	$p < .001$
Paikannettavuus	2.366	$p < .001$

Taulukko 4. Ominaisuuksien vaikutukset virheen havaitsemiseen

5.3 Ominaisuuksien vaikutukset virheen korjaamiseen

Järjestysasteikollista logistista regressioanalyysiä käytettiin selvittämään, vaikuttavatko tietokannanhallintajärjestelmän virheilmoituksen ominaisuudet (positiivisuus, rakentavuus, täsmällisyys ja paikannettavuus) virheilmoituksen koettuun hyödyllisyyteen virheen korjaamisen näkökulmasta. Todennäköisyys sille, että positiiviset virheilmoitukset koettiin hyödyllisiksi virheen korjaamisen kannalta, oli 5.322-kertainen (95 % luottamusväli, 3.702…7.650) ei-positiivisiin virheilmoituksiin verrattuna, $\chi^2(1) = 81.559$, $p < .001$. Todennäköisyys sille, että rakentavat virheilmoitukset koettiin hyödyllisiksi virheen korjaamisen kannalta, oli 0.979-kertainen (95 % luottamusväli, 0.825…1.162) ei-rakentaviin virheilmoituksiin verrattuna, $\chi^2(1) = 0.060$, $p = .807$. Todennäköisyys sille, että täsmälliset virheilmoitukset koettiin hyödyllisiksi virheen korjaamisen kannalta, oli 1.520-kertainen (95 % luottamusväli, 1.325…1.745) ei-täsmällisiin virheilmoituksiin verrattuna, $\chi^2(1) = 35.599$, $p < .001$. Todennäköisyys sille, että paikannetut virheilmoitukset koettiin hyödyllisiksi virheen

korjaamisen kannalta, oli 2.012-kertainen (95 % luottamusväli, 1.697…2.384) ei-paikannettuihin virheilmoituksiin verrattuna, $\chi^2(1) = 65.045$, $p < .001$. Taulukossa 5 on esitetty virheilmoitusten käytettävyyteen liittyvien ominaisuuksien regressiokertoimet ja tilastolliset merkitsevyydet virheen korjaamisen näkökulmasta.

Ominaisuus	**Regressiokerroin**	**Merkitsevyys**
Positiivisuus	5.322	$p < .001$
Rakentavuus	0.979	$p = .807$
Täsmällisyys	1.520	$p < .001$
Paikannettavuus	2.012	$p < .001$

Taulukko 5. Ominaisuuksien vaikutukset virheen korjaamiseen

5.4 Ominaisuuksien vaikutukset virhetilanteesta toipumiseen

Järjestysasteikollista logistista regressioanalyysiä käytettiin selvittämään, vaikuttavatko tietokannanhallintajärjestelmän virheilmoituksen ominaisuudet (positiivisuus, rakentavuus, täsmällisyys ja paikannettavuus) virheilmoituksen koettuun hyödyllisyyteen virhetilanteesta toipumisen näkökulmasta. Todennäköisyys sille, että positiiviset virheilmoitukset koettiin hyödyllisiksi virhetilanteesta toipumisen kannalta, oli 5.045-kertainen (95 % luottamusväli, 3.676…6.923) ei-positiivisiin virheilmoituksiin verrattuna, $\chi^2(1) = 100.455$, $p < .001$. Todennäköisyys sille, että rakentavat virheilmoitukset koettiin hyödyllisiksi virhetilanteesta toipumisen kannalta, oli 0.568-kertainen (95 % luottamusväli, 0.479…0.674) ei-rakentaviin virheilmoituksiin verrattuna, $\chi^2(1) = 42.008$, $p < .001$. Todennäköisyys sille, että täsmälliset virheilmoitukset koettiin hyödyllisiksi virhetilanteesta toipumisen kannalta, oli 0.881-kertainen (95 % luottamusväli, 0.769…1.010) ei-täsmällisiin virheilmoituksiin verrattuna, $\chi^2(1) = 3.304$, $p = .069$. Todennäköisyys sille, että paikannetut virheilmoitukset koettiin hyödyllisiksi virhetilanteesta toipumisen kannalta, oli 1.604-kertainen (95 % luottamusväli, 1.355…1.899) ei-paikannettuihin virheilmoituksiin verrattuna, $\chi^2(1) = 30.057$, $p < .001$. Taulukossa 6 on esitetty virheilmoitusten käytettävyyteen liittyvien ominaisuuksien regressiokertoimet ja tilastolliset merkitsevyydet virhetilanteesta toipumisen näkökulmasta.

