

EIRÍKUR ERNIR ÞORSTEINSSON

# NOTKUN GAGNASAFNA - NÁMSEFNI

TÆKNISKÓLINN

# Efnisyfirlit

<b>1</b>	<b>Inngangur</b>	<b>12</b>
1.1	Hvað er gagnagrunnur?	12
1.2	Hvað er SQL?	12
1.3	Af hverju SQL?	13
1.4	Hvað er gagnagrunnskerfi?	13
	Hvernig vinnum við með gagnagrunnskerfi?	13
1.5	Yfirlit	13
<b>2</b>	<b>Fyrstu skrefin í SQL</b>	<b>15</b>
2.1	Gagnagrunnar	15
2.2	Töflur	16
2.3	Fyrirspurnir	17
2.4	Sýnidæmi í SQL	17
	Skilgreining gagnagrunns	17
	Skilgreining töflu og innsetning gagna	18
	Fyrirspurnir	18
2.5	Keyrsla í MySQL Workbench	19
2.6	Yfirlit	24
<b>3</b>	<b>Uppsetning tafna</b>	<b>25</b>
3.1	Að búa til töflu	25
3.2	Innsetning gagna	26
3.3	Algengar gagnagerðir: Tölur og texti	28
	Heiltölur - INTEGER	28

Texti - VARCHAR og CHAR . . . . .	29
3.4 Dæmi um töflur með tölum og texta . . . . .	31
3.5 Fleiri gagnagerðir . . . . .	33
Tugabrot - DECIMAL . . . . .	33
Rauntölur - DOUBLE . . . . .	33
Dagsetningar - DATE . . . . .	34
3.6 Tóm gildi . . . . .	35
NOT NULL . . . . .	36
3.7 Aðallyklar - PRIMARY KEY . . . . .	37
3.8 Að eyða töflum . . . . .	38
3.9 Yfirlit . . . . .	39
<b>4 Fyrirspurnir . . . . .</b>	<b>40</b>
4.1 SELECT . . . . .	40
SELECT - FROM . . . . .	40
4.2 WHERE klausan . . . . .	44
Röksegðir . . . . .	44
LIKE og “wildcards” . . . . .	47
Mörg aðskilin skilyrði . . . . .	49
Að velja úr mengi - IN . . . . .	50
Að snúa við skilyrði - NOT . . . . .	50
4.3 Um föll . . . . .	51
4.4 Helstu “scalar” föll . . . . .	51
Lengd strengs - CHAR_LENGTH . . . . .	52
UCASE og LCASE . . . . .	52
NOW . . . . .	53
Notkun scalar falla . . . . .	53
4.5 GROUP BY . . . . .	54
4.6 Samsteypuföll . . . . .	54
COUNT . . . . .	55
Önnur samsteypuföll - MIN, MAX, SUM, og AVG . . . . .	58
4.7 HAVING . . . . .	59

4.8	ORDER BY . . . . .	59
4.9	LIMIT . . . . .	61
4.10	Uppbygging SELECT skipunarinnar . . . . .	62
4.11	Yfirlit . . . . .	63
<b>5</b>	<b>Að setja upp gagnagrunn</b>	<b>64</b>
5.1	Lyklar . . . . .	64
	Almennt um lykla - KEY/INDEX . . . . .	64
	Einkvæmir lyklar - UNIQUE KEY . . . . .	65
	Aðallyklar - PRIMARY KEY . . . . .	65
5.2	Margar töflur í sama gagnagrunninum . . . . .	66
5.3	Aðkomulyklar - FOREIGN KEY . . . . .	67
5.4	Margar samtengdar töflur . . . . .	68
5.5	Tengingar . . . . .	71
	1-N: Einn tengdur í marga . . . . .	71
	1-1: Einn tengdur í einn . . . . .	71
	N-N: Margir tengdir í marga . . . . .	71
5.6	Að sjá yfirlit gagnagrunns í MySQL Workbench . . . . .	72
5.7	(Ekki) meira um lykla . . . . .	74
5.8	Yfirlit . . . . .	74
<b>6</b>	<b>Að sameina gögn</b>	<b>75</b>
6.1	Nöfn dálka . . . . .	75
	Að endurnefna - AS . . . . .	76
6.2	Að velja úr meira en einni töflu - JOIN . . . . .	76
	INNER JOIN . . . . .	77
	ÖNNUR JOIN . . . . .	79
6.3	Undirfyrirspurnir . . . . .	80
	Mismunandi hlutverk undirfyrirspurna . . . . .	81
6.4	Yfirlit . . . . .	81
<b>7</b>	<b>Að uppfæra gagnagrunna</b>	<b>82</b>
7.1	DDL og DML . . . . .	82

7.2	Að breyta töflum . . . . .	83
7.3	Að breyta gögnum . . . . .	83
7.4	Að eyða gögnum . . . . .	84
7.5	Viðhald gagnagrunna . . . . .	85
	Aðkomulyklar og breytingar . . . . .	85
	Hreyfingar . . . . .	85
7.6	Yfirlit . . . . .	86
<b>8</b>	<b>Ítarefni</b>	<b>87</b>
8.1	Views . . . . .	87
8.2	Yfirlit yfir helstu gagnagrunnskerfi . . . . .	89
	MySQL . . . . .	89
	PostgreSQL . . . . .	89
	SQLite . . . . .	90
	Microsoft SQL Server . . . . .	90
	Oracle Database . . . . .	90
8.3	Venslálíkanið . . . . .	90
	Mengi . . . . .	91
	Vensl . . . . .	91
8.4	Að tengjast gagnagrunni með PHP . . . . .	92
	Hlutverk gagnagrunna í vefsíðum . . . . .	92
	Uppsetning tengingar með PDO . . . . .	93
	Að sækja gögn með PDO . . . . .	94
	Að setja saman HTML og PHP . . . . .	95
<b>9</b>	<b>Um þessa bók</b>	<b>98</b>
9.1	Um útgáfu . . . . .	98
9.2	Tæknileg atriði . . . . .	98
9.3	Leyfi . . . . .	98
9.4	Þessi útgáfa . . . . .	99

# Myndaskrá

2.1	Uppbygging gagnagrunns	15
2.2	Upphafsskjár MySQL Workbench	20
2.3	Tengingarskjár MySQL Workbench	21
2.4	Nýr gagnagrunnur	22
2.5	Ný tafla	23
3.1	Að sjá töflu í Workbench	26
3.2	CHAR og VARCHAR	30
3.3	High Score tafla	31
3.4	Matseðill	32
4.1	Niðurstöður SELECT í Workbench	42
4.2	Niðurstöður margra dálka SELECT í Workbench	43
4.3	Niðurstöður margra dálka SELECT í Workbench	45
4.4	Röksegð í WHERE klausu	46
4.5	Samsett röksegð	49
4.6	Fall	51
4.7	GROUP BY	55
4.8	GROUP BY og COUNT eftir fögum	56
4.9	GROUP BY og COUNT eftir önnum	57
4.10	FROM, WHERE, GROUP BY og HAVING	60
5.1	Tengsl taflna	68
5.2	Sambandsgerðir	71
5.3	Workbench - Reverse Engineer takki	72

5.4	Tækniskólagagnagrunnurinn	73
6.1	INNER JOIN	77
6.2	INNER JOIN niðurstaða	78
6.3	Mengi	79
7.1	DDL og DML	82
8.1	View	87
8.2	MySQL	89
8.3	PostgreSQL	89
8.4	SQLite	90
8.5	SQL Server	90
8.6	Oracle Database	90
8.7	Heimasíðan	97

# Töfluskrá

2.1	Nokkrir starfsmenn Tækniskólans	16
3.1	Eftir einfalt INSERT	27
3.2	Eftir INSERT á línu	27
3.3	Eftir INSERT á tveimur línunum	27
3.4	Heiltöludálkar	28
3.5	Eldsneyti	33
3.6	Plánetur utan sólkerfisins.	34
4.1	Nemendur	41
4.2	Röksegðir	45
4.3	Röksegðir með strengjum	47
4.4	AND og OR	49
4.5	Lengd nafna	54
4.6	Áfangar	55
4.7	Einkunnir	58
5.1	Áfangar	66
5.2	Fög	67
5.3	Áfangar - endurbætt	67



# Sýnidæmaskrá

2.1	CREATE DATABASE skipun fyrir Tækniskólagagnagrunninn.	17
2.2	CREATE TABLE skipun fyrir starfsmannatöfluna.	18
2.3	INSERT skipun fyrir starfsmannatöfluna.	18
2.4	SELECT skipun sem finnur nafnið á þeim kennara sem er með netfangið <i>kng@tskoli.is</i> . Það nafn er "Konráð".	19
3.1	Mjög einföld tafla	25
3.2	Einföld tafla	26
3.3	INSERT í einfalda töflu	26
3.4	INSERT í tvo dálka í einu	27
3.5	INSERT í tvo dálka í einu	27
3.6	High Score tafla	31
3.7	Matseðill	32
3.8	Eldsneytisverð	33
3.9	Plánetur	34
3.10	Afmælis dagar	35
3.11	Mannanöfn	36
3.12	NOT NULL	36
3.13	DROP TABLE	38
4.1	Lágmarks SELECT	40
4.2	SELECT FROM	41
4.3	SELECT með WHERE klausu - eftir númeri	44
4.4	SELECT með WHERE klausu - eftir nafni	44
4.5	SELECT með WHERE klausu - endurtekin gildi	44
4.6	Stærri-en samanburður við streng	46

4.7	LIKE til að finna orð sem byrja á sama staf	47
4.8	LIKE til að finna orð sem enda eins	47
4.9	LIKE einhvers staðar í streng	48
4.10	LIKE til að finna strengi af ákveðinni lengd	48
4.11	LIKE með nákvæmum stafafjölda	48
4.12	WHERE með AND	50
4.13	WHERE með OR	50
4.14	WHERE með IN	51
4.15	NOT lykilorðið	51
4.16	CHAR_LENGTH	52
4.17	UCASE	52
4.18	CHAR_LENGTH í dálklýsingu	53
4.19	CHAR_LENGTH í WHERE klausu	53
4.20	COUNT á dálk	56
4.21	MIN fallið	58
4.22	AVG fallið	59
4.23	HAVING	59
4.24	ORDER BY	61
4.25	ORDER BY með DESC	61
4.26	ORDER BY með tveimur dálkum	61
4.27	LIMIT	62
4.28	LIMIT með tveimur tölum	62
4.29	Uppbygging	63
5.1	PRIMARY KEY	66
5.2	FOREIGN KEY - tengdar töflur	69
5.3	FOREIGN KEY - abstract dæmi	69
5.4	Starfsmenn og nemendur	70
5.5	Hópar	70
5.6	Hópskráning: N-N	72
6.1	Fullt nafn dálks	76
6.2	Endurnefning	76

6.3	INNER JOIN	78
6.4	INNER JOIN	79
6.5	Meðaltal - upprifjun	80
6.6	Undirfyrirspurn	80
7.1	ALTER TABLE	83
7.2	UPDATE með WHERE	84
7.3	UPDATE án WHERE	84
7.4	DELETE	84
8.1	View	88
8.2	SELECT úr view	88
8.3	Tenging við gagnagrunn með PDO	93
8.4	Upplýsingar sóttar úr gagnagrunni með PHP	95
8.5	HTML-beinagrind	96
8.6	PHP + HTML	97

# 1

## Inngangur

Skjal þetta er ætlað nemendum til gagns og stuðnings í fyrsta áfanga Tækniskólans er varðar notkun gagnagrunna með SQL.

### 1.1 Hvað er gagnagrunnur?

Gagnagrunnur<sup>1</sup> er, í sinni víðustu skilgreiningu, skipulagt samansafn af upplýsingum.

<sup>1</sup> e. *database*

Í þessari bók munum við skoða svokallaða SQL-gagnagrunna. Líta má á sem svo að slíkir gagnagrunnar samanstandi fyrst og fremst af *töflum* sem geyma upplýsingarnar. Það að smíða slíkar töflur, breyta þeim og birta úr þeim upplýsingar með SQL er aðalviðfangsefni áfangans.

### 1.2 Hvað er SQL?

Skammstöfunin SQL stendur fyrir **Structured Query Language**. Skoðum þá skammstöfun nánar.

*Query Language* hefur verið þýtt á íslensku sem *fyrirspurnamál*. SQL er sem sagt mál, líkt og tungumál og forritunarmál, sem nota má til að eiga ákveðin samskipti. SQL er notað til að senda *fyrirspurnir* á gagnagrunnskerfi, oftast í þeim tilgangi að fá upplýsingar frá kerfinu.

*Structured* bendir til þess að málið hafi ákveðna uppbyggingu. Tungumál sem fólk notar til samskipta sín á milli eru oftast mjög sveigjanleg og fær um að koma upplýsingum til skila á marga mismunandi vegu. Tölvur eru hins vegar ekki svo klárar að þær skilji hugtök jafn vel og fólk. Þess vegna þurfa þær fyrirspurnir sem við skrifum að vera á mjög fastmótuðu sniði svo að þær komist til skila. Það að læra SQL snýst að miklu leyti um að læra þetta snið - hvað er leyfilegt innan þess og hvað ekki.

### 1.3 Af hverju SQL?

Vinnsla gagna er stór hluti af nær öllum stórum tölvukerfum, allt frá einföldum vefsíðum upp í stór bankakerfi. Áratuga reynsla hefur sýnt að SQL er mjög hentugt til slíkrar vinnslu. Oftast er ekki verið að vinna með gagnagrunna til þess eins að eiga gagnagrunna, heldur vegna afslins sem gagnagrunnar hafa upp á að bjóða sem hluti af stærra kerfi. Í kafla 8.4 munum við sjá dæmi um hvernig nota má SQL við heimasíðugerð.

### 1.4 Hvað er gagnagrunnskerfi?

Gagnagrunnskerfi<sup>2</sup> er sá hugbúnaður sem tölva notar til að hafa umsjón með gagnagrunnum. Dæmi í þessari bók miðast við að gagnagrunnskerfið MySQL sé notað.

<sup>2</sup> e. *database management system*

Yfirlit yfir nokkur gagnagrunnskerfi sem mikið eru notuð má finna í kafla 8.

#### *Hvernig vinnum við með gagnagrunnskerfi?*

MySQL gagnagrunnskerfi skiptast í *client* og *server*<sup>3</sup>. Server sér um úrvinnslu og meðhöndlun gagna. Client tengist servernum og veitir notandanum aðgang að gagnagrunninum.

<sup>3</sup> orðin *client* og *server* hafa verið þýdd sem “biðlari” og “miðlari” á Íslensku, en þau orð eru í takmarkaðri notkun

Til að keyra MySQL-server þarf að setja upp töluverða umgerð á viðkomandi tölvu. Dæmi um hugbúnaðarpakka sem heldur utan um MySQL-server er XAMPP<sup>4</sup>. Ekki verður farið sérstaklega yfir uppsetningu slíkrar umgjörðar hér, en hún er til staðar á vefþjóni<sup>5</sup> tölvudeildar Upplýsingatækniskólans.

<sup>4</sup><http://www.apachefriends.org/en/xampp.html>

<sup>5</sup><http://tsuts.tskoli.is/>

Hugbúnaðurinn sem notaður er í Tækniskólanum til að tengjast SQL-servernum er MySQL Workbench<sup>6</sup>. Sá hugbúnaður er þegar upp settur á sýndarvélum nemenda Upplýsingatækniskólans.

<sup>6</sup><http://www.mysql.com/products/workbench/>

### 1.5 Yfirlit

Í þessum kafla fórum við yfir eftirfarandi atriði:

- SQL-gagnagrunnur er samansafn af upplýsingum, skipulagt með töflum.
- SQL er fyrirspurnamál, notað til að eiga samskipti við gagnagrunnskerfi.

- Forrit geta tengst SQL-gagnagrunnum og notað þá til að sjá um gagnavinnslu. Þetta er gert í stórum stíl í tölvukerfum í dag.
- Gagnagrunnskerfi er hugbúnaður sem tölva notar til að hafa umsjón með gagnagrunnum. MySQL er dæmi um gagnagrunnskerfi.
- Í þessum áfanga verður gagnagrunnskerfið MySQL notað. Forritið MySQL Workbench verður notað til að tengjast MySQL-server tölvudeildarinnar.

## 2

# Fyrstu skrefin í SQL

SQL er nokkuð sveigjanlegt og yfirgripsmikið mál. Í þessari bók er einungis farið yfir lítið brot af því sem viðfangsefnið hefur upp á að bjóða.

Byrjum á að skoða grundvallaraðgerðirnar í SQL - að búa gagnagrunn, búa til töflur, setja gögn í töflurnar og að skoða gögnin aftur.

### 2.1 Gagnagrunnar

Fram kom í kafla 1 að gagnagrunnur sé *skipulagt samansafn af upplýsingum*. Einnig kom fram að upplýsingarnar eru hlutaðar niður með “töflum”.

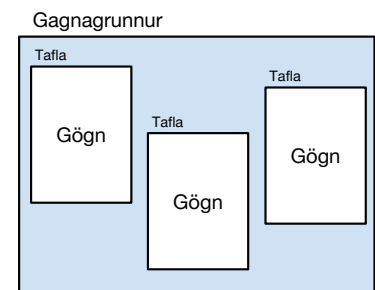
Hér er vert að taka nokkur atriði fram:

- Gagnagrunnur er ekki það sama og tölva. Talað er um að gagnagrunnur sé geymdur á tölvu eða jafnvel að tölva keyri gagnagrunns-server. Ein tölva getur geymt marga gagnagrunna.
- Gagnagrunnur er ekki forrit. Gagnagrunnskerfi<sup>1</sup>, sem er forrit, keyrir á tölvunni og heldur utan um gagnagrunninn. En gagnagrunnurinn sjálfur er ekki forrit.

Líkja mætti gagnagrunni við “möppu” í tölvu sem inniheldur töflur<sup>2</sup>. Hægt er að búa til gagnagrunna á tölvu, setja í hann töflur og skoða innihald þeirra.

Við sjáum dæmi um hvernig búa má til gagnagrunn í undirkafla 2.4.

Mynd 2.1: Uppbygging einfalds gagnagrunns með þremur töflum.



<sup>1</sup> Sjá undirkafla 1.4 og 8.2

<sup>2</sup> Reyndar getur gagnagrunnur innihaldið ýmislegt annað en töflur. Slík atriði eru tekin fyrir í seinni áföngum.

## 2.2 Töflur

Gögn í SQL-gagnagrunni má líta á sem raðir í töflum. Þess vegna er mikilvægt skref í því að læra að nota SQL að vera það að skilja uppbyggingu taflna mjög nákvæmlega. Lítum fyrst á dæmigerða töflu.

Nafn	Starfsheiti	Netfang
Bjargey G. Gísladóttir	Skólastjóri	bbg@tskoli.is
Donatas Butkus	Tölvuþjónusta	db@tskoli.is
Eiríkur Ernir Þorsteinsson	Kennari	eet@tskoli.is
Emil Gautur Emilsson	Kennari	ege@tskoli.is
Geir Sigurðsson	Kennari	ges@tskoli.is
Gunnar Þórunnarson	Kennari	gus@tskoli.is
Guðmundur Jón Guðjónsson	Kennari	gig@tskoli.is
Guðrún Randalín Lárusdóttir	Kennari	grl@tskoli.is
Hallur Ó. Karlsson	Kennari	hal@tskoli.is
Konráð Guðmundsson	Kennari	kng@tskoli.is
Matthias Skúlason	Tölvuþjónusta	matti@tskoli.is
Sigurður R. Ragnarsson	Kennari	srr@tskoli.is
Snorri Emilsson	Kennari	sem@tskoli.is
Tryggvi Jóhannsson	Kerfisstjóri	tj@tskoli.is
Þórarinn J. Kristjánsson	Kennari	tjk@tskoli.is

Tafla 2.1: Nokkrir starfsmenn Tækniskólans.