Ominaisuus	**Regressiokerroin**	**Merkitsevyys**
Positiivisuus	5.045	$p < .001$
Rakentavuus	0.568	$p < .001$
Täsmällisyys	0.881	$p = .069$
Paikannettavuus	1.604	$p < .001$

Taulukko 6. Ominaisuuksien vaikutukset virhetilanteesta toipumiseen

6 Pohdinta

Luvussa esitetään pohdintoja tutkimustuloksiin liittyen. Lisäksi luvussa arvioidaan tutkimuksen luotettavuutta ja esitetään mahdollisia jatkotutkimusaiheita.

6.1 Tulosten merkitys tieteelle

Tietokannanhallintajärjestelmien virheilmoitusten käytettävyyteen liittyvien ominaisuuksien vaikutusta virheilmoitusten koettuun hyödyllisyyteen ei ole aiemmin tutkittu. Tähän tutkimukseen otettiin tulokulmaa ohjelmointikielten kääntäjien virheilmoitusten tutkimuksesta, joka on runsasta. Virheilmoitusten suunnittelun ja tehostamisen ohjenuoria ja näihin sisältyviä laadukkaiden virheilmoitusten ominaisuuksia on esitetty useissa tutkimuksissa. Tässä tutkimuksessa selvitettiin näiden ominaisuuksien vaikutusta PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten koettuun hyödyllisyyteen.

Tulokset osoittavat, että positiivisuus ja virheen oikea paikantaminen nostavat ymmärrettävästi virheilmoitusten koettua hyödyllisyyttä kaikkien tutkimuskysymysten osalta. Muun muassa Traver (2010) huomauttaa virheen oikean paikantamisen olevan tärkeää, koska väärin paikannettuna virhe voi hämmentää ohjelmoijaa. Useissa tutkimuksissa (mm. Traver 2010; Shneiderman 1982) on myös korostettu virheilmoitusten positiivisen sävyn tärkeyttä. Epämääräisten ja tylyjen ilmausten sijaan virheilmoitusten tulisi sisältää ystävällisessä sävyssä esitettyä rakentavaa informaatiota, joka auttaa käyttäjää virheen korjaamisessa. Tältä osin tulokset siis vahvistavat positiivisuuden ja paikannettavuuden merkityksen laadukkaiden virheilmoitusten ominaisuuksina.

Tulosten perusteella rakentavuus laskee virheilmoitusten koettua hyödyllisyyttä kaikkien tutkimuskysymysten osalta, mutta ainoastaan virhetilanteesta toipumisen näkökulmasta (TK3) vaikutus on tilastollisesti merkitsevä. Tämä tulos on mielenkiintoinen ja melko yllättävä. Aineiston koodausvaiheessa virheilmoitukset, jotka sisälsivät virheen korjaamista helpottavaa informaatiota, koodattiin rakentaviksi. Rakentavan informaation lisääminen virheilmoituksiin muun muassa esimerkein ja vihjein pidentää luonnollisesti virheilmoitusta. Samalla myös virheilmoituksen sisältämä kognitiivinen kuorma kasvaa. Sweller (1988) ja

Smelcer (1995) huomauttavat ihmisen työmuistin kapasiteetin olevan rajallinen, joten ihminen kykenee käsittelemään vain rajallisen määrän informaatiota kerralla tehokkaasti. Ehkä vastaajat siten kokevat rakentavan informaation ylimääräisenä ja ehkä jopa virhetilanteesta toipumista häiritsevänä tekijänä. Toisaalta kaikki virheilmoitukset koodattiin lyhyiksi, joten pituutensa puolesta niiden kognitiivisen kuorman ei pitäisi olla kohtuuttoman suuri. PostgreSQL:n virheilmoituksissa esitetyt vihjeet virheen korjaamiseksi ovat useiden virheilmoitusten kohdalla melko geneerisiä. Tämä voi myös olla osasyynä rakentavuuden vähäiseen ja jopa negatiiviseen vaikutukseen virheilmoitusten koetun hyödyllisyyden selittäjänä. Täsmällisemmät vihjeet auttaisivat luultavasti käyttäjää enemmän.

Denny ym. (2014) huomauttavat, että huonot, epätarkat ja epätäsmälliset virheilmoitukset voivat aiheuttaa jopa uusia virheitä. Tutkituissa virheilmoituksissa niiden rakentava luonne saattaa siis hämmentää vastaajia, jos rakentava informaatio on kovin epätäsmällistä. Rakentavuuden huomioiminen paremmin virheilmoituksissa lisäisi luultavasti niiden koettua hyödyllisyyttä ja tehostaisi todennäköisesti myös SQL-kielen oppimista.

Virheilmoitusten käytettävyyteen liittyvissä ominaisuuksissa on jokseenkin päällekkäisiä piirteitä, minkä vuoksi niiden erottelu toisistaan on toisinaan hieman hankalaa. Shneidermanin (1982) mukaan esimerkiksi täsmällisyys tarkoittaa virheen riittävän tarkkaa kuvaamista ilman epämääräisiä ilmauksia. Hän käyttää esimerkkinä epämääräisyydestä ilmausta *SYNTAX ERROR*. Periaatteessa täsmällisen virheilmoituksen voitaisiin ajatella osoittavan myös virheen sijainnin ohjelman lähdekoodissa tai SQL-kyselyssä. Tässä tutkimuksessa paikannettavuus erotettiin kuitenkin erilliseksi ominaisuudeksi. Joidenkin virheilmoitusten osalta virhe on kuvattu riittävän täsmällisesti, mutta sitä ei ole kuitenkaan paikannettu täysin oikein. Tutkimustuloksista voidaan havaita, että virhetilanteesta toipumisen näkökulmasta (TK3) täsmällisyydellä ei ole vaikutusta virheilmoitusten koettuun hyödyllisyyteen. Paikannettavuudella sen sijaan on positiivinen vaikutus koettuun hyödyllisyyteen.

Etenkin laajemmissa ohjelmistoissa virheiden paikantamiseen liittyy haasteita, jotka voivat hankaloittaa virheiden korjaamista. Kuten luvussa 3.3 mainittiin, virhe voidaan havaita kaukana todellisesta sijainnistaan, jolloin sen paikantaminen täydellisesti voi olla vaikeaa. SQL-kyselyt ja ohjelmistot eroavat usein laajuudessa toisistaan, joten ainakin periaatteessa

lyhyemmissä SQL-kyselyissä virheen paikantamisen pitäisi olla helpompaa. Virheilmoituksessa jopa hieman puutteellisesti paikannettu virhe voi auttaa käyttäjää riittävästi SQL-kyselyn korjaamisessa. Joka tapauksessa virheen paikantamisella on kiistatta suuri merkitys sekä ohjelmointikielen kääntäjän että tietokannanhallintajärjestelmän virheilmoituksissa.

6.2 Tulosten merkitys teollisuudelle

Tutkimuksen tulokset osoittavat, että positiiviset, täsmälliset ja paikannetut virheilmoitukset vaikuttavat positiivisesti virheilmoitusten koettuun hyödyllisyyteen. Tietokannanhallintajärjestelmien toimittajien kannattaakin huomioida näiden ominaisuuksien merkitys virheilmoitusten suunnittelussa. PostgreSQL:n osalta näitä ominaisuuksia voitaisiin edelleen kehittää virheilmoituksissa melko paljonkin. Virheilmoitusten käytettävyyteen liittyviä ominaisuuksia kehittämällä tietokannanhallintajärjestelmän toimittaja voi myös saada etua suhteessa kilpailijoihin. Traver (2010) huomauttaa virheilmoitusten laadun vaikuttavan merkittävästi ohjelmoijien työhön, ja sitä kautta koko ohjelmistokehitysprojektiin ja sen lopputuotteena syntyvään ohjelmistoon. Tietokannanhallintajärjestelmän virheilmoituksilla on yhtä lailla vaikutusta ohjelmistokehitysprojektien tehokkuuteen.