Eins og allar alvöru töflur inniheldur þessi starfsmannatafla annars vegar *dálkheiti* og hins vegar *gögn*. Dálkheitin eru “Nafn”, “Starfsheiti” og “Netfang”. Dæmi um upplýsingar eru að til sé starfsmaður sem heitir “Eiríkur Ernir Þorsteinsson”, sem er “Kennari” og hefur netfangið “eet@tskoli.is”.

Mikilvægt er að átta sig á þessum mun - hver einasta tafla sem unnið er með inniheldur dálkheiti og gögn, sem eru aðskilin fyrirbrigði. Þetta á við “hefðbundnar” töflur sem við sjáum á prenti og við töflur í forritum á borð við Microsoft Excel.

En þetta á líka við töflur sem við skilgreinum með SQL-skipunum. SQL-töflur innihalda dálkheiti og gögn, alveg eins og við myndum búast við af hefðbundnum töflum.

Þegar töflur eru sýndar á prenti er venjan að dálkheitin komi fram í fyrstu línu töflunnar (og oftast aðskilin gögnunum með striki). Gögnin koma fram í næstu línunum.

Þegar SQL er notað til að lýsa töflum eru dálkheitin og aðrar upplýsingar sem skilgreina töfluna sjálfa búnar til með sérstökum skipunum. Aðrar skipanir eru notaðar til að vinna með gögnin sjálf. Við sjáum dæmi um þessar skipanir í undirkafla 2.4.<sup>3</sup>

<sup>3</sup> Farið í muninn á skipunum sem skilgreina töflur og skipunum sem vinna með gögn í kafla 7.



## 2.3 Fyrirspurnir

Ekki er mikið gagn í því að geyma upplýsingar í töfluformi nema að hægt sé að ná í þær aftur.

Einfalt er að fletta upp upplýsingum í litlum töflum á borð við töflu 2.1 þegar þær eru prentaðar út. Viljum við t.d. komast að því hver er með netfangið “kng@tskoli.is” dugar okkur að láta augun reika yfir töfluna þar til við rekumst á netfangið og líta svo í starfsmannadálkinn.

Væri taflan örlítið stærri væri verkefnið strax erfiðara. Væri taflan á stærð við símaskrána væri það nær ómögulegt.

Slíkar uppflettingar, stórar og smáar, eru sérsvið SQL. Þær eru nefndar *fyrirspurnir* og eru framkvæmdar með mjög mikilvægri SQL-skipun sem heitir *SELECT*. Við sjáum dæmi um slíka skipun í næsta undirkafla (2.4) og kynnumst þeim náið í kafla 4.

## 2.4 Sýnidæmi í SQL

Skoðum hvernig búa má til töflu 2.1 með SQL, setja í hana gögn og sækja gögnin aftur.

Eins og fram hefur komið þarf að nota SQL-skipanir til þess.

### Skilgreining gagnagrunns

Okkar fyrsta verk verður að vera að búa til gagnagrunn til að setja töfluna í. Slíka skipun má sjá á sýnidæmi 2.1.

---

```
CREATE DATABASE taekniskolinn;
```

---

Sýnidæmi 2.1: CREATE DATABASE skipun fyrir Tækniskólagagnagrunninn.

Skipunin er ein lína. Hér vill reyndar svo til að hún er nálægt því að vera málfræðilega rétt setning á ensku. Hún er einfaldlega lýsing á því sem við viljum að gagnagrunnskerfið geri fyrir okkur: “búðu til gagnagrunn með nafnið taekniskolinn”!

Þetta er mjög einkennandi fyrir SQL. Í SQL erum við nær alltaf að lýsa því *hvað* við viljum gera frekar en *hvernig* við gerum það. Þetta gerir SQL frábrugðið flestum vinsælum forritunarmálum.

*Skilgreining töflu og innsetning gagna*

Skipunina til að skilgreina töfluna má sjá á SQL-sýnidæmi 2.2. Þetta er mjög dæmigerð skipun til að búa til töflu. Þar kemur fram hvað gera skal (búa til töflu → CREATE TABLE) og hver dálkheitin eru (nafn, Starfsheiti og netfang).<sup>4</sup>

<sup>4</sup> Við skulum ekki örvænta þó að skipanirnar séu óskiljanlegar á þessum tímapunkti. Við skoðum skipanirnar vandlega í næsta kafla, þessar eru einungis svo að við getum fengið tilfinningu fyrir því hvernig þær geta litið út.

---

```
CREATE TABLE Starfsmenn
(
  nafn VARCHAR(50),
  starfsheiti VARCHAR(20),
  netfang VARCHAR(20),
  id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT
);
```

---

Sýnidæmi 2.2: CREATE TABLE skipun fyrir starfsmannatöfluna.

Hér ber að athuga að enn eru engin gögn komin inn í töfluna. Það má gera með skipuninni í SQL-sýnidæmi 2.3.

---

```
INSERT INTO
  Starfsmenn(nafn, starfsheiti, netfang)
VALUES
  ('Bjargey G. Gísladóttir', 'Skólastjóri', 'bgg@tskoli.is'),
  ('Donatas Butkus', 'Tölvuþjónusta', 'db@tskoli.is'),
  ('Eiríkur Benediktsson', 'Kennari', 'ebe@tskoli.is'),
  ('Eiríkur Ernir Þorsteinsson', 'Kennari', 'eet@tskoli.is'),
  ('Emil Gautur Emilsson', 'Kennari', 'ege@tskoli.is'),
  ('Geir Sigurðsson', 'Kennari', 'ges@tskoli.is'),
  ('Guðmundur Jón Guðjónsson', 'Kennari', 'ggj@tskoli.is'),
  ('Guðrún Randalín Lárusdóttir', 'Kennari', 'grl@tskoli.is'),
  ('Gunnar Sigurðsson', 'Kennari', 'gus@tskoli.is'),
  ('Hallur Ó. Karlsson', 'Kennari', 'hal@tskoli.is'),
  ('Konráð Guðmundsson', 'Kennari', 'kng@tskoli.is'),
  ('Matthias Skúlason', 'Tölvuþjónusta', 'matti@tskoli.is'),
  ('Sigurður R. Ragnarsson', 'Kennari', 'srr@tskoli.is'),
  ('Snorri Emilsson', 'Kennari', 'sem@tskoli.is'),
  ('Tryggvi Jóhannsson', 'Kerfisstjóri', 'tj@tskoli.is'),
  ('Þórarinn J. Kristjánsson', 'Kennari', 'tjk@tskoli.is');
```

---

Sýnidæmi 2.3: INSERT skipun fyrir starfsmannatöfluna.

*Fyrirspurnir*

Til að sækja gögn úr töflunni þarf síðan að framkvæma fyrirspurn. Dæmi um fyrirspurn (SQL-skipun) sem finnur nafn þess starfsmanns sem er með netfangið "kng@tskoli.is" má sjá á sýnidæmi 2.4.

---

```
SELECT nafn
FROM Starfsmenn
WHERE netfang = 'kng@tskoli.is';
```

---

Athugum hvað kemur fram í fyrirspurninni. Við getum séð þrjá aðalhluta - lýsingu á því hvað gera skal (finna nafn/SELECT nafn), hvaðan upplýsingarnar skulu koma (úr starfsmannatöflunni/FROM Starfsmenn) og hvaða skilyrði gilda um gögnin sem finna skal (gögnin þar sem netfangið er kng@tskoli.is/WHERE netfang = "kng@tskoli.is").

## 2.5 Keyrsla í MySQL Workbench

Lítum örstutt á hvernig nota má MySQL Workbench til að búa til okkar fyrstu töflu með SQL.

Um nokkur skref er að ræða.

1. Ræsa þarf forritið. Við það birtist upphafsskjár, líkur þeim sem sjá má á mynd 2.2.<sup>5</sup>
2. Mynda þarf tengingu við MySQL-server. Henni þarf að gefa nafn, IP-tölu MySQL serversins, notandanafn og lykilorð.<sup>6</sup> Sjá mynd 2.3.
3. Búa þarf til nýjan gagnagrunn á MySQL-servernum á skjánum sem birtist eftir að tengingin er mynduð. Það er gert með CREATE DATABASE skipun, sjá mynd 2.4. Þegar þessu er lokið höfum við keyrt okkar fyrstu SQL-skipun!
4. Skipunin í sýnidæmi 2.2 er slegin inn í aðalgluggann og keyrð. Taflan er þá komin inn.

Ávallt má gera ráð fyrir að SQL-sýnidæmi í þessari bók megi keyra með hliðstæðum hætti - í aðalglugga MySQL Workbench. T.d. mætti keyra sýnidæmi 2.3 og 2.4 á þann hátt.

Sýnidæmi 2.4: SELECT skipun sem finnur nafnið á þeim kennara sem er með netfangið kng@tskoli.is. Það nafn er "Konráð".

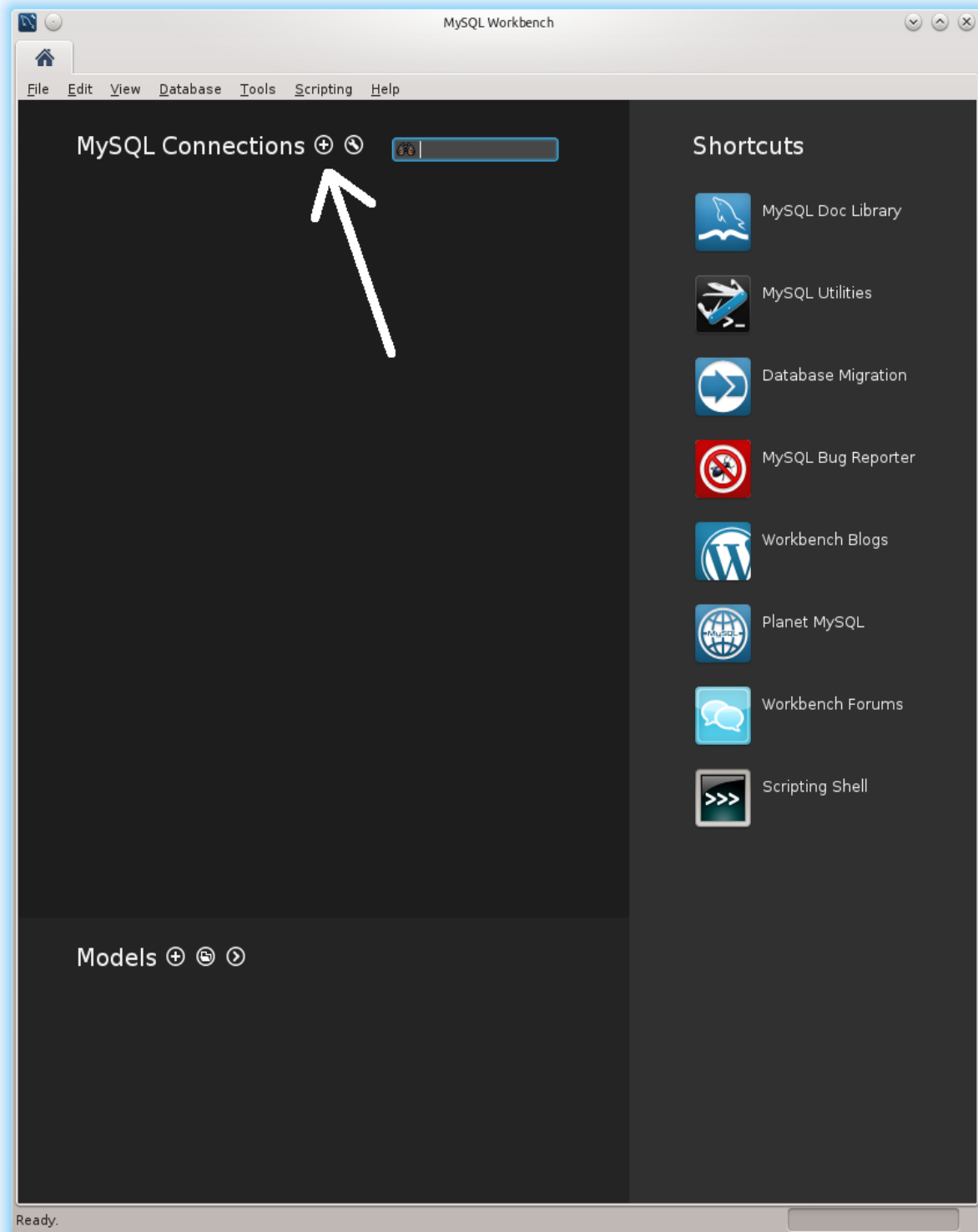
Við getum spurt okkur af hverju öll SQL-sýnidæmin eru með sum orðin í hástöfum og af hverju svona fá orð eru í hverri línu. Þetta er vegna þess að

1. Löng hefð er fyrir því að skrifa SQL-lykilorð í hástöfum.
2. Línubilin eru á þeim stöðum sem höfundur þykir hjálpa mest upp á læsileika.

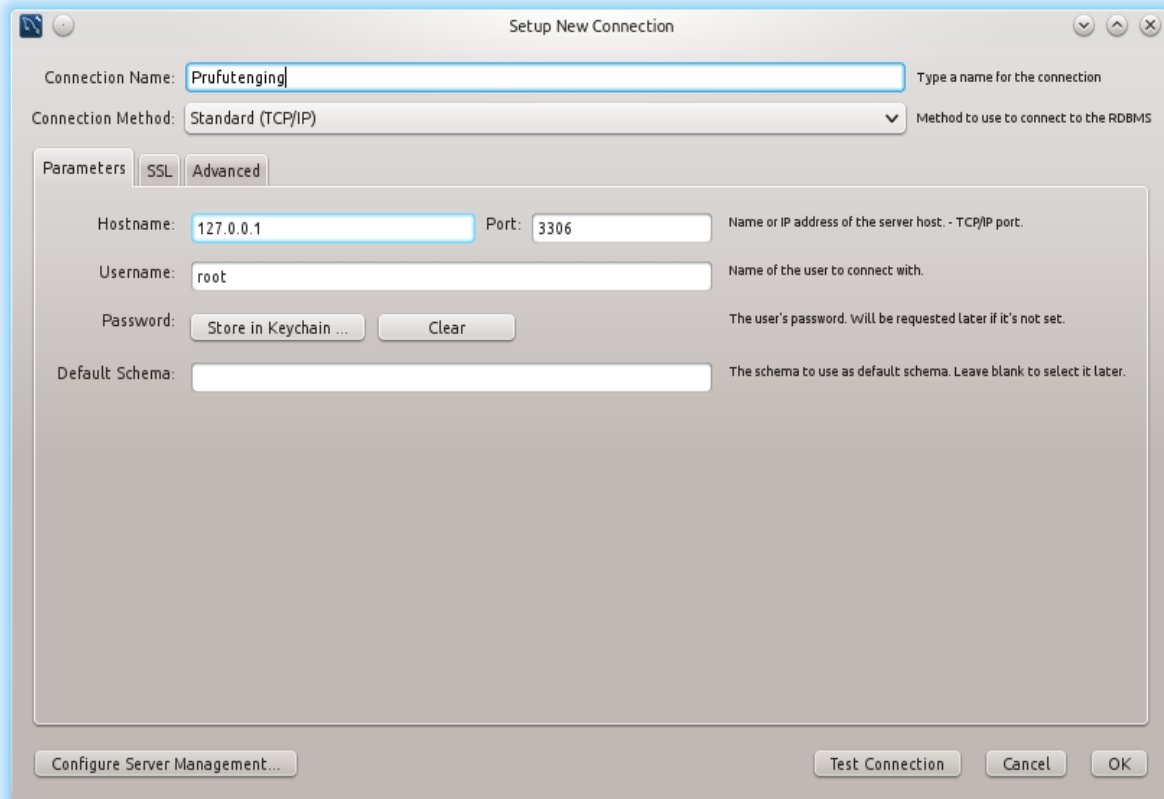
Hvorugt er strangt til tekið nauðsynlegt.

<sup>5</sup> Útlit MySQL Workbench er eðli málsins samkvæmt örlítið mismunandi eftir stýrikerfum og útgáfum á forritinu. Skjáskotin eru tekin af Workbench útgáfu 6.0, á Linux vél.

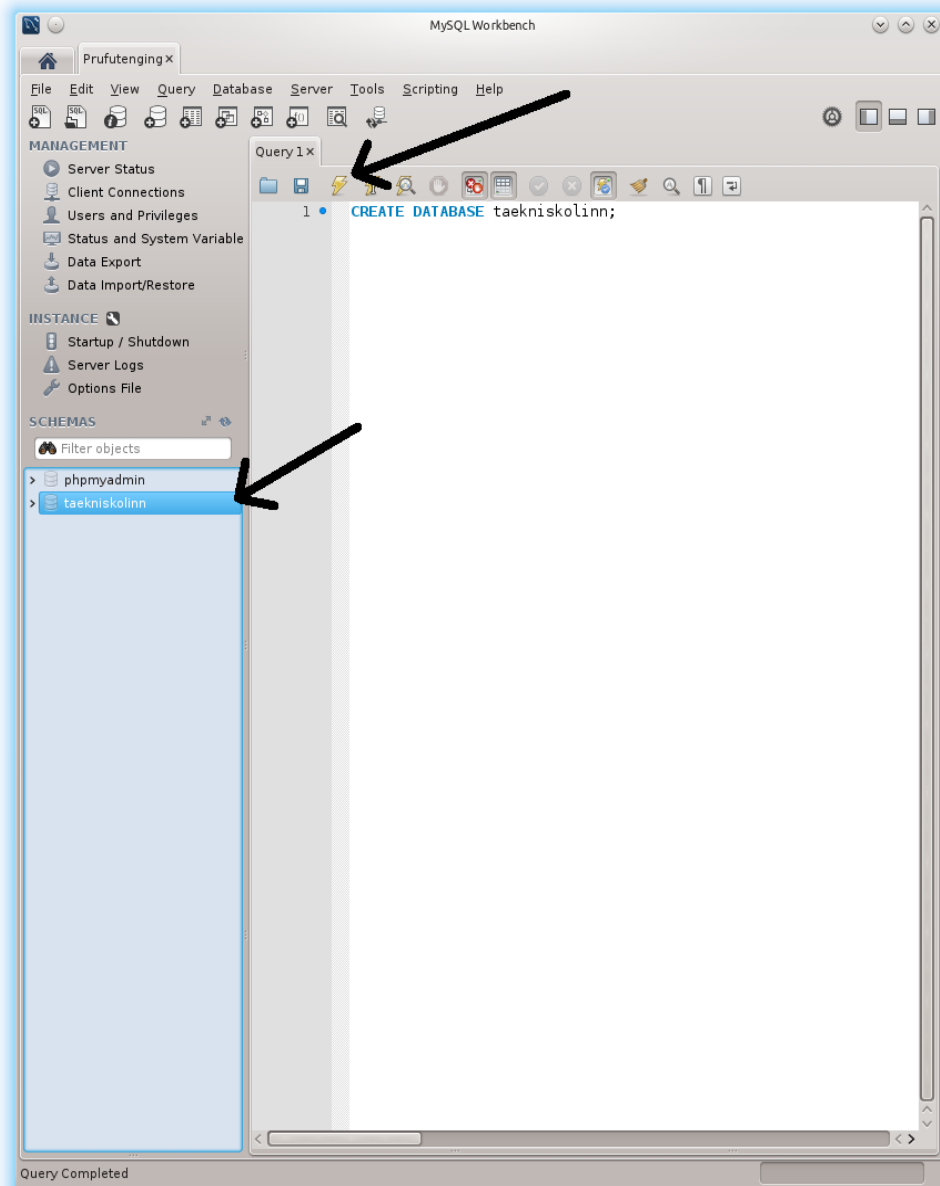
<sup>6</sup> Nemendur Tölvudeildar Tækniskólans skulu biðja kennarann um þessar upplýsingar.



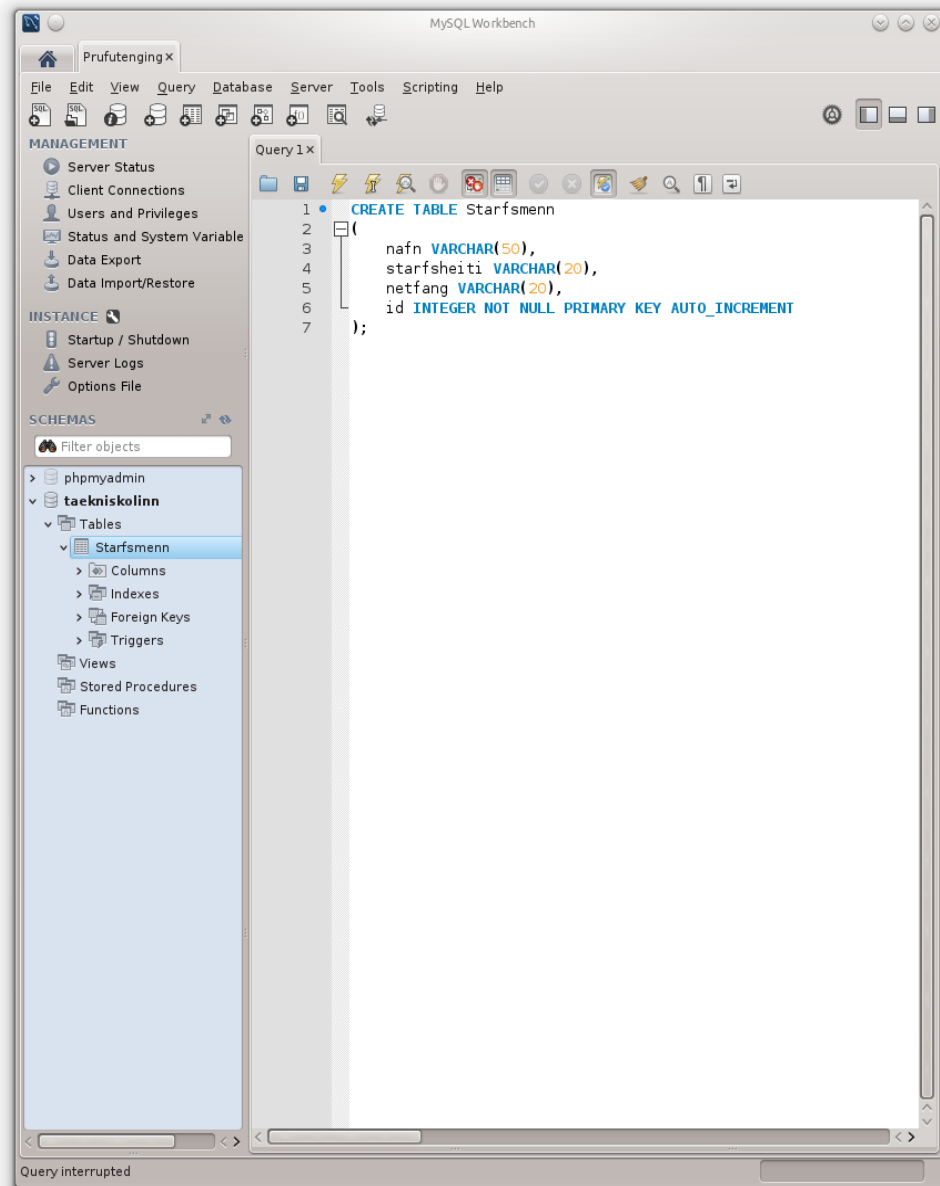
Mynd 2.2: Upphafsskjár MySQL Workbench. Örin vísar á hnapp sem nota má til að búa til nýja tengingu.



Mynd 2.3: Tengingarskjár MySQL Workbench. Hér er verið að búa til tenginguna "Prufutenging", sem tengist MySQL-þjóni sem keyrir á sömu tölvu og Workbench-inn (127.0.0.1) með notandanafninu "root".



Mynd 2.4: Nýr gagnagrunnur búinn til með MySQL Workbench. Efri örin vísar á hnappinn sem ýta þarf á til að keyra SQL-skipunina. Neðri örin vísar á lista af gagnagrunnum sem sýnilegir eru á servernum. Birtist gagnagrunnurinn sem búinn er til ekki um leið og skipunin er keyrð, hægri-smellið þá á listann og “refresh”ið hann.



Mynd 2.5: Ný tafla búin til með MySQL Workbench. Skipunin er slegin inn í aðalgluggann og keyrð með “eldingarhnaðnum” eins og skipunin á mynd 2.4.

## 2.6 Yfirlit

Í þessum kafla fengum við örstutta kynningu því hvernig SQL-gagnagrunnar eru uppbyggðir og hvernig við getum átt samskipti við þá.

Helstu atriðin eru:

- Gögn í SQL-gagnagrunni má líta á sem línur í töflum.
- SQL-skipanir eru notaðir til að skilgreina töflur og setja gögn í þær.
- Fyrirspurnir eru notaðar til að sækja gögn úr gagnagrunnum. Fyrirspurnir eru ákveðin gerð SQL-skipana.
- SQL-skipanir má keyra úr aðalglugga MySQL Workbench.

Athugum að við höfum ekki farið yfir uppbyggingu skipananna. Það að læra á skipanirnar sjálfar er viðfangsefni næstu kafla.



## 3

# Uppsetning taflna

Í kafla 2.4 sáum við dæmi um hvernig búa má til töflu með SQL-skipun. Hins vegar eyddum við ekki sérstaklega miklum tíma í að reyna að skilja hvernig skipunin er uppbyggð, hvað öll lykilorðin sem fram komu þýddu eða hvað er leyfilegt. Viðfangsefni kaflans sem við erum stödd í núna verður að leiðrétta þennan trassaskap og sökkva okkur í töflugerð.

### 3.1 Að búa til töflu

Athugum nú hversu einfalda töflu við getum búið til. Hún gæti verið á þá leið sem sjá má á sýnidæmi 3.1.

---

```
CREATE TABLE NafnToflu
(
  nafnDalks INTEGER
);
```

---

Sýnidæmi 3.1: Mjög einföld tafla. Athugum að “NafnToflu” og “nafnDalks” er ekki hluti af SQL-málinu, heldur bara dæmi um hvernig heiti á töflum og dálkum eru skilgreind.

Skoðum þessa skipun nú mjög vandlega.

- Hún hefst á að lýsa yfir hvað gera skal - hér er það CREATE. Við ætlum að búa til töflu, svo við segjum TABLE. Næst kemur nafn töflunnar fram.
- Þegar nafn töflunnar hefur verið gefið opnast svigi.
- Inni í sviganum kemur nafnið á dálki og orðið INTEGER<sup>1</sup>. Í þessari töflu er einungis einn dálkur.
- Skipuninni lýkur á því að sviganum er lokað og semíkomma (;) sett á eftir.

<sup>1</sup> Merkingin á þessum lykilorðum sem við höfum séð á eftir dálkheitunum, *INTEGER* og *VARCHAR*, útskýrist í næsta undirkafla (3.3).

Skoðum næst sýnidæmi 3.2, sem er örlítið stærra.

---

```
CREATE TABLE NafnAnnarrarToflu
(
  nafnFyrstaDalks INTEGER,
  nafnAnnarsDalks INTEGER
);
```

---

Hér sjáum við ágætlega hvernig bæta má við öðrum dálki. Fyrsta dálklýsingin er skrifuð, síðan kemur komma, síðan fylgir næsta dálklýsing.

Athugum að engin komma er á eftir síðustu dálklýsingunni. Það er vegna þess að dálkarnir eru einungis *aðskildir* með kommu, komman er ekki hluti af dálklýsingunni sjálfri.

### 3.2 Innsetning gagna

Þegar töflur eru búnar til með CREATE skipun eru þær tómar. Til að fylla töflu með gögnum þarf að nota aðra skipun - INSERT. Til að setja töluna 1 inn í töfluna sem við bjuggum til með sýnidæmi 3.1 mætti nota INSERT skipunina í sýnidæmi 3.3.

---

```
INSERT INTO
  NafnToflu(nafnDalks)
VALUES
  (1);
```

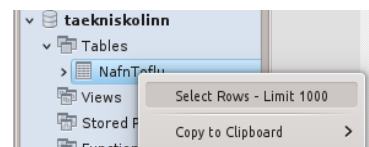
---

Skoðum þessa skipun í smáatriðum líka.

- Aftur hefst skipunin á því að lýsa því yfir hvað gera skal. Hér ætlum við að setja inn gögn - INSERT. Við ætlum að setja gögnin inn í eitthvað, svo við segjum INSERT INTO.
- Næst kemur nafn töflunnar sem gögnin eiga að fara í.
- Svigi opnast, nafn dálksins sem gögnin eiga að fara inn í er skrifað og sviginn lokast.<sup>2</sup>
- Að lokum segjum við hvað við ætlum að setja inn. Það eru gögn (eða “gildi”, VALUES). Gögnin sem við ætlum að setja inn er ein lína, sem við afmörkum með svigum. Hér samanstendur línan af einni tölu. Sem fyrr lokum við skipuninni með semíkommu.

Sýnidæmi 3.2: Einföld tafla

Mynd 3.1: Er erfitt að vita hvort að skipunin tókst eða ekki? Hægt er að hægri-smella á nafn töflu í MySQL workbench og biðja um “select rows”. Þetta birtir fyrstu þúsund línur sem eru í viðkomandi töflu - sem dugar oftast til að fá yfirlit.



Sýnidæmi 3.3: INSERT í einfalda töflu. Þessi skipun setur töluna 1 inn dálkinn *nafnDalks*. Niðurstaðan er tafla 3.1.

<sup>2</sup> Strangt til tekið er hægt að sleppa þessum sviga. Gagnagrunnskerfið reynir þá að setja gögnin inn í þá dálka sem það finnur í töflunni. Þessar ágískanir geta verið til vandræða, betra er að venja sig á að taka dálkheitin alltaf fram.

nafnDalks
1

Mikilvægt er að vita að INSERT skipun setur alltaf inn *heila línu* af gögnum í töfluna. Sé taflan með fleiri en einn dálk<sup>3</sup> þarf að skilgreina öll gögnin í einu. Sýnidæmi 3.4 er dæmi um slíkt.

#### INSERT INTO

```
NafnAnnarrarToflu(nafnFyrstaDalks, nafnAnnarsDalks)
```

#### VALUES

```
(1, 2);
```

Tafla 3.1: Niðurstaðan eftir að sýnidæmi 3.1 og 3.3 hafa verið keyrð - ofurlítil tafla með einum dálki og einni línu af gögnum. Flestar ölvörur töflu hafa fleiri en einn dálk.

Sýnidæmi 3.4: INSERT í tvo dálka í einu. Þessi skipun setur tölurnar 1 og 2 inn í sömu línu. Niðurstaðan er tafla 3.2.

nafnFyrstaDalks	nafnAnnarsDalks
1	2

Tafla 3.2: Niðurstaðan eftir að sýnidæmi 3.2 og 3.4 hafa verið keyrð. Tafla með tveimur dálkum og einni línu af gögnum.

Til að skrifa meira en eina línu af gögnum inn í gagnagrunn má keyra INSERT skipun oftari en einu sinni. Væri skipunin í sýnidæmi 3.2 keyrð tvisvar myndi línan sem hún lýsir vera tvítekin í töflunni.

Það að skrifa alla INSERT INTO romsuna upp upp á nýtt fyrir hverja línu sem setja skal inn getur verið þreytandi. Þess vegna býður MySQL upp á leið til að setja inn margar línur í einu. Við sjáum dæmi um það í sýnidæmi 3.5.

#### INSERT INTO

```
NafnAnnarrarToflu(nafnFyrstaDalks, nafnAnnarsDalks)
```

#### VALUES

```
(1, 2),
```

```
(3, 4);
```

Sýnidæmi 3.5: INSERT í tvo dálka í einu. Þessi skipun setur tölurnar 1 og 2 inn í eina línu og tölurnar 3 og 4 í þá næstu. Niðurstaðan er tafla 3.3.

nafnFyrstaDalks	nafnAnnarsDalks
1	2
3	4

Tafla 3.3: Niðurstaðan eftir að sýnidæmi 3.2 og 3.5 hafa verið keyrð. Tafla með tveimur dálkum og tveimur línum af gögnum.

### 3.3 Algengar gagnagerðir: Tölur og texti

Einni stórrí spurningu um `CREATE TABLE` dæmin framar í þessum kafla er enn ósvarað - hvað er þetta `INTEGER`?

Integer er dæmi um svokallaða gagnagerð<sup>4</sup>. Nánar til tekið er þetta gagnagerð sem táknar heiltölur<sup>5</sup> Þegar `INTEGER` kemur fyrir í dálkskilgreiningu erum við sem sagt að segja gagnagrunnskerfinu að við munum einungis geyma heiltölur í þessum dálki.

Skoðum nokkrar helstu gagnagerðir í MySQL og hvenær við notum þær.

#### Heiltölur - `INTEGER`

Fram hefur komið að `INTEGER` dálkur geymi heiltölur. Hann tekur ekki við kommutölum.<sup>6</sup>

Lægsta talan sem slíkur dálkur getur geymt er  $-2147483648$  og sú hæsta  $2147483647$ . Ástæðan fyrir því að lægri og hærri tölur en þessar valda villum er sú að MySQL notar einungis fjögur bæti til að geyma hverja heiltölu - hærri og lægri tölur komast ekki fyrir í svo litlu geymsluplássi.

Sé nauðsynlegt að geyma tölur sem ekki passa inn á þetta bil má nota gagnagerðina `BIGINT` í stað `INTEGER`. Slíkur dálkur geymir einnig heiltölur, en hefur átta bæti til að geyma hverja tölu. `BIGINT` dálkur getur því geymt mun stærri (eða "lengri") tölur.

Sé vitað að tölurnar sem fara inn í dálkinn séu allar mjög litlar um sig má nota gagnagerðirnar `TINYINT`, `SMALLINT` og `MEDIUMINT` í MySQL. Þær taka hver um sig 1, 2 og 3 bæti til að geyma hverja tölu. Sjaldnast er ástæða til að nota þessar gagnagerðir í dag. Tafla þyrfti að innihalda tugi milljóna raða<sup>7</sup> til að plásssparnaðurinn gæti skipt máli á nútíma tölvukerfum.

Nafn	Stærð í bætum	Lægsta gildi	Hæsta gildi
<code>TINYINT</code>	1	$-128$	127
<code>SMALLINT</code>	2	$-32768$	32767
<code>MEDIUMINT</code>	3	$-8388608$	8388607
<code>INT</code>	4	$-2147483648$	2147483647
<code>BIGINT</code>	8	$-9223372036854775808$	9223372036854775807

Ef sérstök skilyrði koma ekki upp er best og einfaldast að halda sig við `INTEGER` til að geyma heiltölur. Þetta getur sérstaklega borgað sig

<sup>4</sup> e. *data type*

<sup>5</sup> Tölur 1 og  $-256$  og allar tölur sem við höfum sett inn í töflur framar í bókinni eru heiltölur. Tölur eins og 1,1 eru ekki heiltölur, heldur kommutölur.

<sup>6</sup> Í stað `INTEGER` má skrifa styttinguna `INT`, sem hefur sömu áhrif.

<sup>7</sup> Reyndar eru í dag til gagnagrunnstöflur sem innihalda *hundruðir milljarða* raða.

Tafla 3.4: Heiltölugagnagerðir í MySQL og stærðir þeirra.

ef seinna reynist nauðsynlegt að færa gagnagrunninn á milli gagnagrunnskerfa - sum gagnagrunnskerfi styðja ekki allar gagnagerðirnar sem MySQL býður upp á. Gamla góða INTEGER er hins vegar alltaf til staðar.

ÍSLENSKAR KENNITÖLUR eru varhugaverðar. Freistandi getur verið að geyma kennitölur í heiltöludálkum. Þær eru jú kommulausar tölur, ekki satt? Nokkur atriði leiða þó til þess að þær passa ekki mjög vel í slíka dálka.

- Stærsta tala sem hægt er að geyma í venjulegum INTEGER dálki er 2147483647. Það þýðir að kennitölur allra sem fæddir eru á 22. degi mánaðar eða seinna komast ekki fyrir í dálkinum! Það þyrfti BIGINT.
- Kennitölur eru ekki notaðar eins og flestar tölur. Þær eru t.d. ekki lagðar saman eða bornar saman með < og > virkjum. Við þurfum oftast að skoða ákveðna stafi í kennitölunni (t.d. fimmta og sjötta stafinn til að komast að fæðingarári) frekar en stærð tölunnar sjálfrar.
- Kennitölur eru venjulega skrifaðar með bandstrikum, sem eiga ekki heima í heiltöludálki.

Vænlegra er að nota CHAR dálk til að geyma kennitölur.

### Texti - VARCHAR og CHAR

Í forritun lítum við oftast á texta sem safn af stöfum, SQL er engin undantekning. Slíkt safn er kallað strengur<sup>8</sup>. 'Ari', 'Ari sá sól' og 'Ari á 10 krónur' eru allt dæmi um strengi.

<sup>8</sup> e. *string*

Eins og sjá má eru strengir afmarkaðir með gæsalöppum. Mikilvægt er að gleyma þeim ekki þegar strengir eru slegnir inn (sjá kafla 3.2). Gæsalappir eru aðal leiðin sem gagnagrunnskerfið hefur til að vita hvort að um streng eða dálkheiti sé að ræða.

Venja er að nota einfaldar gæsalappir (') í SQL til að afmarka strengi. Tvöfaldar gæsalappir (") virka líka í MySQL.

Talað er um að strengur hafi ákveðna *lengd*. Lengd strengs er fjöldi stafa í strengnum, að bilum meðtöldum. Gæsalappirnar eru ekki talðar með, þær umlykja strenginn en eru ekki hluti af honum. Þannig er 'Ari' strengur af lengd 3, 'Ari sá sól' er strengur af lengd 10.

Strengir geta innihaldið næstum hvaða stafi sem er, líka tölustafi og íslenska stafi. 'Ari á 10 krónur' er löglegur strengur. Strengur getur meira að segja innihaldið ekkert nema tölustafi. "10" er strengur, 10 er heiltala.