Virheilmoitusten tehostamiseen on olemassa useita keinoja, joita on esitelty kattavasti luvussa 3.3. Tutkimustulosten perusteella virheilmoitusten rakentavuudessa on paljon parannettavaa, jotta kyseinen ominaisuus koettaisiin hyödylliseksi virheen havaitsemisen, korjaamisen ja virhetilanteesta toipumisen näkökulmista. Rakentavuus ominaisuutena vaatii varmasti vielä tarkempaa lisätutkimusta, mutta tietokannanhallintajärjestelmien kehittäjät voisivat huomioida rakentavuuden paremmin suunnitellessaan virheilmoituksia. Rakentavan informaation sisällyttäminen virheilmoituksiin voisi onnistua pidentämättä virheilmoitusta kohtuuttomasti. Oletetaan esimerkiksi, että käyttäjä on tehnyt SQL-kyselyssään liitoksen taulujen välillä IN-predikaattia käyttäen ja alikyselyn SELECT-osassa on esitelty enemmän kuin yksi taulu. Tietokannanhallintajärjestelmän virheilmoitus voisi olla tällöin muotoa: *It looks like there is a problem on line number <line number>: Subquery can have only one column*. Näin ollen virheilmoitus kertoo käyttäjälle virheen sijainnin, syyn sekä vihjeen virheen korjaamiseksi. Virheilmoitus on myös suhteellisen ytimekäs, ja se on esitetty luonnollista kieltä käyttäen ymmärrettävässä muodossa.

6.3 Tutkimuksen rajoitteet

Yksi merkittävimmistä tutkimuksen luotettavuuteen vaikuttavista tekijöistä liittyy aineiston koodaamiseen. Virheilmoitukset koodattiin tutkijan henkilökohtaisen näkemyksen mukaan, joten koodauksissa on mahdollisesti jonkin verran tulkinnanvaraa. Tutkijan omat kokemukset virheilmoituksista ovat voineet vaikuttaa koodausprosessiin. Koodaamista hankaloitti joiltakin osin myös konkreettisten ohjeiden puute siitä, miten tietty ominaisuus ilmenee virheilmoituksessa. Virheilmoitukset koodattiin Shneidermanin (1982) heuristiikkojen mukaan, mutta esimerkiksi ymmärrettävyyden osalta heuristiikat jättävät paljon tulkinnanvaraa. Shneidermanin (1982) mainitsema ymmärrettävyys viittaa Beckerin ym. (2019) esittämään luettavuuteen, mutta luettavuuden saavuttamiseksi tai arvioimiseksi ei ole toistaiseksi esitetty konkreettisia ohjeita. Kaikki virheilmoitukset koodattiin lyhyiksi ja ymmärrettäviksi, vaikka etenkin ymmärrettävyyden osalta virheilmoituksissa olisi melko paljonkin parantamisen varaa. Vaikka nämä ominaisuudet jätettiinkin regressiomallin ulkopuolelle, ovat ne esimerkkejä koodauksen tulkinnanvaraisuudesta. Huomionarvoista on myös se, että vain yksi virheilmoitus koodattiin positiiviseksi. Tälläkin on mahdollisesti merkitystä tulosten kannalta.

Tutkimusmenetelmäksi valittiin järjestysasteikollinen logistinen regressioanalyysi, koska riippuvien ja riippumattomien muuttujien luonteesta johtuen kyseinen menetelmä vaikutti tähän tutkimukseen sopivalta. Tutkimustulokset ovat melko hyvin linjassa aiemmin tehtyjen tutkimusten kanssa, koska positiivisuuden, täsmällisyyden ja paikannettavuuden on todettu parantavan virheilmoitusten laatua. Rakentavuuden osalta tulokset jättävät sen sijaan aihetta pohtimiseen. Määrällistä analyysiä varten aineistoa saatiin kerättyä varsin hyvin, koska 190 vastaajaa osallistui tutkimukseen. Näin ollen analysoituja virheilmoituksia oli kaiken kaikkiaan 16 * 190 = 3040.