TIL AÐ GEYMA TEXTA höfum við nokkrar gagnagerðir, líkt og við höfum nokkrar gagnagerðir til að geyma heiltölur. Þær helstu eru CHAR og VARCHAR. “Char” stendur hér fyrir enska orðið “character”, sem þýða má sem “stafur”. “Var” í VARCHAR stendur fyrir “variable”.

Þegar CHAR og VARCHAR gagnagerðirnar eru notaðar þurfum við að taka fram hversu langa strengi dálkurinn á að geta tekið við<sup>9</sup>. Þetta er gert með því að setja hámarkslengdina inn í sviga. CHAR(3) er dálkur sem tekur við strengjum af lengd 3.

Sé strengur sem er of langur settur inn í CHAR eða VARCHAR dálk er klippt af hægri enda hans svo að hann passi. Væri t.d. reynt að geyma 'Ari sá sól' í CHAR(5) dálk væri útkoman 'Ari s'.

Munurinn á CHAR og VARCHAR kemur fyrst og fremst fram þegar of stuttir strengir eru settir inn í dálkinn.

- Sé strengur styttri en breidd CHAR dálks sem hann er settur inn í er bilum bætt við hægri enda strengsins svo hann passi nákvæmlega. Þannig yrði strengurinn 'Ari sá sól' að 'Ari sá sól ' væri hann settur inn í CHAR(15) dálk.
- Sé strengur sem er of stuttur settur inn í VARCHAR dálk er hann einfaldlega geymdur eins og hann er. VARCHAR dálkur getur geymt “of stutta” strengi.

CHAR dálkur getur verið að hámarki 255 stafir að breidd. VARCHAR dálkur getur verið allt að 65.535 stafa breiddur.

Hljómi VARCHAR hér eins og almennt gagnlegri gagnagerð er það líklega vegna þess að það er rétt. VARCHAR eyðir minna plássi og breytir strengjum ekki að óþörfu. CHAR dálkar eru gagnlegir þegar vitað er fyrirfram að allir strengir sem eiga að fara inn í dálkinn séu af sömu lengd (t.d. kennitölur) eða ef seinni tíma vinnsla krefst þess að gögnin séu sjálfum sér mjög samkvæm. Leiki vafi á er einfaldast að nota bara VARCHAR.

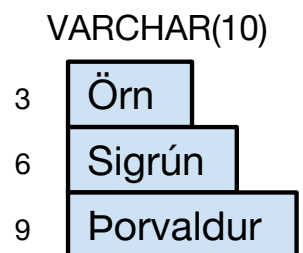
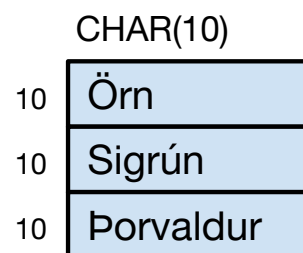
MJÖG LANGUR TEXTI getur verið erfiður í meðförum. Sé ætlunin t.d. að geyma bækur, blogg færslur eða fréttagreinar í gagnagrunni, hversu stóran VARCHAR dálk þyrfti til að halda utan um textann? Svarið er: Of stóran.

MySQL býður upp á aðrar gagnagerðir sem eru meira viðeigandi fyrir slíka vinnslu - hér eru helstar TEXT og LONGTEXT. LONGTEXT dálkur er fær um að geyma strengi sem eru allt að 4 GiB að lengd.

Sé ætlunin að geyma upplýsingar sem almennt eru af skynsamlegri stærð, t.d. mannanöfn, þá er hagkvæmara að nota VARCHAR dálk.

<sup>9</sup> Oft er talað um “breidd” á þessum dálkum. Til að strengur af lengd 5 komist inn í dálk þarf dálkurinn að vera 5 stafa breiddur.

Mynd 3.2: Hvernig CHAR og VARCHAR dálkar geyma texta. Talan til vinstri táknar lengd strengsins eins og hann er geymdur.



### 3.4 Dæmi um töflur með tölum og texta

Ýmsar töflur má búa til með tölum og texta. Fyrsta taflan sem við sáum, tafla 2.1, er dæmi um slíka töflu. Lítum á fleiri dæmi.

PLAYER	SCORE
EET	12000
HKS	11000
GUT	10000
HAI	9000
JOH	8000
JTH	7000
PHS	6000

Mynd 3.3: “High score” tafla fyrir tölvuleik. Fyrir nokkrum áratugum litu high score listar í spilakössum yfirleitt út á þennan hátt. Hver lína samanstóð af þremur upphafsstöfum og stiga-fjölda. Hana mætti búa til með skipuninni í sýnidæmi 3.6.

---

```
CREATE TABLE HighScores
(
  player CHAR(3),
  score INTEGER
);

INSERT INTO HighScores
  (player, score)
VALUES
  ('EET', 12000),
  ('HKS', 11000),
  ('GUT', 10000),
  ('HAI', 9000),
  ('JOH', 8000),
  ('JTH', 7000),
  ('PHS', 6000);
```

---

Sýnidæmi 3.6: High score taflan á mynd 3.3 búin til með SQL-skipun. Hér vitum við að spilararnir nota alltaf nákvæmlega þrjá upphafsstafi til að auðkenna sig, svo CHAR dálkur er viðeigandi. Stigin sjálf eru geymd í INTEGER dálki.

1.	<b>Hamborgari</b> 120gr. nautakjöt, kál, tómatar	<b>690 kr.</b>
2.	<b>Ostborgari</b> 120gr. nautakjöt, ostur, kál, tómatar	<b>750 kr.</b>
3.	<b>Beikonborgari</b> 120gr. nautakjöt, ostur, beikon, kál, tómatar	<b>890 kr.</b>
4.	<b>Sá stóri</b> 200gr. nautakjöt, ostur, beikon, kál, tómatar	<b>1180 kr.</b>

Mynd 3.4: Hamborgaramatseðill á veitingastað. Töflur geta litið út á ýmsan hátt. Matseðil af þessari gerð mætti geyma í gagnagrunni, þó að hann líti e.t.v. ekki út eins og tafla við fyrstu sýn. Töflu sem heldur utan um upplýsingarnar á honum má sjá á sýnidæmi [3.7](#).

---

```
CREATE TABLE Hamborgarar
```

```
(
  numer INTEGER,
  nafn VARCHAR(50),
  verd INTEGER,
  lysing VARCHAR(255)
);
```

```
INSERT INTO
```

```
Hamborgarar(numer, nafn, verd, lysing)
```

```
VALUES
```

```
(1, 'Hamborgari', 690,
  '120g nautakjöt, kál, tómatar'),
(2, 'Ostborgari', 750,
  '120g nautakjöt, ostur, kál, tómatar'),
(3, 'Beikonborgari', 890,
  '120g nautakjöt, ostur, beikon, kál, tómatar'),
(4, 'Sá stóri', 1180,
  '200g nautakjöt, ostur, beikon, kál, tómatar');
```

---

Sýnidæmi 3.7: SQL-framsetning á matseðlinum á mynd [3.4](#). Við gerum ráð fyrir að lýsingin á réttinum þurfi meira pláss en nafn hans.



### 3.5 Fleiri gagnagerðir

Þó að ýmislegt sé hægt að gera með einungis texta og heiltölum, þá býður MySQL upp á mun fleiri gagnagerðir. Lítum stuttlega á nokkrar af þeim.

#### Tugabrot - DECIMAL

Til að geyma tugabrot (kommutölur, t.d. 1,5 og 5,2) dugur INTEGER dálkur ekki. Til þess getum við notað DECIMAL dálk.

Til að búa til DECIMAL dálk þurfum við að skilgreina tvær tölur. Sú fyrri er heildarfjöldi tölustafa sem mega vera í tölunni, sú seinni er fjöldi tölustafa “hægra megin” við kommu. Þannig myndi DECIMAL(5,2) dálkur passa akkúrat utan um töluna 123,45.

Gerð	Verð (kr.)
95 oktan	252,9
Dísel	242,3
Vélaolía	174,3
98 oktan	298,9

Tafla 3.5: Eldsneytisverð á bensínstöð. DECIMAL dálkur er notaður til að halda utan um bensínverðið með nákvæmlega einum aukastaf.

---

```
CREATE TABLE Eldsneyti
```

```
(
  gerð VARCHAR(20),
  verd DECIMAL(4,1)
);
```

```
INSERT INTO
```

```
Eldsneyti(gerð, verd)
```

```
VALUES
```

```
( '95 oktan', 252.9 ),
( 'Dísel', 242.3 ),
( 'Vélaolía', 174.3 ),
( '98 oktan', 298.9 );
```

---

Sýnidæmi 3.8: SQL-framsetning á eldsneytisverðinu í töflu 3.5. Verðið er geymt í dálki sem tekur við tugabroti með fjóra markverða stafi, þar af einum fyrir aftan kommu. Athugum að INSERT skipunin tekur við tölum á ensku formi, sem notar punkta þar sem kommur eru notaðar í íslensku (og öfugt). Væri reynt að setja tugabrotið inn með kommu væri það túlkað sem skipting á milli dálka!

#### Rauntölur - DOUBLE

Það að þurfa að taka fram stærð talna getur verið mjög takmarkandi. Hvað ef skali talnanna er mjög mismunandi eða ef geyma þarf gríðarlega “langar” tölur?



---

```
CREATE TABLE Afmaeli
```

```
(
  nafn VARCHAR(50),
  dagsetning DATE
);
```

```
INSERT INTO
```

```
Afmaeli(nafn, dagsetning)
```

```
VALUES
```

```
('Ernir', '1987-10-21');
```

---

Sýnidæmi 3.10: SQL-tafla sem táknar afmæli. Hér er dagsetningin 21. október árið 1987 sett inn í gagnagrunninn.

MySQL gerir ráð fyrir að fá dagsetningar á ákveðnu sniði. Það snið er: 'ÁÁÁÁ-MM-DD' þar sem ÁÁÁÁ stendur fyrir árið, MM fyrir mánuðinn og DD fyrir daginn. Þannig myndi '2008-07-01' tákna dagsetninguna 1. júlí árið 2008.<sup>11</sup>

Sem fyrr segir, þá eru dagsetningar meðhöndlaðar á allt annan hátt en strengir, þó þær líti svipað út. Oft geymir gagnagrunnskerfið dagsetningar sem heiltölur (en *sýnir* þær líkt og strengi). Þetta gerir það að verkum að t.d. er hægt að bera saman dagsetningar til að athuga hvor þeirra sé stærri.

MySQL hefur fleiri gagnagerðir sem meðhöndla tíma, t.d. TIME og TIMESTAMP. Við förum ekki sérstaklega yfir þær í þessari bók. Flestar hafa þær svipaða eiginleika og DATE.

<sup>11</sup> MySQL *getur* skilið dagsetningar á nokkrum öðrum sniðum. Betra er að halda sig við stöðluðu útgáfuna.

### 3.6 Tóm gildi

Hvað gerum við ef við þekkjum ekki allar upplýsingarnar sem tilheyra ákveðinni línu?

Sérstakt lykilorð er notað í SQL til að tákna það að ákveðið gildi sé óþekkt eða ekki til. Það lykilorð er NULL.

Mikilvægt er að rugla ekki saman NULL hugtakinu og tölunni 0. NULL þýðir að gildi sé óþekkt eða ekki til en talan 0 er alvöru tala. Ekki má heldur rugla því saman við strenginn 'NULL' eða strenginn ''. Fyrri strengurinn er einfaldlega orðið "null" geymt í gagnagrunni, sá seinni er alvöru strengur sem svo vill til að inniheldur enga stafi en er engu að síður þekkt gildi.

Hægt er að setja NULL beint inn í gagnagrunn með INSERT skipun. Slíkt getur verið viðeigandi þegar gildin eru ekki til. Sjá sýnidæmi [3.11](#).

---

```

CREATE TABLE Mannanofn
(
  eiginNafn1 VARCHAR(40),
  eiginNafn2 VARCHAR(40),
  milliNafn VARCHAR(40),
  kenniNafn VARCHAR(50)
);

INSERT INTO
  Mannanofn(eiginNafn1, eiginNafn2, milliNafn, kenniNafn)
VALUES
  ('Halldór', NULL, NULL, 'Ásgrímsson'),
  ('Geir', 'Hilmar', 'Haarde', NULL),
  ('Jóhanna', NULL, NULL, 'Sigurðardóttir'),
  ('Sigmundur', 'Davíð', NULL, 'Gunnlaugsson');

```

---

Sýnidæmi 3.11: Nokkur nöfn nýlegra forsætisráðherra sett sundurliðuð inn í gagnagrunn, með *NULL* gildum þar sem viðkomandi nafn er ekki til. T.d. er Sigmundur Davíð Gunnlaugsson ekki með millinafn, svo línan sem tilheyrir Sigmundi fær gildið *NULL* í þeim dálki.

Hingað til höfum við alltaf talið upp alla dálka hverrar töflu þegar *INSERT* skipun er notuð. Slíkt er þó ekki alltaf viðeigandi eða nauðsynlegt. Sé dálki sleppt í dálkaupptalningunni í *INSERT* skipun fær línan (eða línurnar) einfaldlega gildið *NULL* í þeim dálki.

### NOT NULL

Sumir dálkar eru þess eðlis að óviðeigandi eða órökrétt er að leyfa *NULL* gildi í þeim. Til þess að hindra það að *NULL* gildi séu sett inn í slíkan dálk er hægt að setja á hann skorðu<sup>12</sup> þess eðlis. Það má gera í *CREATE TABLE* skipuninni fyrir töfluna sem dálkurinn tilheyrir með því að bæta við lykilorðunum *NOT NULL* fyrir aftan skilgreininguna á gagnagerðinni. Sjá dæmi 3.12.

<sup>12</sup> e. *constraint*

---

```

CREATE TABLE Mannanofn
(
  eiginNafn1 VARCHAR(40) NOT NULL,
  eiginNafn2 VARCHAR(40) NULL,
  milliNafn VARCHAR(40) NULL,
  kenniNafn VARCHAR(50) NULL
);

```

---

Sýnidæmi 3.12: Tafla 3.11 endurtekin, en hér hefur sú ákvörðun verið tekin að allir skulu hafa a.m.k. eitt eiginnafn. Nú myndi gagnagrunnskerfið kvarta væri reynt að setja *NULL* gildi inn í fyrri eiginnafnsdálkinn.

Sé annað ekki tekið fram gerir MySQL ráð fyrir því að NULL sé leyfilegt í öllum dálkum. Hægt er að taka sérstaklega fram að NULL sé leyfilegt með því að skrifa NULL fyrir aftan skilgreiningu á gagnagerð dálks. Þetta hefur einnig verið gert í sýnidæmi [3.12](#).

ÞÓ AÐ NULL EIGI SINN SESS Í GAGNAGRUNNUM er það ekki endilega alltaf æskilegt. NULL getur valdið vandræðum við samanburð og talningu, sjá kafla [4](#). Einnig getur mikill fjöldi NULL dálka bent til þess að gagnagrunninum ætti að skipta upp í fleiri töflur, sjá kafla [5](#).

### 3.7 Aðallyklar - PRIMARY KEY

Lyklar eru mikilvægt atriði sem við höfum ekki enn skoðað.

Líta má á lykil<sup>13</sup> sem “efnisyfirlit” fyrir töflu. Þessi lykill gerir gagnagrunnskerfinu auðveldara að leita í töflunum sem honum tilheyrir.

<sup>13</sup> e. *key* eða *index*

Ein gerð af lyklum er svokallaður aðallykill<sup>14</sup>. Aðallykill hefur það hlutverk að einkenna hverja línu í gagnagrunninum fyrir sig, að vera nokkurs konar raðnúmer línunnar. Við höfum séð eitt dæmi um aðallykil í töflu fyrr í bókinni, í dæmi [2.2](#).

<sup>14</sup> e. *primary key*

Þar er það línan `id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT` sem býr til aðallykilinn. Skoðum þá línu aðeins betur:

- Fyrsti hluti línunnar, `id INTEGER`, segir okkur að um venjulegan heiltöludálg er að ræða. Hann heitir hér *id* (“auðkenni” á íslensku).
- Næst er okkur sagt að dálkurinn skuli vera `NOT NULL`. Þetta er mikilvægt vegna þess að við ætluðum að nota aðallykilinn sem raðnúmer sem getur auðkennt hverja einustu línu. Myndum við leyfa NULL gildi gætum við lent í því að fá tvö NULL gildi - sem eru alveg eins.
- Þá kemur loks `PRIMARY KEY` skilgreiningin. Hún segir okkur að dálkurinn sem við vorum að búa til sé aðallykill. Þá veit gagnagrunnskerfið að það getur notað þennan dálk til að þekkja hverja línu fyrir sig.
- Það síðasta, `AUTO_INCREMENT`, er bara til þess að aðstoða okkur við að setja inn gögn. Sé dálkur merktur með `AUTO_INCREMENT` þýðir það að við þurfum ekki að setja gildi inn í hann sjálf, gagnagrunnskerfið sér um að finna viðeigandi raðnúmer fyrir okkur. Þetta sést til dæmis á sýnidæmi [2.3](#). Þar er lykildálkinum einfaldlega sleppt í `INSERT` skipuninni.

Romsan `id` `INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT` skilgreinir sem sagt heiltöludálk sem má ekki fá `NULL`, sem gegnir hlutverki aðallykils og uppfærist sjálfur.

Nær allar töflur sem fara í notkun við raunverulegar aðstæður ættu að hafa aðallykil. Öll sýnidæmi í bókinni héðan í frá munu innihalda aðallykil.

Við kynnumst aðallyklum og öðrum lyklum örlítið betur í undirkafla [5.1](#).

### 3.8 Að eyða töflum

Það að gera villur er eðlilegur hluti af því að byrja að læra á SQL. Því miður er oft erfiðara að laga villur í `CREATE` og `INSERT` skipunum heldur en villum í öðrum forritunarmálum. Þegar `CREATE` eða `INSERT` skipun hefur verið keyrð er sjaldnast hægt að ýta bara á “Undo” takka til að komast til baka - upplýsingarnar eru komnar inn í gagnagrunninn.

Fyrsta aðferðin sem við lærum til að leiðrétta mistök er að eyða töflunni í heilu lagi. Til þess notum við, viti menn, SQL-skipun. Hún heitir `DROP TABLE`. Til að eyða töflu má skrifa `DROP TABLE` og svo nafnið á töflunni. Sjá sýnidæmi [3.13](#).

---

```
DROP TABLE NafnToflu;
```

---

Við sjáum fleiri leiðir til að uppfæra gagnagrunna í kafla [7](#). Til að byrja með látum við það að henda töflunum út í heilu lagi og búa þær til aftur duga til að komast í gegnum fyrsta hjallann.

Sýnidæmi 3.13: Töflu með nafnið `NafnToflu` er eytt úr gagnagrunninum. Þetta ber að gera með varúð - það að eyða töflu er varanlegt. Til að fá gögnin aftur inn í gagnagrunninn þyrfti að keyra aftur allar SQL-skipanirnar sem bjuggu hana til.

### 3.9 Yfirlit

Í þessum kafla kynntumst við helstu atriðum sem snúa að töflugerð í MySQL.

- Búa má til töflur með `CREATE TABLE` skipuninni.
- Setja má gögn inn í töflur með `INSERT` skipuninni.
- Nokkrar gagnagerðir í MySQL eru `INTEGER`, `CHAR`, `VARCHAR`, `DECIMAL`, `DOUBLE` og `DATE`. Gagnagerðin er ákveðin fyrir hvern dálk í töflu. Hún er tekin fram sem hluti af `CREATE TABLE` skipuninni.
- Ekki-gildið `NULL` táknar gögn sem eru óþekkt eða ekki til. Banna má dálki að taka við `NULL` með því að skrifa `NOT NULL` fyrir aftan gagnagerðina.
- Lyklar auðvelda gagnagrunnskerfinu að vinna með töflur. Allar alvöru töflur eiga að hafa aðallykil, `PRIMARY KEY`.
- Töflum má eyða með `DROP TABLE` skipun.

## 4

# Fyrirspurnir

Upplýsingar eru sjaldnast geymdar í gagnagrunnum upplýsinganna vegna. Markmiðið með að geyma upplýsingar er að gera það mögulegt að ná í þær aftur seinna.

Til að ná í upplýsingar notum við svokallaða SELECT skipun. Þetta er skipun sem við munum koma til með að nota mikið og kynnast vel.

Hver SELECT skipun er *lýsing* á einhverjum upplýsingum sem við viljum fá. Við lýsum því hvaða upplýsingar við viljum, gagnagrunnskerfið sér svo um að finna þær fyrir okkur.

### 4.1 SELECT

Allar SELECT skipanir innihalda að minnsta kosti lýsingu á því hvaða upplýsingar þarf að ná í.

Dæmi um ofurlitla SELECT skipun má sjá í sýnidæmi [4.1](#).

---

```
SELECT 2+2;
```

---

#### SELECT - FROM

Pegar SELECT skipun er skrifuð er það oftast í þeim tilgangi að ná upplýsingum úr SQL-töflu.

Pegar ná á upplýsingum úr töflu þarf að minnsta kosti tvennt að koma fram - hvar upplýsingarnar er að finna og hvaða upplýsingar skal velja. Þetta má gera í eftirfarandi skrefum:

Sýnidæmi 4.1: Lítil *SELECT* skipun. Hún inniheldur lýsingu á því hvaða upplýsingar á að finna: summuna  $2 + 2$ . Gagnagrunnskerfið getur reiknað hana út fyrir okkur.



1. Skrifa orðið **SELECT** (sem gerir skipunina að **SELECT** skipun)
2. Skrifa nöfn dálkanna sem velja skal
3. Skrifa orðið **FROM**
4. Skrifa nafn töflunnar sem velja skal úr.

Fyrri tvö skrefin lýsa þá því hvaða upplýsingar skal velja, þau seinni tvö lýsa því hvaðan þær koma.<sup>1</sup>

**SELECT** skipun með **FROM** klausu má sjá á sýnidæmi 4.2.

<sup>1</sup> Seinni tvö skrefin lýsa svokallaðri **FROM** klausu. **FROM** klausa lýsir því hvaðan upplýsingar koma. Oft er þetta bara nafn á einni töflu.

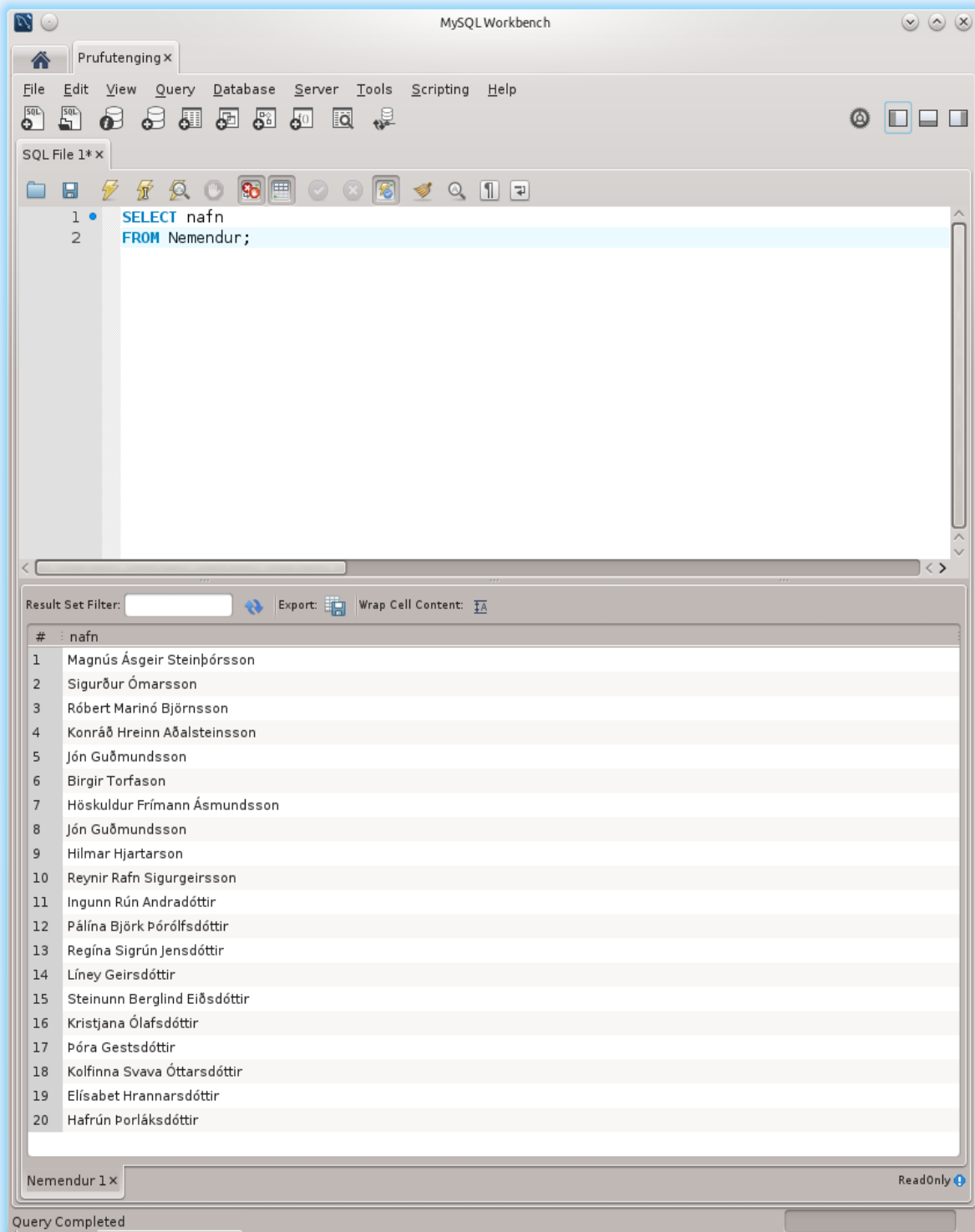
numer	nafn	kennitala	innritun
1	Magnús Ásgeir Steinþórsson	090698-6489	2014-07-01
2	Sigurður Ómarsson	251198-1369	2014-06-04
3	Róbert Marínó Björnsson	060998-2489	2014-07-14
4	Konráð Hreinn Aðalsteinsson	120498-8869	2014-06-02
5	Jón Guðmundsson	230598-2159	2014-07-03
6	Birgir Torfason	170798-7249	2014-06-06
7	Höskuldur Frímann Ásmundsson	020298-4139	2014-07-08
8	Jón Guðmundsson	210498-7889	2014-06-11
9	Hilmar Hjartarson	020798-4599	2014-07-16
10	Reynir Rafn Sigurgeirsson	211298-7239	2014-06-12
11	Ingunn Rún Andradóttir	161298-1589	2014-07-05
12	Pálína Björk Þórólfsdóttir	030798-0829	2014-06-09
13	Regína Sigrún Jensdóttir	140798-6499	2014-07-08
14	Líney Geirsdóttir	111098-3289	2014-06-21
15	Steinunn Berglind Eiðsdóttir	190398-1889	2014-07-04
16	Kristjana Ólafsdóttir	230298-4759	2014-06-01
17	Þóra Gestsdóttir	010498-8489	2014-07-05
18	Kolfinna Svava Óttarsdóttir	210498-5759	2014-06-02
19	Elísabet Hrannarsdóttir	050298-3109	2014-07-09
20	Hafrún Þorláksdóttir	250498-2849	2014-06-19

Tafla 4.1: Nokkrir uppskáldaðir nemendur fæddir árið 1998. Við munum velja upplýsingar úr þessari töflu í næstu sýnidæmum.

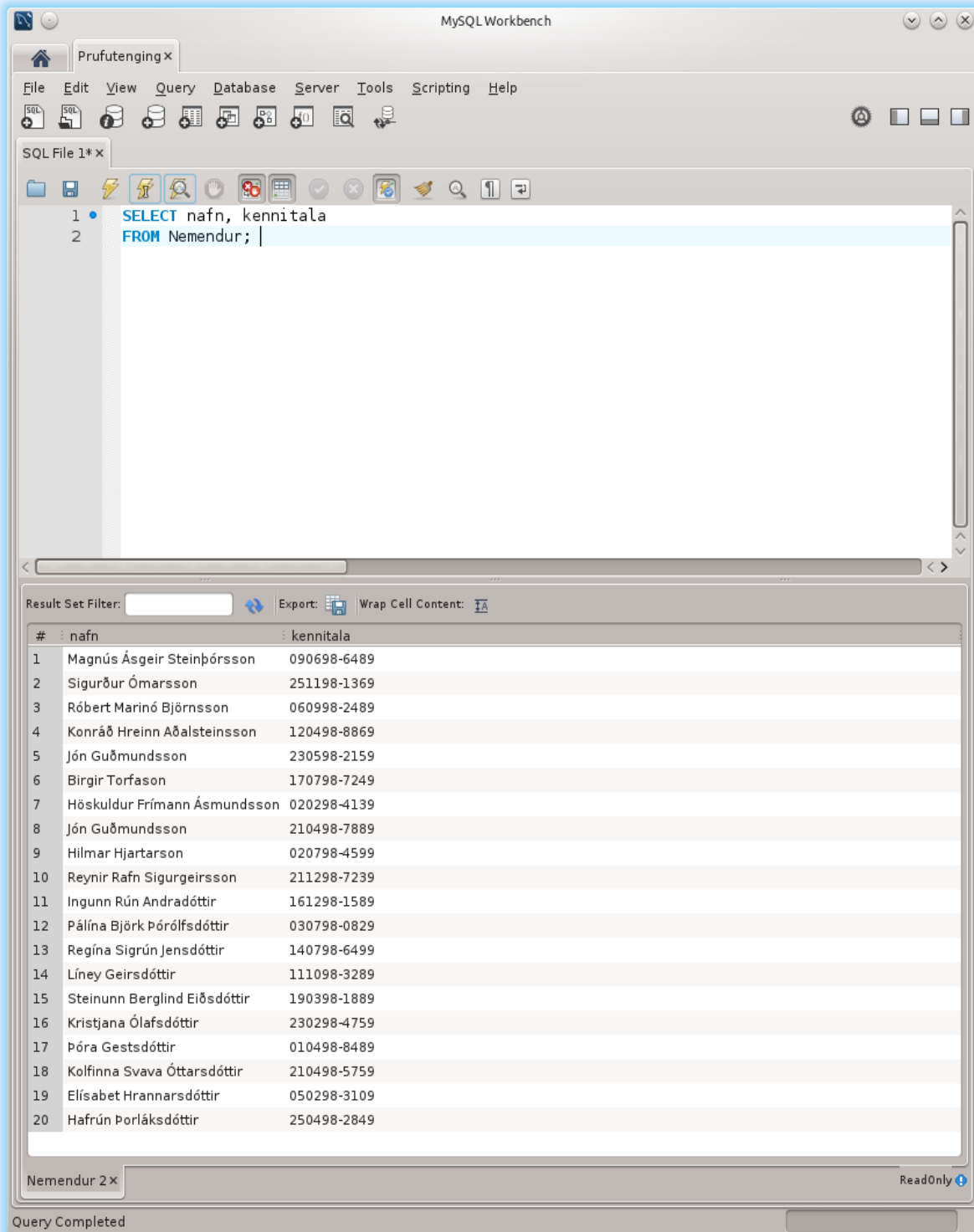
```
SELECT nafn
FROM Nemendur;
```

Sýnidæmi 4.2: **SELECT** skipun með **FROM** klausu. Hún velur allan “nafn” dálkinn úr töflunni Nemendur (4.1).

**SELECT** skipun getur náð í marga dálka (eða mörg atriði) í einu. Atriðin eru þá einfaldlega aðgreind með kommu. Þetta má sjá á mynd 4.2.



Mynd 4.1: Hér sést hvernig keyra má *SELECT* skipunina úr sýni-dæmi 4.2 í MySQL Workbench. Skipunin er í aðalglugganum, niðurstaða hennar sést fyrir neðan.



Mynd 4.2: *SELECT* skipun sem nær í marga dálka getur litið út á þessa leið í MySQL workbench. Allar upplýsingarnar úr dálkunum "nafn" og "kennitala" voru valdar. Aðrir dálkar sjást ekki.

## 4.2 WHERE klausan

Hingað til höfum við valið heila dálka með `SELECT` skipunum. Það sem við viljum hins vegar oftast gera er að finna ákveðnar upplýsingar í töflunni, frekar en að fá þær allar.

Til þess að fá bara þær upplýsingar sem við viljum búum við til “sú”<sup>2</sup> sem hleypir engum upplýsingum í gegn nema þeim sem við viljum.

Slík sía þarf að innihalda lýsingu á þeim gögnum sem hún á að hleypa á í gegn. Það er gert í klausu sem við köllum `WHERE` klausu og kemur fyrir aftan `FROM` klausuna. Dæmi um þetta má sjá í sýnidæmum 4.3 til 4.3.

---

```
SELECT nafn
FROM Nemendur
WHERE numer = 11;
```

---



---

```
SELECT kennitala
FROM Nemendur
WHERE nafn = 'Sigurður Ómarsson';
```

---



---

```
SELECT nafn, kennitala
FROM Nemendur
WHERE nafn = 'Jón Guðmundsson';
```

---

### Röksegðir

Í öllum `WHERE` klausunum hér á undan er um að ræða síur sem ekki hleypa neinum línunum í gegn nema að þær uppfylli eitt, nákvæmt skilyrði. Skilyrðið í sýnidæmi 4.3, `numer = 11`, hleypir til dæmis einungis þeim nemanda sem er með nákvæmlega númerið 11 í gegn.

Skilyrðin geta þó verið margs konar. `WHERE` klausan tekur nefnilega við næstum hvaða röksegð<sup>3</sup> sem er, þ.e.a.s. alls kyns samanburðum og staðhæfingum sem að lokum gefa gildin “satt” eða “ósatt”.<sup>4</sup>

<sup>2</sup> e. *filter*

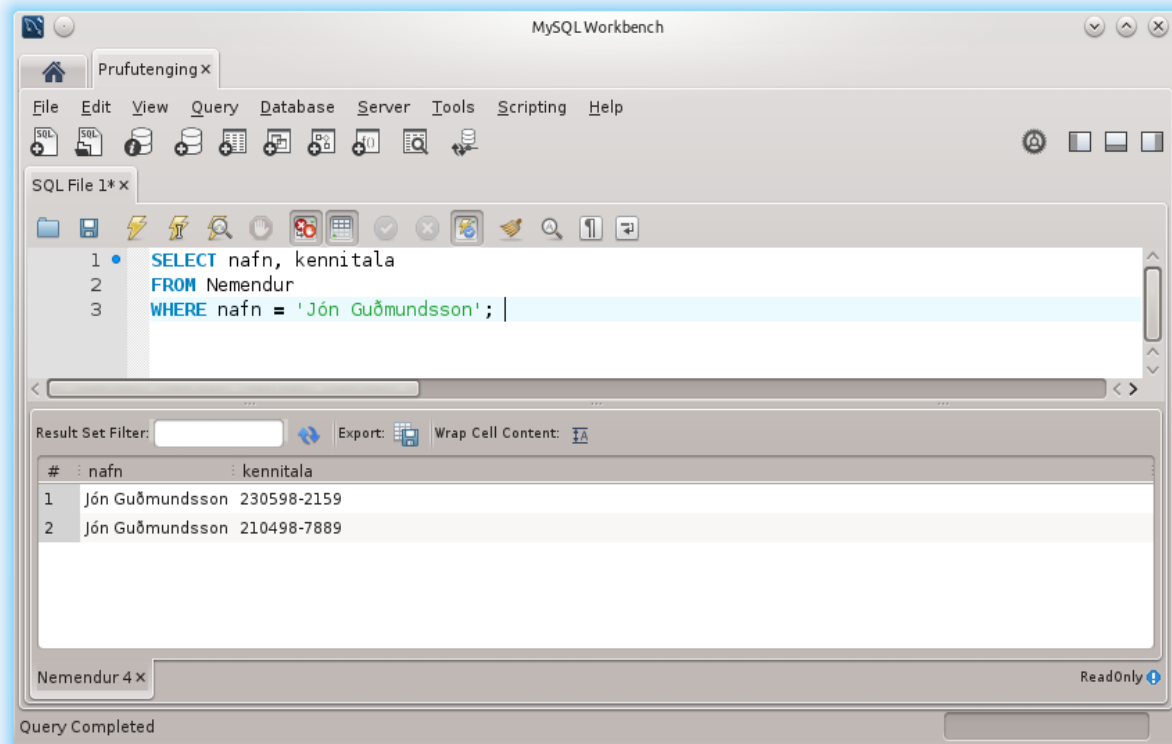
Sýnidæmi 4.3: `SELECT` skipun með `WHERE` klausu sem nær í nafn nemanda (úr töflu 4.1) þar sem “numer” dálkurinn er með gildið 11. Hún skilar einni línu, nafninu Ingunn Rún Andradóttir.

Sýnidæmi 4.4: `SELECT` skipun með `WHERE` klausu sem nær í kennitölu nemanda eftir nafni hans. Hún skilar einni línu, kennitölunni 251198-1369.

Sýnidæmi 4.5: Skilyrðið sem sett er fram í `WHERE` klausu getur átt við meira en eina línu í töflunni. Þessi skipun finnur nöfn og kennitölu allra sem heita Jón Guðmundsson. Þeir reynast vera tveir, með kennitölurnar 230598-2159 og 210498-7889.

<sup>3</sup> e. *boolean expression*

<sup>4</sup> Í MySQL er “satt” táknad með tölunni 1 eða orðinu `TRUE`. “Ósatt” er táknad með 0 eða `FALSE`.



Mynd 4.3: Hér sést sýnidæmi 4.5 í MySQL Workbench.

Orðið “röksegð” kann að hljóma undarlega, en um kunnuglegt fyrirbæri er að ræða. “Skilyrðið” *numer* = 11 er til dæmis röksegð. Segðin er sönn þegar *numer* tekur gildið 11, annars ekki. Fleiri dæmi um segðir má sjá á töflu 4.2.

Segð	Útskýring
<i>numer</i> = 5	Sönn þegar gildið í <i>numer</i> er nákvæmlega 5.
<i>numer</i> > 5	Sönn þegar gildið í <i>numer</i> er stærra en 5.
<i>numer</i> < 5	Sönn þegar gildið í <i>numer</i> er minna en 5.
<i>numer</i> >= 5	Sönn þegar gildið í <i>numer</i> er 5 eða stærra.
<i>numer</i> <= 5	Sönn þegar gildið í <i>numer</i> er 5 eða minna.
<i>numer</i> != 5	Sönn þegar gildið í <i>numer</i> er ekki 5.