Tutkimukseen osallistuneet vastaajat olivat Jyväskylän yliopiston informaatioteknologian tiedekunnan *Tietokannat ja tiedonhallinta* -kurssin opiskelijoita. Lähtökohtaisesti vastaajat olivat osaamisensa puolesta samalla tasolla, eikä ainakaan suurimmalla osalla heistä ollut aiempaa kokemusta SQL-kielestä ennen kurssia. Näin ollen tutkimuksessa saatiin varsin hyvin tietoa siitä, miten hyödyllisiksi nimenomaan aloittelijat kokevat PostgreSQL:n

virheilmoitukset. Ennen kyselyyn vastaamista vastaajat olivat osallistuneet kurssin opetukseen tärkeimmistä asiasisällöistä, joten ainakin periaatteessa heillä oli tarvittavat tietotaidot suoriutua hyvin kyselyn tehtävistä.

6.4 Jatkotutkimusaiheita

Tutkimuksessa selvitettiin positiivisuuden, rakentavuuden, täsmällisyyden ja paikannettavuuden vaikutusta virheilmoitusten koettuun hyödyllisyyteen. Nämä ominaisuudet perustuvat paikannettavuutta lukuun ottamatta Shneidermanin (1982) esitykseen virheilmoitusten käytettävyyttä parantavista ominaisuuksista. Kirjallisuudessa on esitetty myös lukuisia muita ominaisuuksia, joiden vaikutusta virheilmoitusten koettuun hyödyllisyyteen olisi mielenkiintoista tutkia. Esimerkiksi Horning (1976), Traver (2010) ja Barik (2018) esittävät virheilmoitusten laatuun vaikuttavia ominaisuuksia. Eri tutkijoiden esittämissä ominaisuuksissa on toki päällekkäisyyksiä, mutta ominaisuuksien määrittelyt voivat hieman poiketa toisistaan.

Virheilmoitusten tehostamisen ohjenuorat ja niihin sisältyvät ominaisuudet vaativat vielä paljon empiirisiä todisteita siitä, mikä niiden vaikutus virheilmoitusten tehokkuuteen todellisuudessa on (Becker ym. 2019). Tulevissa tutkimuksissa voisi olla hyödyllisempää keskittyä tutkimaan tietyn yksittäisen ominaisuuden vaikutusta virheilmoitusten hyödyllisyyteen. Esimerkiksi Shneidermanin (1982) esittämälle ymmärrettävyydelle, jolla Becker ym. (2019) tarkoittavat luettavuutta, tulisi määrittää konkreettisempia mittareita sen saavuttamiseksi ja arvioimiseksi. Kun ominaisuus on määritelty tarkasti, on sen vaikutuksen selvittäminen esimerkiksi virheilmoitusten hyödyllisyyteen helpompaa.

Erityisesti rakentavuuden osalta tutkimustulokset ovat mielenkiintoisia ja jopa yllättäviä. Kyseisen ominaisuuden vaikutusta virheilmoitusten hyödyllisyyteen olisi jatkossa mielenkiintoista tutkia tarkemmin esimerkiksi virheilmoitusten tehostamisen kautta. Tehostamalla virheilmoitusten rakentavia piirteitä voitaisiin vertailla, onko tehostamisella vaikutusta virheilmoitusten hyödyllisyyteen. Virheilmoitusten tehostaminen olisi ylipäätään mielenkiintoinen näkökulma tutkimukseen.

Luonnollisesti olisi mielenkiintoista selvittää, miten virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoitusten koettuun hyödyllisyyteen muissa suosituimmissa tietokannanhallintajärjestelmissä. Tutkimustulosten perusteella voitaisiin vertailla esimerkiksi rakentavuuden vaikutusta virheilmoitusten hyödyllisyyden selittäjänä eri tietokannanhallintajärjestelmissä.

7 Yhteenveto

Tässä tutkimuksessa selvitettiin, mitkä PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten käytettävyyteen liittyvät ominaisuudet vaikuttavat virheilmoitusten koettuun hyödyllisyyteen aloittelijoiden näkökulmasta. Tutkitut ominaisuudet olivat *positiivisuus*, *rakentavuus*, *täsmällisyys* ja *paikannettavuus*. Tulosten perusteella positiivisuudella ja paikannettavuudella on positiivinen vaikutus virheilmoitusten koettuun hyödyllisyyteen virheen havaitsemisen, korjaamisen ja virhetilanteesta toipumisen näkökulmista. Täsmällisyys vaikuttaa myös positiivisesti virheilmoitusten hyödyllisyyteen virheen havaitsemisen ja korjaamisen näkökulmista. Rakentavuuden osalta tulokset ovat hieman yllättäviä. Rakentavuus vaikuttaa virheilmoituksen koettuun hyödyllisyyteen ainoastaan virhetilanteesta toipumisen näkökulmasta, ja vaikutus on negatiivinen.

Aiemmin tehtyjen tutkimusten mukaan edellä mainittujen ominaisuuksien huomioiminen virheilmoitusten suunnittelussa parantaa virheilmoitusten laatua. Tämän tutkimuksen tulokset vahvistavat, että positiivisuuden, täsmällisyyden ja paikannettavuuden huomioiminen virheilmoituksissa lisää PostgreSQL-tietokannanhallintajärjestelmän virheilmoitusten koettua hyödyllisyyttä. Ominaisuuksien vaikutusta virheilmoitusten hyödyllisyyteen on kuitenkin syytä tutkia lisää tulevaisuudessa. Useiden ominaisuuksien osalta vaaditaan vielä lisää empiiristä näyttöä niiden todellisista vaikutuksista virheilmoitusten hyödyllisyyteen. Etenkin rakentavuuden vaikutusta virheilmoitusten hyödyllisyyteen on syytä tutkia vielä tarkemmin. Virheilmoitusten ominaisuuksien määrittelyyn olisi suotavaa esittää konkreettisempia ohjeita tulkinnanvaraisuuden vähentämiseksi.

Lähteet

Ahadi, A., Behbood, V., Vihavainen, A., Prior, J. & Lister, R. 2016. Students' Syntactic Mistakes in Writing Seven Different Types of SQL Queries and its Application to Predicting Students' Success. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16)*.

Ayres, P. L. & Sweller, J. 1990. Locus of Difficulty in Multistage Mathematics Problems. *The American Journal of Psychology 103 (2), s. 167–193*.

Ayres, P. L. 2001. Systematic Mathematical Errors and Cognitive Load. *Contemporary Educational Psychology 26 (2), s. 227–248*.

Barik, T., Ford. D., Murphy-Hill, E. & Parnin, C. 2018. How Should Compilers Explain Problems to Developers? *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018), s. 633-643*.

Becker, B. A. 2016. An Effective Approach to Enhancing Compiler Error Messages. *SIGCSE '16 Proceedings of the 47th ACM Technical Symposium on Computing Science Education*.

Becker, B. A., Glanville, G., Iwashima, R., McDonnell, C., Goslin, K. & Mooney, C. 2016. Effective Compiler Error Message Enhancement for Novice Programming Students. *Computer Science Education 26 (2-3), s. 148–175*.

Becker, B. A. 2019. Parlez-vous Java? Bonjour La Monde != Hello World: Barriers to Programming Language Acquisition for Non-Native English Speakers. *Proceedings of the 30th Annual Conference of the Psychology of Programming Interest Group (PPIG '19)*.

Becker, B. A., Denny, P., Pettit, R., Bouchard, D., Bouvier, D. J., Harrington, B., Kamil, A., Karkare, A., McDonald, C., Osera, P-M., Pearce, J. L. & Prather, J. 2019. Compiler Error Messages Considered Unhelpful: The Landscape of Text-Based Programming Error Message Research. *ITiCSE Working Group Reports, July 15-17, 2019, Aberdeen, Scotland UK.*

Brass, S. & Goldberg, C. 2005. Semantic Errors in SQL Queries: A Quite Complete List. *Journal of Systems and Software 79 (5) s. 630-644*.