Þegar dálkheiti eru notuð í WHERE klausu má því líta á það sem svo að við skoðum öll gildi sem eru í dálkinum, eitt í einu<sup>5</sup>, skiptum dálkheitinu út fyrir gildið og athugum hvort að segðin sé sönn. Ef segðin sem fæst út línu í gagnagrunninum er sönn, þá fær línan að fara áfram í niðurstöðurnar. Dæmi um hvernig flokkun af þessu tagi fer fram má sjá á mynd 4.4.

Tafla 4.2: Röksegðir sem nota mætti í WHERE klausu SELECT skipunar. Hér er *numer* nafnið á dálki sem inniheldur tölur.

<sup>5</sup> Reyndar getur gagnagrunnskerfi oft gert mun betur en svo að þurfa að skoða öll gildin. Þetta tengist sérstaklega *lyklum*, sem við lítum á í undirkafla 5.1.

SQL-skipun: `SELECT x, y  
FROM A  
WHERE y > 3;`

Taflan A		Segð	S/O	Niðurstaða
x	y	$\{ y > 3 \}$		x y
1	5	$\Rightarrow 5 > 3$	$\Rightarrow$ Satt	$\Rightarrow$ 1 5
2	2	$\Rightarrow 2 > 3$	$\Rightarrow$ Ósatt	
7	3	$\Rightarrow 3 > 3$	$\Rightarrow$ Ósatt	
7	4	$\Rightarrow 4 > 3$	$\Rightarrow$ Satt	$\Rightarrow$ 7 4

Mynd 4.4: Röksegð í WHERE klausu.

Eins og við höfum t.d. séð á sýnidæmi 4.5 þarf röksegð ekki að innihalda eingöngu samanburði með tölum. T.d. er hægt að bera saman strengi og dagsetningar.

Dagsetningar eru bornar saman líkt og tölur. Strengir eru hins vegar bornir saman í stafrófsröð<sup>6</sup>. Dæmi um segðir sem innihalda strengi má sjá á töflu 4.3.

Nokkur atriði ber þó að varast þegar strengir eru notaðir til samanburðar í MySQL.

SÉRSTAKIR STAFIR OG TÁKN á borð við upphrópunarmerki og bil eru ekki alltaf borin saman á þann hátt sem við giskum á. Stafrófsröð er ekki skilgreind fyrir þessi tákn. Þösum okkur þegar við notum `>` eða `<` til að bera saman strengi sem ekki innihalda eingöngu bókstafi.<sup>7</sup>

SÉU STRENGIR AF MISMUNANDI LENGÐUM BORNIR SAMAN er bilum bætt við hægra megin við styttri strenginn áður en samanburðurinn er framkvæmdur. Þannig myndi `'a' > 'aa'` vera breytt í `'a ' > 'aa'`. Þetta getur leitt til óvæntra niðurstaðna. Dæmi um niðurstöðu sem við fyrstu sýn kann að virðast undarleg má sjá á sýnidæmi 4.6.

---

```
SELECT nafn
FROM Nemendur
WHERE nafn > 'M';
```

---

<sup>6</sup> Þannig að stafurinn *a* sé “minni en” stafurinn *b*.

<sup>7</sup> Gagnlegt getur verið að lesa sér til um *Collations* í MySQL til að öðlast frekari skilning á strengjasamanburðum. Ekki er fjallað um það í þessari bók.

Sýnidæmi 4.6: *SELECT*-skipun sem ber saman öll nöfn við strenginn *'M'*. Skipunin skilar nöfnum allra nemanda sem eru á eftir *M* í stafrófinu, að *nemendum sem byrja á M meðtöldum!* Þetta er vegna þess að strengurinn *M* er lengdur með bilum áður en samanburðurinn er framkvæmdur. Bil er álitid “minna” en allir bókstafir, svo segðin verður sönn fyrir öll nöfn sem byrja á *M*.

Segð	Gildi
'a' = 'a'	Satt.
'a' >= 'a'	Satt.
'b' > 'a'	Satt, af því að <i>b</i> er á eftir <i>a</i> í stafrófinu.
'a' > 'b'	Ósatt, af því að <i>b</i> er á eftir <i>a</i> í stafrófinu.
'ab' > 'aa'	Satt, af því að aftari stafir útkljá jafntefli.
'a b' > 'a a'	Satt, af því að <i>bil</i> er sagt minna en <i>a</i> .
'a b' > 'aaa'	Ósatt, af því að <i>bil</i> er sagt minna en <i>a</i> .
'ab' > 'aabb'	Satt, af því að í samanburði mislangra strengja er <i>bilum</i> bætt við hægra megin við styttri strenginn.

Tafla 4.3: Dæmi um hvernig MySQL meðhöndlar röksegðir með strengjum.

### LIKE og "wildcards"

Táknin sem við höfum séð í WHERE klausum (t.d. = og >) hafa gert okkur kleift að skrifa margar mismunandi röksegðir.

Ein gerð af segð sem við höfum ekki getað gert er að athuga hvort að strengur passi við hluta af öðrum streng. Til þess höfum við ákveðið lykilorð sem heitir LIKE.

Það að nota LIKE í röksegð er líkt og að nota =. LIKE hefur þó ákveðinn möguleika sem = hefur ekki - þann að skilja eftir "óþekkta" stafi í strengnum sem notaður er til samanburðar. Þessi óvissutákn geta komið í stað hvaða stafs sem er. Þau eru kölluð "wildcards" á ensku. Við skoðum tvö mismunandi wildcards í þessum undirkafla.

Annars vegar er táknið \_ (strik niðri). \_ getur komið í staðinn fyrir hvaða *einn* staf sem er í strengnum.

Hins vegar er táknið % (prósentermerki). % getur komið í staðinn fyrir hvaða fjölda stafa sem er (líka fjöldann 0).

Dæmi 4.7 til 4.11 sýna notkun LIKE og wildcard-tákna.

```
SELECT nafn, kennitala
FROM Nemendur
WHERE nafn LIKE 'K%';
```

Sýnidæmi 4.7: *SELECT* skipun sem finnur alla nemendur í nemendatöflunni sem byrja á stafnum *K*. Til þess er notaður samanburðarstrengur sem hefur wildcard-táknið % á eftir stafnum *K*, svo að % komi í staðinn fyrir allt sem á eftir *K* kemur.

```
SELECT nafn, kennitala
FROM Nemendur
WHERE nafn LIKE '%dóttir';
```

Sýnidæmi 4.8: *SELECT* skipun sem finnur alla nemendur í nemendatöflunni sem endar á "dóttir". % kemur hér á undan "dóttir" svo að það geti komið í staðinn fyrir alla stafi sem gætu verið þar á undan.

---

```
SELECT nafn, kennitala
FROM Nemendur
WHERE nafn LIKE '%geir%';
```

---

Sýnidæmi 4.9: *SELECT* skipun sem finnur alla nemendur í nemendatöflunni sem innihalda strenginn “geir” einhvers staðar í nafni sínu. Segðin í *WHERE*-klausunni er t.d. sönn fyrir nemandann sem heitir Ásgeir að millinafni og nemandann sem er Sigurgeirsson.

---

```
SELECT nafn, kennitala
FROM Nemendur
WHERE nafn LIKE '_____';
```

---

Sýnidæmi 4.10: *SELECT* skipun sem finnur alla nemendur sem eru með nákvæmlega 17 stafi í nafni sínu (að bilum meðtöldum). Hvert `_` tákn kemur í stað nákvæmlega eins stafs. Við sjáum aðra leið til að gera þetta í undirkafla 4.4.

---

```
SELECT nafn, kennitala
FROM Nemendur
WHERE kennitala LIKE '___04%';
```

---

Sýnidæmi 4.11: *SELECT* skipun sem finnur alla nemendur í nemendatöflunni sem fæddir eru í apríl (með stafina `04` í þriðja og fjórða sæti í kennitölu sinni). Hún finnur einungis þá nemendur, vegna þess að `_` táknin tvö geta komið í staðinn fyrir nákvæmlega tvö önnur tákn, hvorki fleiri né færri.



## Mörg aðskilin skilyrði

Hver SELECT skipun getur einungis haft eina WHERE klausu. Engu að síður er hægt að setja mörg skilyrði um þær raðir sem eiga að birtast í niðurstöðum skipunarinnar með því að tengja margar röksegðir saman. Það má gera með lykilorðunum AND og OR<sup>8</sup>. Þannig mætti til dæmis búa til samsettu röksegðina  $x < 1$  OR  $x > 5$  úr segðunum  $x < 1$ ,  $x > 5$  og lykilorðinu OR.

Þegar skoða á hvort að samsett segð sé sönn eða ósönn þarf fyrst að athuga hvort að segðirnar sem hún er búin til úr séu sannar eða ósannar. Þá verður úr röð af gildunum "satt" og "ósatt", með AND og OR á milli.

Næst þarf að athuga öll AND í röðinni. Þegar AND stendur á milli tveggja gilda sem eru sönn mynda þau saman gildið "satt", annars gildið "ósatt". Þegar AND stendur á milli tveggja gilda þurfa sem sagt bæði gildin að vera sönn til að útkoman sé sönn.

Að lokum eru öll OR skoðuð. Þegar OR stendur á milli tveggja gilda er nóg að annað gildanna sé "satt" til að þau myndi saman gildið "satt". Séu bæði gildin ósönn mynda þau gildið "ósatt".

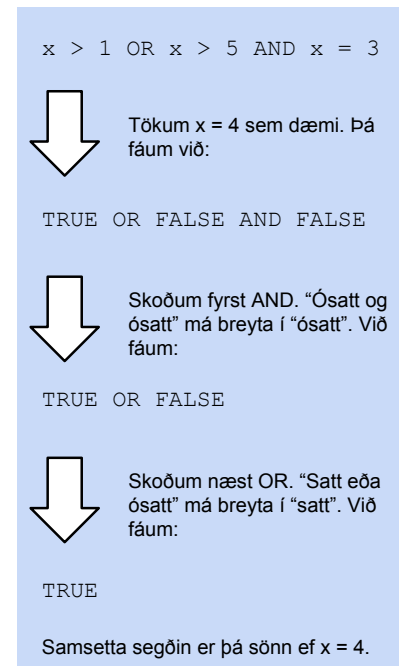
ALLT ÞETTA TAL um segðir og samsetningar kann að hljóma þurrt. Þetta á sér þó hliðstæður í mun raunverulegri aðstæðum.

Skoðum dæmi 4.7 og 4.8 aftur. Í fyrra dæminu finnum við alla nemendur sem heita nafni sem byrjar á bókstafnum "K", í seinna dæminu finnum við alla nemendur sem heita nafni sem endar á "dóttir". Viljum við finna alla nemendur með nafn sem bæði byrjar á "K" og endar á "dóttir" getum við tengt skilyrðin tvö saman. Úr verður WHERE klausan í sýnidæmi 4.12.

OR kemur fyrir þegar einungis eitt af tveimur skilyrðum þarf að vera uppfyllt. Lítum aftur á dæmi 4.11, þar sem allir nemendur sem fæddir eru í apríl eru valdir. Vildum við breyta skipuninni svo að hún myndi ná í alla nemendur sem fæddir eru annaðhvort í apríl eða í júní gætum við notað OR á svipaðan hátt og gert er í dæmi 4.13.

<sup>8</sup> Fleiri lykilorð af þessum toga eru til, til dæmis XOR.

Mynd 4.5: Hér sést hvernig lesa má út úr samsettri röksegð.



Útkoma AND			Útkoma OR		
x	y	x AND y	x	y	x OR y
1	1	1	1	1	1
1	0	0	1	0	1
0	1	0	0	1	1
0	0	0	0	0	0

Tafla 4.4: Hér sést hvaða áhrif það hefur að setja AND og OR á milli tveggja breyta. Gildi breytanna er í vinstri dálkunum, útkoman sést í dálkunum til hægri.

---

```
SELECT nafn, kennitala
FROM Nemendur
WHERE nafn LIKE 'K%' AND nafn LIKE '%dóttir';
```

---

Sýnidæmi 4.12: *SELECT* skipun sem finnur alla nemendur í nemendatöflunni sem heita nafni sem bæði byrjar á “K” og endar á “dóttir”.

---

```
SELECT nafn, kennitala
FROM Nemendur
WHERE kennitala LIKE '___04%'
OR kennitala LIKE '___06%';
```

---

Sýnidæmi 4.13: *SELECT* skipun sem finnur alla nemendur í nemendatöflunni sem fæddir eru í apríl eða júní.

Freistandi getur verið að skrifa *WHERE* klausur þar sem mörg samanburðargildi eru talin upp hvert af öðru, tengd með *AND* eða *OR*. Þetta gengur ekki! Munum alltaf að það eru röksegðir sem *AND* og *OR* tengja saman, það að reyna að nota þau til að tengja saman almenna strengi eða tölur getur leitt til óvæntra niðurstaðna.<sup>9</sup>

Munum líka að *OR* í röksegð er ekki alveg það sama og “eða” í talmáli. Í talmáli þýðir “eða” oft val á milli tveggja möguleika (má bjóða þér salat eða franskar með hamborgaranum?) en í röksegðum skilar samsett segð sönnu svo lengi sem annar “möguleikinn” á við.<sup>10</sup>

<sup>9</sup> Dæmi um þessa villu væri t.d. að skrifa *WHERE x = 1 OR 3* til að finna línurnar þar sem *x* er 1 eða 3. Rétt væri að skrifa *WHERE x = 1 OR x = 3*.

<sup>10</sup> Um þetta er til gamall brandari.

Eiginmaður tölvunarfræðings spyr: “Langar þig í fisk eða kjöt í kvöldmat?” Tölvunarfræðinginn langar í kjöt, svo hún svarar: “Já.”

### Að velja úr mengi - *IN*

Oft þarf að gera mjög marga samanburði í *WHERE* klausu. Þá getur orðið flókið að koma skilyrðunum fyrir.

Hægt er að gera suma samanburði (sérstaklega þá sem annars þyrftu mörg *OR* skilyrði) einfaldari með því að nota lykilorðið *IN*. Segð sem inniheldur *IN* ber saman gildi við “mengi” gilda og skilar sönnu ef samanburðargildið er í menginu.<sup>11</sup>

Dæmi um notkun *IN* má sjá á sýnidæmi 4.14.

<sup>11</sup> *IN* getur verið sérstaklega gagnlegt þegar undirfyrirspurnir koma við sögu. Sjá kafla 6.3.

### Að snúa við skilyrði - *NOT*

Merkingu flestra röksegða má snúa við með því að setja lykilorðið *NOT* fyrir framan segðina. Hafi segðin áður skilað sönnu skilar hún þess í stað ósönnu, hafi hún skilað ósönnu skilar hún sönnu. Dæmi um notkun *NOT* má sjá á sýnidæmi 4.15.

---

```

SELECT nafn, kennitala
FROM Nemendur
WHERE numer = 1
      OR numer = 3
      OR numer = 5
      OR numer = 7;

SELECT nafn, kennitala
FROM Nemendur
WHERE numer IN (1, 3, 5, 7);

```

---

Sýnidæmi 4.14: Tvær *SELECT* skipanir sem gera sama hlutinn. Þær finna báðar nemendurna með númerin 1, 3, 5 og 7. Seinni skipunin er þó töluvert skárri!

---

```

SELECT nafn, kennitala
FROM Nemendur
WHERE nafn NOT LIKE 'K%';

```

---

Sýnidæmi 4.15: *SELECT* skipun sem finnur alla nemendur sem ekki byrja á stafnum “K”.

### 4.3 Um föll

Fall<sup>12</sup> er mikilvægt stærðfræðihugtak sem kemur víða við í forritun, þar á meðal í SQL.

Líta má á fall sem nokkurs konar “vél” sem breytir einu í annað. Fall *tekur eitthvað inn* og *skilar* einhverju öðru.

Við getum t.d. litið á brauðrist sem fall. Brauðrist *tekur inn* brauð og *skilar* ristuguð brauði. Á stærðfræðimáli myndum við skrifa það svona:

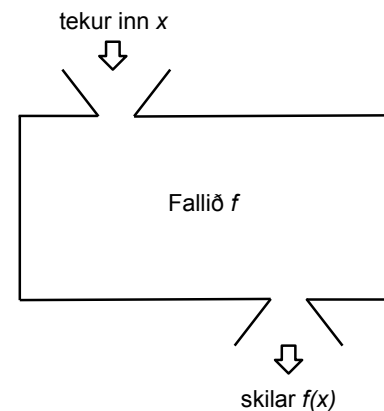
$$\text{brauðrist}(\text{brauð}) = \text{ristað brauð}$$

### 4.4 Helstu “scalar” föll

Í þessum undirkafla ætlum við að skoða MySQL-föll sem taka inn eitt eða ekkert gildi og skila einu gildi. Þau eru kölluð “scalar föll”. Ekki er til íslenskt orð yfir hugtakið sem er í almennri notkun.

<sup>12</sup> e. *function*

Mynd 4.6: Líta má á fall  $f$  sem vél sem breytir gildinu  $x$  í gildið  $f(x)$ .



*Lengd strengs - CHAR\_LENGTH*

Fyrsta fallið sem við skoðum er MySQL-fallið `CHAR_LENGTH`. Það fall tekur inn streng og skilar fjölda stafa sem er í strengnum. Bil og önnur tákn eru tekin með. Almenn lýsing á fallinu gæti þá verið svona:

$$\text{CHAR\_LENGTH}(\text{texti}) = \text{fjöldi stafa í texta}$$

Og dæmi um notkun (á stærðfræðiformi)

$$\text{CHAR\_LENGTH}(\text{'Jónas'}) = 5$$

Notkun fallsins í MySQL má sjá í sýnidæmi 4.16.

---

```
SELECT CHAR_LENGTH('Jónas');
```

---

Annað fall, sem heitir einfaldlega `LENGTH`, er líka til. Það fall skilar fjölda bæta sem eru í strengnum. Oft er einn bókstafur geymdur í einu bæti, svo `LENGTH` og `CHAR_LENGTH` skila oft sömu tölu, en munur getur komið fram þegar bókstafir eru geymdir í meira en einu bæti. Séríslenskir stafir eru t.d. venjulega geymdir í tveimur bætum. `LENGTH('Jónas')` myndi þess vegna skila tölunni 6.

Sýnidæmi 4.16: *SELECT* skipun sem finnur fjölda stafa í orðinu “Jónas” með `CHAR_LENGTH` fallinu. Þessi skipun skilar sem sagt tölunni 5.

*UCASE og LCASE*

`UCASE` er fall sem tekur inn streng og skilar honum í hástöfum. `UCASE` er stytting á enska orðinu “uppercase”. Notkun fallsins í MySQL má sjá í sýnidæmi 4.17.

---

```
SELECT UCASE('Jónas');
```

---

Sýnidæmi 4.17: *SELECT* skipun sem skilar strengnum “JÓNAS”.

Fallinu má lýsa á eftirfarandi hátt:

$$\text{UCASE}(\text{textinn}) = \text{textinn í hástöfum}$$

Dæmi um notkun er:

$$\text{UCASE}(\text{'Jónas'}) = \text{'JÓNAS'}$$

`LCASE` er svipað fall. Það fall tekur inn streng og skilar honum í lágstöfum. `LCASE` er stytting á orðinu “lowercase”.