Chen, P. P-S. 1976. The Entity-Relationship Model – Toward a Unified View of Data. *ACM Transactions on Database Systems 1 (1), s. 9-36.*

Codd, E. F. 1970. A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM 13 (6), s. 377-387.*

Connolly, T. M. & Begg, C. E. 2005. Database Systems: A Practical Approach to Design, Implementation, and Management (4th ed.). *Addison-Wesley*.

Coull, N. J. & Duncan, I. M. M. 2011. Emergent Requirements for Supporting Introductory Programming. *Innovations in Teaching and Learning in Information and Computer Sciences (ITaLICS) 10 (1), s. 78–85*.

Darwen, H. 2010. An Introduction to Relational Database Theory. *Ventus Pub-lishing ApS.*

Date, C. J. 1995. An Introduction to Database Systems (6th ed.). *Addison-Wesley.*

DB-Engines. 2021. DB-Engines Ranking. Haettu 4.4.2021 osoitteesta https://db-engines.com/en/ranking.

Denny, P., Luxton-Reilly, A. & Carpenter, D. 2014. Enhancing Syntax Error Messages Appears Ineffectual. *Proceedings of the 19th Conference on Innovation and Technology in Computer Science Education (ITiCSE '14)*.

Eisenberg, A. & Melton, J. 1999. SQL:1999, formerly known as SQL3. *SIGMOD Record 28 (1), s. 131-138.*

Eisenberg, A., Melton, J., Kulkarni, K., Michels, J-E. & Zemke, F. 2004. SQL:2003 Has Been Published. *SIGMOD Record 33 (1), s. 119-126.*

Elmasri, R. & Navathe, S. B. 2017. Fundamentals of Database Systems (7th ed.). *Pearson*.

Flowers, T., Carver, C., & Jackson, J. 2004. Empowering Students and Building Confidence in Novice Programmers through Gauntlet. *34th ASEE/IEEE Annual Frontiers in Education, FIE 2004*.

Foster, E. C. & Godbole, S. 2016. Database Systems: A Pragmatic Approach. *Apress.*

Harrington, J. L. 2009. Relational database design and implementation: clearly explained. *Morgan Kaufmann/Elsevier.*

Horning, J. J. 1976. What the Compiler Should Tell the User. *Compiler Construction: An Advanced Course, G Goos and J Hartmanis (Eds.). SpringerVerlag, Berlin-Heidelberg, s. 525–548*.

Hundhausen, C. D., Olivares, D. M. & Carter, A. S. 2017. IDE-Based Learning Analytics for Computing Education: A Process Model, Critical Review, and Research Agenda. *ACM Transactions on Computing Education 17 (3)*.

Kantorowitz, E. & Laor, H. 1986. Automatic Generation of Useful Syntax Error Messages. *Software: Practice and Experience 16 (7), s. 627–640*.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J & Szabo, C. 2018. Introductory Programming: A Systematic Literature Review. *Proceedings of 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18)*.

McCall, D. & Kölling, M. 2014. Meaningful Categorisation of Novice Programmer Errors. *IEEE Frontiers in Education Conference (FIE '14)*.

McIver, L & Conway, D. 1996. Seven Deadly Sins of Introductory Programming Language Design. *International Conference on Software Engineering: Education and Practice (SEEP '96)*.

Pettit, R. S., Homer, J. & Gee, R. 2017. Do Enhanced Compiler Error Messages Help Students? Results Inconclusive. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*.

PostgreSQL. 2021a. What is PostgreSQL? Haettu 4.4.2021 osoitteesta https://www.postgresql.org/about/.

PostgreSQL. 2021b. The Path of a query. Haettu 5.4.2021 osoitteesta https://www.postgresql.org/docs/current/query-path.html.

PostgreSQL. 2021c. The Parser Stage. Haettu 6.4.2021 osoitteesta https://www.postgresql.org/docs/current/parser-stage.html.

Shneiderman, B. 1982. Designing computer System Messages. *Communications of the ACM 25 (9), s. 610-611*.

Silberschatz, A., Korth, H. F. & Sudarshan, S. 2006. Database System Concepts (5th ed.). *McGraw-Hill*.