## NOW

NOW er fall sem skilar núverandi tímasetningu (skv. klukku MySQL-serversins). NOW er ólíkt föllunum hér á undan að því leytinu til að það þarf ekki að taka neitt inn, það skilar bara gildi (þessi vél þarf ekkert hráefni!).

Til að prenta núverandi tímasetningu út mætti sem sagt einfaldlega keyra skipunina `SELECT NOW()`; <sup>13</sup>

<sup>13</sup> Þó að NOW fallið taki ekkert inn þurfa svigarnir samt að vera til staðar. Svigarnir segja MySQL að um fall sé að ræða.

## Notkun scalar falla

Föllin sem við höfum farið yfir eru sjaldnast notuð ein og sér. Oftast eru þau notuð á eftirfarandi vegu:

1. Sem hluti af lýsingu á því hvaða dálka SELECT skipun á að ná í.
2. Sem hluti af WHERE klausu, til samanburðar við gildi.

Fyrri notkunina má sjá í sýnidæmi 4.18. Þar er allur dálkurinn *nafn* settur inn í CHAR\_LENGTH fallið.

Áðan sögðum við að CHAR\_LENGTH taki aðeins við einum streng. Þegar heill dálkur er settur inn í fall sem einungis tekur við einum streng (scalar fall) er fallið notað á hverja línu í dálkinum, eina í einu. Niðurstöðurnar birtast þá sem heill dálkur líka, líkt og sjá má á töflu 4.5.

---

```
SELECT nafn, CHAR_LENGTH(nafn)
FROM Nemendur
WHERE nafn LIKE '%dóttir';
```

---

Sýnidæmi 4.18: SELECT skipun sem finnur lengd nafna allra nemenda í nemendatöflunni sem enda á "dóttir". Niðurstöðu má sjá á töflu 4.5.

Seinni notkunina má sjá í sýnidæmi 4.19. Þar er fallið CHAR\_LENGTH notað í WHERE klausu. Reiknað er upp úr fallinu fyrir hverja línu og útkoman borin saman við tölu.

---

```
SELECT nafn
FROM Nemendur
WHERE CHAR_LENGTH(nafn) = 16;
```

---

Sýnidæmi 4.19: SELECT skipun sem finnur nöfn allra nemenda sem heita sextán stafa nafni (að bilum meðtöldum).

nafn	CHAR_LENGTH(nafn)
Ingunn Rún Andradóttir	22
Pálína Björk Þórólfsdóttir	26
Regína Sigrún Jensdóttir	24
Líney Geirsdóttir	17
Steinunn Berglind Eiðsdóttir	28
Kristjana Ólafsdóttir	21
Þóra Gestsdóttir	16
Kolfinna Svava Óttarsdóttir	27
Elísabet Hrannarsdóttir	23
Hafrún Þorláksdóttir	20

Tafla 4.5: Niðurstaða sýnidæmis 4.18.

#### 4.5 GROUP BY

Hingað til höfum við unnið með hrá gögn eins og þau koma fram í upphaflegu töflunni. Hver lína í niðurstöðunum samsvarar nákvæmlega einni línu í töflunni sem valið er úr.

En þetta dugar okkur ekki alltaf. Stundum þurfum við að geta litið á margar línur í einu.

Til þess að skoða margar línur í einu er GROUP BY klausan gagnleg. Hún tekur töfluna sem valið er úr og skiptir henni í “hópa”.

Hópar af línunum eru búnir til út frá sameiginlegum atriðum. Þessi “sameiginlegu atriði” eru gildi í einum eða fleiri dálkum.

Einföld GROUP BY klausa tekur sem sagt við nafni á dálki og setur allar línur sem hafa sama gildið í þeim dálki saman í hóp. Dæmi um hvernig svona skipting gæti gengið fyrir sig má sjá á mynd 4.7.

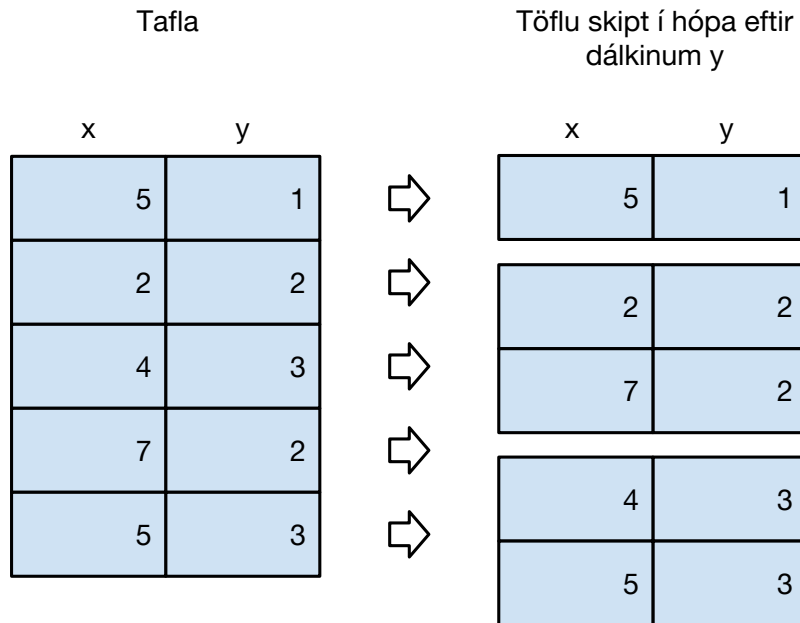
Mikilvægt er að átta sig á að þegar GROUP BY klausa er til staðar í SELECT skipun samanstanda niðurstöðurnar ekki af línunum, heldur *hópum* af línunum. Klausan breytir eðli skipunarinnar.

#### 4.6 Samsteypuföll

Föllin sem við höfum skoðað hingað til hafa ekki tekið inn meira en eitt gildi í einu. Nú lítum við á föll sem geta tekið inn fjöldann allan af gildum og unnið með þau sem eina heild. Slík föll eru kölluð samsteypuföll<sup>14</sup>.

Samsteypuföll í MySQL geta sem sagt tekið inn mengi af gildum og skilað svo upplýsingum um mengið. Mengið getur verið heill dálkur eða “hópur” af gildum sem GROUP BY klausa býr til.

<sup>14</sup> e. *aggregate function*



Mynd 4.7: Sýnir hvernig GROUP BY klausa gæti skipt töflu upp í hópa eftir því hvaða gildi er í dálkinum y.

## COUNT

Mest notaða samsteypufallið er líklega fallið COUNT. COUNT tekur inn mengi af gildum og skilar fjölda þeirra - fallið telur stökin.

Til að kynnst COUNT aðeins betur skulum við byrja á að skoða nýja töflu, töflu 4.6.

numer	audkenni	fag	onn
1	FOR1A3U	Forritun	1
2	VSH1A3U	Vefhönnun	1
3	GSÖ1G2U	Notkun gagnasafna	1
4	TÆK1A1U	Tölvutækni	1
5	FOR1B3U	Forritun	2
6	VSH2A3U	Vefhönnun	2
7	GSÖ1F2U	Notkun gagnasafna	2
8	TÆK2A3U	Tölvutækni	2
9	FOR2B2U	Forritun	3
10	VSH2B2U	Vefhönnun	3
11	GSÖ2B2U	Notkun gagnasafna	3
12	TÆK2B2U	Tölvutækni	3
13	GRU2L4U	Lokaverkefni grunndeildar	3

Tafla 4.6: Áfangar grunndeildar tölvubrautar Tækniskólans. Í töflunni eru upplýsingar um auðkenni áfangans (nafn hans), fagið sem áfangarnir tilheyra, og önnina sem þeir eru kenndir á. Dálkurinn *numer* er aðalkey.

Við getum t.d. notað COUNT fallið til að komast að því hversu margir áfangar eru í grunndeildinni. Slíka skipun má sjá á sýnidæmi 4.20.

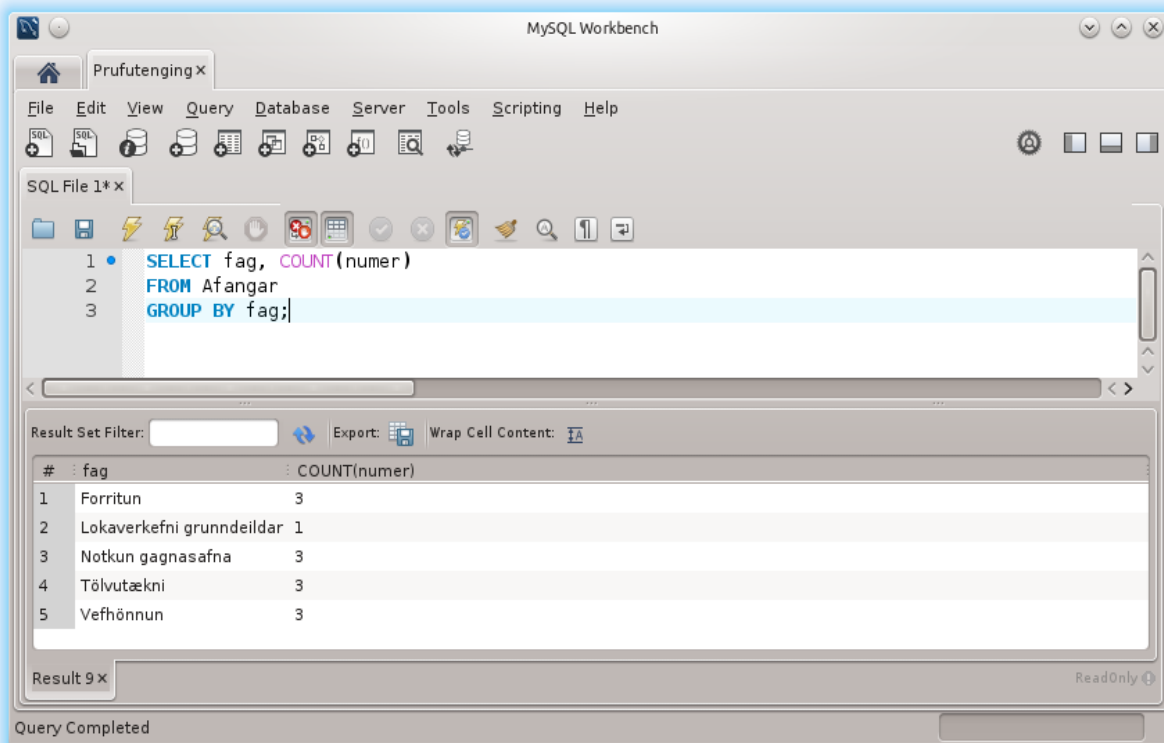
---

```
SELECT COUNT (numer)
FROM Afangar;
```

---

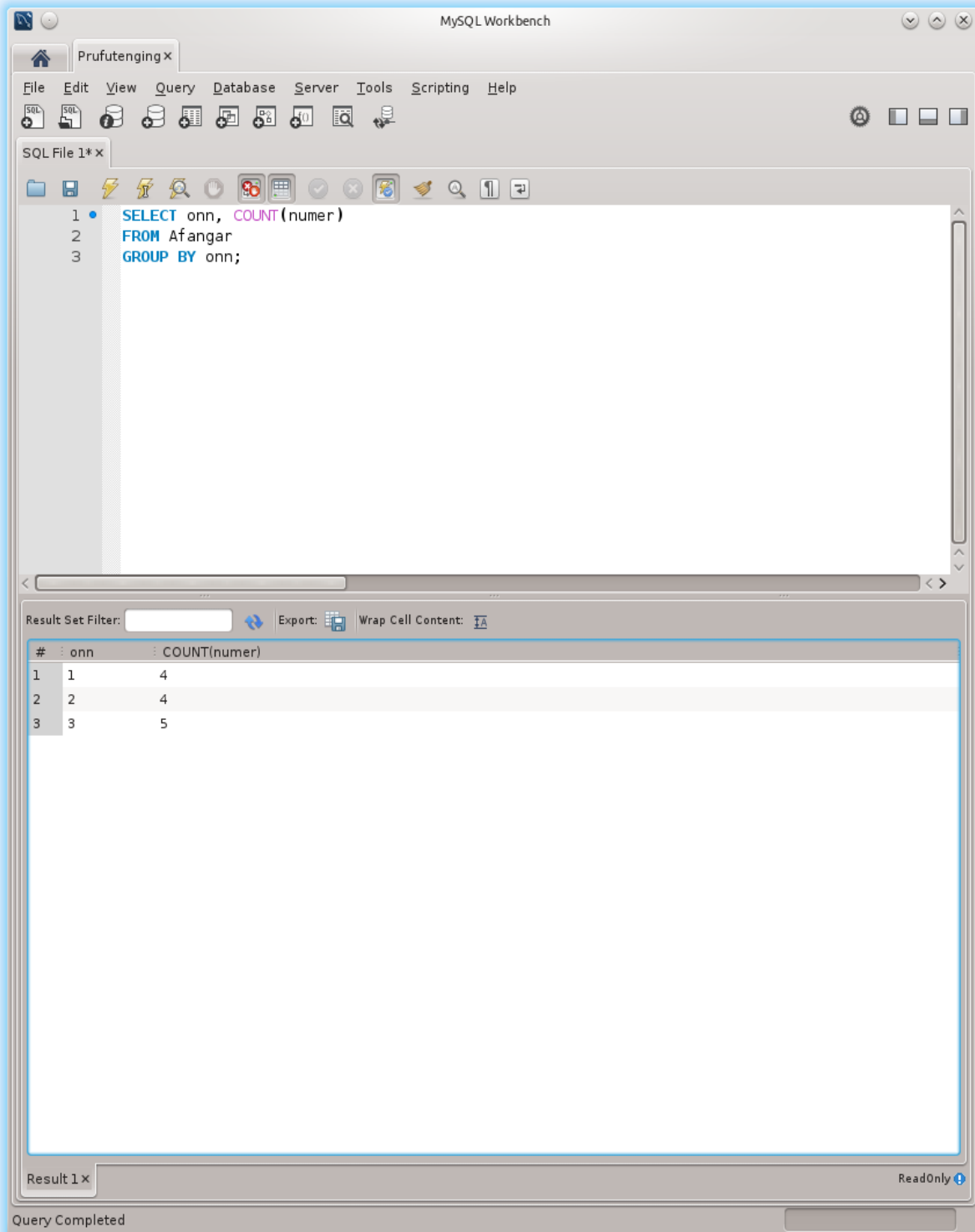
Sé COUNT notað með GROUP BY klausu er ekki allur dálkurinn talinn, heldur er fjöldi staka (eða lína) í hverjum hópi talinn fyrir sig. Dæmi um slíka skipun og hvernig niðurstöður hennar geta litið út má sjá á myndum 4.8 og 4.9.

Sýnidæmi 4.20: *SELECT* skipun sem finnur fjölda lína í *Afangar* töflunni (töluna 13). Í raun er fjöldi gilda í dálkinum *numer* talinn, en þar sem við vitum að *numer* er aðalkeyll getum við verið viss um að allar línurnar séu taldar.



Mynd 4.8: Dæmi um *SELECT* skipun með *GROUP BY* klausu sem raðar áföngum saman í hópa eftir því hvaða fagi þeir tilheyra. Síðan telur *COUNT* hversu margar áfangar tilheyra hverju fagi.





Mynd 4.9: Dæmi um *SELECT* skipun með *GROUP BY* klausu sem raðar áföngum saman í hópa eftir því hvaða önn þeir eru kenndir á. Síðan telur *COUNT* hversu margir áfangar eru á hverri önn.

*Önnur samsteypuföll - MIN, MAX, SUM, og AVG*

Fleiri samsteypuföll eru til en COUNT. Lítum á nokkur til viðbótar. Öll eiga þau sameiginlegt að taka inn mengi af gildum og skila okkur upplýsingum um mengið sjálft.

- MIN tekur inn mengi og skilar minnsta stakinu í menginu.
- MAX tekur inn mengi og skilar stærsta stakinu í menginu.
- SUM tekur inn mengi og skilar summu staka í menginu.
- AVG tekur inn mengi og skilar meðaltali staka í menginu.

Þau eru notuð á svipaðan hátt og COUNT. Skoðum enn aðra töflu, töflu 4.7, til að kynnst þeim betur.

numer	nafn	einkunn
1	Birgir Torfason	5.5
2	Höskuldur Frímann Ásmundsson	3.0
3	Jón Guðmundsson	9.0
4	Hilmar Hjartarson	5.0
5	Reynir Rafn Sigurgeirsson	8.0
6	Ingunn Rún Andradóttir	10.0
7	Pálína Björk Þórólfsdóttir	3.5
8	Regína Sigrún Jensdóttir	1.5
9	Líney Geirsdóttir	2.5
10	Steinunn Berglind Eiðsdóttir	4.0

Tafla 4.7: Nokkrir nemendur og einkunnir þeirra.

Við getum notað samsteypuföll til að fá frekari upplýsingar um töfluna. Sjá sýnidæmi 4.21 og 4.22.

```
SELECT MIN(einkunn)
FROM Einkunnir;
```

Sýnidæmi 4.21: *SELECT* skipun sem finnur lægstu einkunnina í einkunnatöflunni. Hún finnur gildið 1.5.

*Athugum þó* að ekki væri hægt að finna hver hefur þá einkunn með því að skrifa *SELECT nafn, MIN(einkunn)*. *MIN* er samsteypufall sem vinnur (hér) með dálkinn *einkunn* í heild sinni, það veit ekki af öðrum dálkum. Undirkaflar 4.7 og 6.3 sýna dæmi um vinnslu með samsteypuföll.

---

```
SELECT AVG(einkunn)
FROM Einkunnir;
```

---

Sýnidæmi 4.22: *SELECT* skipun sem finnur meðaleinkunnina í einkunnatöflunni. Hún finnur gildið 5.2.

## 4.7 HAVING

Við vorum búin að komast að því að *GROUP BY* klausa breytir eðli *SELECT* skipunar sem hún tilheyrir. Niðurstöður skipunar sem inniheldur *GROUP BY* klausuna eru nefnilega hópar af línur (hóplínur), ekki línur.

Þetta hefur nokkrar afleiðingar í för með sér. Sérstaklega er sú afleiðing áberandi að *WHERE* klausan virkar ekki sem fyrr. *WHERE* klausan vinnur með línur, hún getur ekki unnið með hóplínur.

Til þess að sía burt hóplínur þarf að nota enn aðra klausu - *HAVING*. *HAVING* virkar eins og *WHERE* klausan, nema hún vinnur með hóplínur.

*HAVING* klausuna má sjá á sýnidæmi 4.23.

---

```
SELECT fag, COUNT(enumer)
FROM Afangar
GROUP BY fag
HAVING COUNT(enumer) = 3;
```

---

Sýnidæmi 4.23: *SELECT* skipun með *HAVING* klausu sem finnur öll fög með þrjá áfanga. Væri reynt að nota *WHERE* klausu hér í stað *HAVING* myndi það leiða til villu!

Yfirlit yfir það hvernig *WHERE* og *HAVING* virka saman má sjá á mynd 4.10. Myndin sýnir líka í hvaða röð *FROM*, *WHERE*, *GROUP BY* og *HAVING* klausurnar koma fyrir í *SELECT* skipunum.