Smelcer, J. B. 1995. User errors in database query composition. *International Journal of Human-Computer Studies 42 (4), s. 353-381*.

Sweller, J. 1988. Cognitive Load During Problem Solving: Effects on Learning. *Cognitive science 12 (2), s. 257–285*.

Taipalus, T., Siponen, M. & Vartiainen, T. 2018. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education 18 (3)*.

Taipalus, T. & Perälä, P. 2019. What to Expect and What to Focus on in SQL Query Teaching. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19), s. 198-203*.

Taipalus, T. 2020. Explaining Causes Behind SQL Query Formulation Errors. *Proceedings of the 50th IEEE Frontiers in Education Conference*.

Traver, V. J. 2010. On Compiler Error Messages: What they Say and What They Mean. *Advances in Human-Computer Interaction.* Haettu 25.2.2021 osoitteesta https://www.hindawi.com/journals/ahci/2010/602570/.

Watson, C., Li, F. W. B. & Godwin, J. L. 2012. BlueFix: Using Crowd-Sourced Feedback to Support Programming Students in Error Diagnosis and Repair. *Proceedings of the 11th International Conference on Advances in Web-Based Learning (ICWL'12)*.

Welty, C. 1985. Correcting user errors in SQL. *International Journal of Man-Machine Studies 22 (4), s. 463-477*.

Liitteet

A Kyselylomake (tehtävä 1)

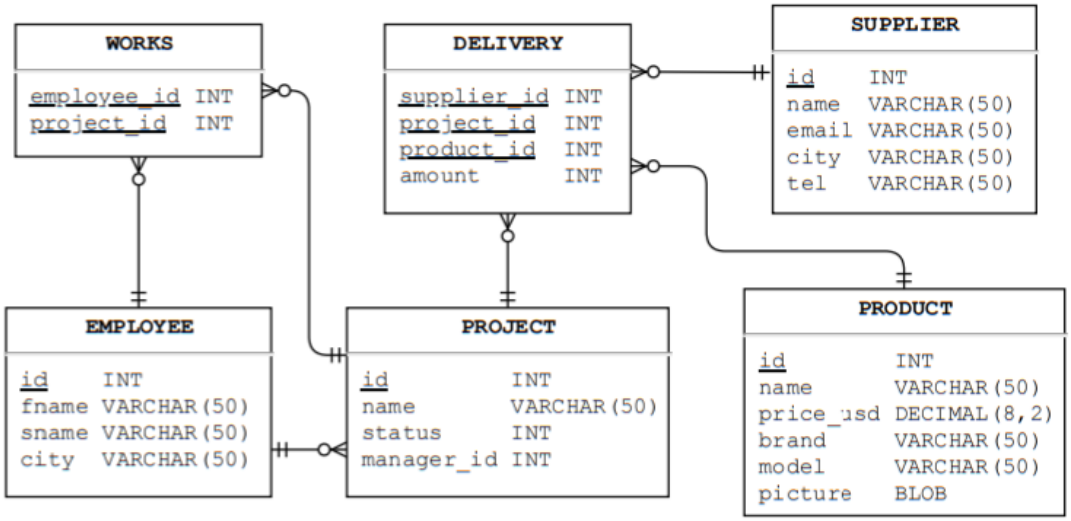


Fig. 1. The database schema

Find the names of suppliers who have delivered at least one Apple product priced over 50 USD.

```
SELECT name
FROM   supplier
JOIN   delivery    ON (supplier.id = delivery.supplier_id)
JOIN   product     ON (delivery.product_id = product.id)
WHERE  product.price_usd > 50
AND    product.brand = 'Apple';
```

```
ERROR:  column reference "name" is ambiguous
LINE 1: SELECT  name
                ^
```

Please type the fixed SQL query here.

Please evaluate the error message (1 = strongly disagree, 2 = disagree, 3 = neutral, 4 = agree, 5 = strongly agree).

	1	2	3	4	5
The error message was useful in finding the error	☐	☐	☐	☐	☐
The error message was useful in fixing the error	☐	☐	☐	☐	☐
The error message increased my confidence in error recovery	☐	☐	☐	☐	☐