## 4.8 ORDER BY

Niðurstöður *SELECT* skipananna sem við höfum séð hafa ekki birst í neinni sérstakri röð. Þær hafa birst í þeirri röð sem þær eru geymdar í gagnagrunninum.<sup>15</sup>

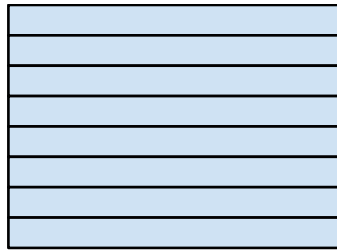
Til að láta gögnin birtast í ákveðinni röð má nota klausu sem heitir *ORDER BY*. *ORDER BY* klausan tekur við nafni á a.m.k. einum dálki og raðar línunum í röð eftir gildunum í þeim dálki.

Hægt er að raða línunum eftir tölum, bókstöfum, dagsetningum, hverju sem er sem hægt er að bera saman.<sup>16</sup>

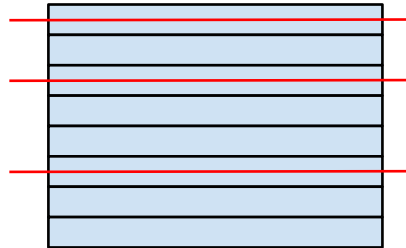
<sup>15</sup> Þetta er oftast innsetningarröð gagnanna. Ekki er samt alltaf hægt að treysta á það.

<sup>16</sup> Hér getur verið gott að rifja upp sam-  
anburði með röksegðum úr undirkafla 4.2.

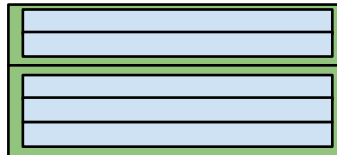
FROM klausan skilgreinir töflu sem valið er úr.



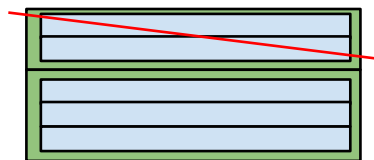
WHERE klausan síar burt línur.



GROUP BY klausan sameinar línur svo úr verða hópar af línum.



HAVING klausan síar burt hópa af línum.



Mynd 4.10: Lýsing á því hvernig FROM, WHERE, GROUP BY og HAVING klausurnar vinna saman.

Sé annað ekki tekið fram er línunum raðað í hækkandi röð. “Lægstu” gildin koma fremst í röðinni, “hæstu” gildin koma aftast. Röðinni má snúa við með því að bæta lykilorðinu DESC fyrir aftan dálkupptalninguna í ORDER BY klausunni.

Notkun ORDER BY klausunnar má sjá á sýnidæmum [4.24](#) til [4.26](#).

---

```
SELECT nafn, kennitala
FROM Nemendur
ORDER BY nafn;
```

---

Sýnidæmi 4.24: *SELECT* skipun sem velur nemendur úr töflunni í stafrófsröð eftir nafni með því að nota *ORDER BY* klausu.

---

```
SELECT nafn, kennitala
FROM Nemendur
ORDER BY nafn DESC;
```

---

Sýnidæmi 4.25: *SELECT* skipun sem velur nemendur úr töflunni í öfugri stafrófsröð.

---

```
SELECT onn, audkenni
FROM Afangar
ORDER BY onn, audkenni;
```

---

Sýnidæmi 4.26: *SELECT* skipun sem sýnir áfanga, fyrst raðaða eftir önninni sem þeir eru kenndir á, svo er stafrófsröð notuð til að raða áföngunum innbyrðis innan annanna.

## 4.9 LIMIT

Stundum þurfum við ekki allar upplýsingarnar úr töflu. Til að takmarka hversu margar línur birtast í niðurstöðu *SELECT* skipunar má nota síðustu klausuna sem við förum yfir í þessari bók - *LIMIT*.

*LIMIT* klausan tekur við einni eða tveimur tölum.

Fái *LIMIT* klausan eina tölu mun *SELECT* skipunin einungis sýna þann fjölda af línur í niðurstöðunni. Fyrstu línurnar úr niðurstöðunni eru valdar. Þannig myndi t.d. *LIMIT 5* sía allar línur úr niðurstöðunni nema þær fimm fyrstu.

Fái *LIMIT* klausan tvær tölur skilar *SELECT* skipunin ekki (endilega) fyrstu línunum úr niðurstöðunni. Klausan byrjar þá að telja á því línunúmeri sem fyrri talan segir til um, sú seinni segir til um hversu margar tölur eru valdar. Línunúmerin byrja að telja frá núlli. Þannig myndi t.d. *LIMIT 5, 10* hleypa í gegn línur 6 – 15 og *LIMIT 0, 5* myndi gera það sama og *LIMIT 5*.

Sýnidæmi [4.27](#) og [4.28](#) sýna notkun *LIMIT* klausunnar.

---

```
SELECT nafn, kennitala
FROM Nemendur
ORDER BY nafn
LIMIT 3;
```

---



---

```
SELECT nafn, kennitala
FROM Nemendur
ORDER BY nafn
LIMIT 10,1;
```

---

Sýnidæmi 4.27: *SELECT* skipun sem velur þá þrjá nemendur sem fremstir eru í stafrófsröðinni. Niðurstöðunum raðað með *ORDER BY*, síðan er öllum línunum eftir þá þriðju hent með *LIMIT* klausunni.

Sýnidæmi 4.28: *SELECT* skipun sem velur nemanda númer 11 í stafrófsröðinni.

#### 4.10 Uppbygging *SELECT* skipunarinnar

Við höfum farið yfir fjölmargar klausur sem tilheyra *SELECT* skipuninni.

Það skiptir máli í hvaða röð þessar klausur koma fram í *SELECT* skipun. Þær þurfa að koma fram í nákvæmlega þeirri röð sem við kynntum þær. Sú röð er:

1. Haus *SELECT* skipunarinnar
2. *FROM*
3. *WHERE*
4. *GROUP BY*
5. *HAVING*
6. *ORDER BY*
7. *LIMIT*

Sleppa má hverri af klausunum sem er<sup>17</sup>. Þetta breytir ekki röðinni - *WHERE* þarf alltaf að koma á eftir *FROM*, *ORDER BY* þarf alltaf að koma á eftir *GROUP BY*, og svo framvegis. Almennt getur *SELECT* skipun með þeim atriðum sem við höfum kynnst því litið út á þann hátt sem sjá má á sýnidæmi 4.29.

<sup>17</sup> *HAVING* er þó mjög sjaldan notuð án *GROUP BY*. Ekki er hægt að sleppa sjálfu *SELECT* lykilordinu.

---

```

SELECT "dálkar"
FROM "Tafla"
WHERE "skilyrði"
GROUP BY "dálkar"
HAVING "skilyrði um hóplínur"
ORDER BY "dálkar"
LIMIT "tala eða tvær tölur"

```

---

Sýnidæmi 4.29: Uppbygging *SELECT* skipunar með þeim atriðum sem koma fyrir í þessum (kafla 4).

#### 4.11 Yfirlit

Í þessum kafla fórum við yfir *SELECT* skipunina í MySQL.

- *SELECT* skipun er notuð til að ná í upplýsingar.
- Þegar *FROM* klausa fylgir *SELECT* skipun er klausan notuð til að skilgreina hvaðan upplýsingarnar koma.
- *WHERE* klausan síar burt upplýsingar frá *FROM* klausunni. Til þess eru notaðir samanburðir.
  - Hægt er að nota alls kyns samanburði í *WHERE* klausu svo lengi sem samanburðurinn einfaldast niður í "satt" eða "ósatt".
- Scalar föll í MySQL taka inn eitt eða ekkert gildi og skila einu gildi. Sé dálkur af gildum settur inn í scalar fall sem tekur við einu gildi er fallið notað á hvert gildi í dálkinum fyrir sig.
- *GROUP BY* klausan sameinar gögn eftir sameiginlegum einkennum svo úr verða "hóplínur". Þetta er oftast gert til þess að nota samsteypuföll á hóplínurnar.
- Samsteypuföll taka inn hóp af gildum (dálk, hóplínu) og skila upplýsingum um hópinn.
- *HAVING* klausan virkar eins og *WHERE*, nema fyrir hóplínur.
- *ORDER BY* klausan raðar línur í niðurstöðum *SELECT* skipunar í röð eftir einum eða fleiri dálkum.
- *LIMIT* klausan setur takmörk á hversu margar línur geta birst í niðurstöðunum.

## 5

# Að setja upp gagnagrunn

Við höfum kynnst því hvernig búa má til töflur (kafla 3) og hvernig ná má í gögn úr töflunum (kafla 4).

Þessi vitneskja getur komið okkur furðu langt - en ekki alveg nógu langt. Nú skulum við skoða hvernig vinna má með stærri töflur og margar töflur í sama grunninum.

### 5.1 Lyklar

Lyklar<sup>1</sup> eru mikilvægir í öllum skilvirkum gagnagrunnum. Lyklar eru m.a. notaðir til að gera fyrirspurnir hraðvirkari og til að tengja töflur saman.

<sup>1</sup> e. *key* eða *index*

Lyklar eru stundum kallaðir *vísar*. “Lykill” og “vísir” þýða það sama þegar kemur að MySQL.

#### *Almennt um lykla - KEY/INDEX*

Ímyndum okkur að við séum að leita að bók sem geymd er á mjög frumstæðu bókasafni. Á þessu bókasafni eru nefnilega engir titlar á kili bókanna og allar bækurnar líta eins út. Erfitt verk, ekki satt? Að meðaltali þyrftum við að fara í gegnum hálf bókasafnið áður en við rekumst á bókina sem við viljum.

Þetta er verkið sem stendur frammi fyrir gagnagrunnskerfum í hvert skipti sem við notum þau til að leita að gögnum í dálki sem ekki er með lykil. Gagnagrunnskerfi geta ekki borið saman gögn án þess að lesa þau. Sem betur fer eru tölvur mjög fljótar að bryðja sér leið í gegnum mikið magn af gögnum - en við getum gert betur en að geyma allar upplýsingarnar okkar ómerktar og óráðar.

Það að setja vísi á dálk í gagnagrunnstöflu samsvarar því að búa til



bókasafnskerfi sem getur sagt okkur í hvaða hillum og hvar í hillunum við getum fundið hverja bók. Það gefur auga leið að þetta gerir leitina auðveldari.

Hver tafla getur haft marga lykla.

### *Einkvæmir lykilar - UNIQUE KEY*

Hægt er að setja þá takmörkun á lykil að öll stök sem honum tilheyra þurfi að vera einstök.

Slíkur lykill, sem kalla má einkvæman lykil<sup>2</sup>, hefur þá ekki einungis það hlutverk að gera leit í gagnagrunninum auðveldari, heldur einnig það hlutverk að passa upp á að gögnin séu einstök.

<sup>2</sup> e. *unique key* eða *unique index*

Þetta þýðir að það að reyna að setja inn endurtekin gögn í dálk (dálka) sem er með einkvæman lykil veldur villu. Einkvæmur lykill myndar sem sagt takmörkun á því hvaða gildi mega fara inn í dálka sem hann nær yfir. Talað er um að einkvæmir lykilar myndi skorðu<sup>3</sup> á dálkinn eða dálkana.

<sup>3</sup> e. *constraint*

Íslenskar kennitölur eru dæmi um gögn sem oft væri gott að vera með einkvæman lykil á. Það að fletta upp kennitölum er algeng aðgerð og við vitum að allar kennitölur eiga að vera einstakar.

### *Aðallyklar - PRIMARY KEY*

Við höfum kynnst aðallyklum áður. Við sáum útskýringu á hvernig búa má til slíkan lykil í undirkafla 3.7.

Nú getum við séð að aðallykill er ekkert annað en sérstök gerð af einkvæmum lykli. Aðallykill er lykill sem hefur það sérstaka hlutverk að einkenna hverja línu fyrir sig, svo að hægt sé að vísa í hana á ótvíræðan hátt.

Þar sem að aðallykill töflu á að geta einkennt hverja línu er oftast best að lykillinn sé ekki byggður á gögnunum í línunni<sup>4</sup>. Einnig er gott að aðallykillinn sé lítill og einfaldur<sup>5</sup>.

<sup>4</sup> af því að við viljum alls ekki þurfa að breyta aðallykli línu ef að gögnin breytast

<sup>5</sup> Vegna þess að hann er oftast mikið notaður af gagnagrunnskerfinu.

Kennitölur eru þess vegna ekki sérstaklega góðir aðallyklar. Þær geyma upplýsingar um einstaklinginn (fæðingardagsetningu hans), þær geta breyst (þó það sé sjaldgæft) og þær eru langar (sem er tímafrekt að lesa). Þess vegna eru romsur á borð við þá sem við höfum séð í `CREATE TABLE` skipunum, `id INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT` mikið notaðar til að skilgreina aðallykla. Þær búa til lykla sem

- Auðkenna línuna fullkomlega

- Eru heiltölur (og þar með litlar og auðveldar í vinnslu)
- Eru aldrei NULL
- Og eru óháðir gögnunum.

Þó að tafla geti verið með marga lykla, þá er hver tafla aðeins með einn aðallykil.

---

```
CREATE TABLE Nemendur
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  nafn VARCHAR(100),
  kennitala CHAR(11),
  innritun DATE
);
```

---

Sýnidæmi 5.1: Til upprifjunar: aðallykill skilgreindur sem hluti af *CREATE TABLE* skipun. Þessi skipun býr til töflu 4.1, sem við notuðum mikið í síðasta kafla.

## 5.2 Margar töflur í sama gagnagrunninum

Við komumst að því strax í kafla 2 að gagnagrunnur getur innihaldið margar töflur. Hingað til höfum við samt verið að vinna með töflur eina í einu, óháð hvor annarri.

Lítum á dæmi um hvernig tvær töflur sem eru staddar í sama gagnagrunni geta tengst. Skoðum aftur áfangatöfluna okkar frá því í síðasta kafla (5.1).

numer	audkenni	fag	onn
1	FOR1A3U	Forritun	1
2	VSH1A3U	Vefhönnun	1
3	GSÖ1G2U	Notkun gagnasafna	1
4	TÆK1A1U	Tölvutækni	1
5	FOR1B3U	Forritun	2
6	VSH2A3U	Vefhönnun	2
7	GSÖ1F2U	Notkun gagnasafna	2
8	TÆK2A3U	Tölvutækni	2
9	FOR2B2U	Forritun	3
10	VSH2B2U	Vefhönnun	3
11	GSÖ2B2U	Notkun gagnasafna	3
12	TÆK2B2U	Tölvutækni	3
13	GRU2L4U	Lokaverkefni grunndeildar	3

Tafla 5.1: Tafla 4.6 endurtekin.

Sú tafla er ágæt, en við getum gert hana betri með því að skipta henni upp í tvær töflur sem eru tengdar saman.<sup>6</sup> Tökum eftir því að nafnið á hverju fagi er endurtekið í dálkinum *fag*.

Við gætum komist hjá þessari endurtekningu með því að geyma nöfnin á fögunum í sérstakri töflu.<sup>7</sup> Sú tafla gæti verið eins og tafla 5.2. Þar má sjá að við höfum gefið hverju fagi númer. Gefum okkur það að dálkurinn sem inniheldur þau númer sé aðalylkill (sjá undirkafla 5.1). Þá getum við notað númerin til þess að vísa í línurnar. Þetta hefur verið gert á töflu 5.3, nöfnunum á fögunum hefur verið skipt út fyrir númer þeirra í töflu 5.2.

numer	nafn
1	Forritun
2	Vefhönnun
3	Notkun gagnasafna
4	Tölvutækni
5	Lokaverkefni grunndeildar

numer	audkenni	fagNumer	onn
1	FOR1A3U	1	1
2	VSH1A3U	2	1
3	GSÖ1G2U	3	1
4	TÆK1A1U	4	1
5	FOR1B3U	1	2
6	VSH2A3U	2	2
7	GSÖ1F2U	3	2
8	TÆK2A3U	4	2
9	FOR2B2U	1	3
10	VSH2B2U	2	3
11	GSÖ2B2U	3	3
12	TÆK2B2U	4	3
13	GRU2L4U	5	3

Hægt er að búa til mjög stórar fjölskyldur af töflum með því að tengja þær saman á þennan hátt með sameiginlegum gildum.

### 5.3 Aðkomulyklar - FOREIGN KEY

Þegar gildi í dálki eiga að vísa í línur annarrar töflu er hægt að tryggja sambandið með því að búa til svokallaðan aðkomulykil<sup>8</sup>.

Aðkomulykill myndar formlegt samband á milli tveggja taflna. Taflan sem inniheldur gögnin sjálf er kölluð foreldrið í sambandinu. Taflan

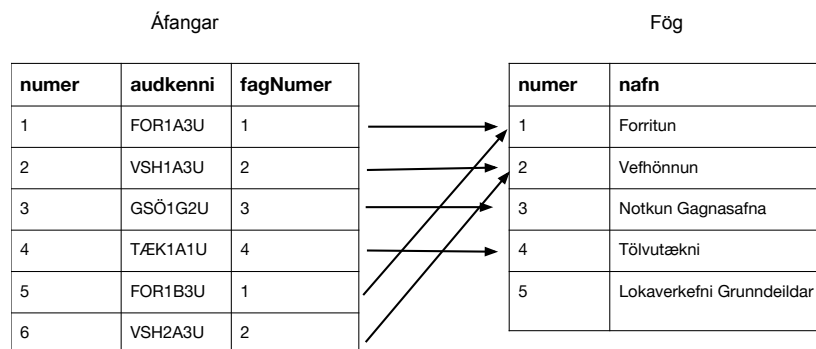
<sup>6</sup> Af hverju er betra að vera ekki með svona endurtekningar? Hér má til dæmis nefna minni plásseyðslu og hversu mikið auðveldara verður að uppfæra gildin í töflunum séu engar endurtekningar til staðar.

<sup>7</sup> Það sem við erum að gera hér er kallað að "normalísera" (e. *normalize*) gagnagrunninn. Normalisering er viðfangsefni fyrir lengra komna.

Tafla 5.2: Fög, sem áður voru í gömlu áfangatöflunni (4.6/5.1). Þessa töflu má tengja við endurbættu áfangatöfluna - töflu 5.3.

Tafla 5.3: Áfangataflan, þar sem nöfnunum á fögunum hefur verið skipt út fyrir númer þeirra, sem birtast í töflu 5.2.

<sup>8</sup> e. *Foreign key*.



Mynd 5.1: Sýnir hvernig töflur 5.2 og 5.3 tengjast saman á dálkunum *fagNumer* í áfangatöflunni og *numer* í fagtöflunni. (Ath: Töflurnar eru ekki sýndar í heilu lagi.)

sem inniheldur vísunina í gögnin er kölluð barnið<sup>9</sup>. Í dæminu okkar á undan er 5.2 því foreldrið og 5.3 barnið.

Aðkomulykillinn passar upp á að barnið geti ekki vísað í gögn sem ekki eru til í foreldrinu. Sé reynt að nota `INSERT` skipun á barnið til að setja inn vísun í línu sem ekki er til í foreldrinu veldur það villu sé lykillinn rétt upp settur.<sup>10</sup>

Aðkomulykillinn er skilgreindur í `CREATE TABLE` skipun barnsins.

Mikilvægt er að dálkarnir í barninu og foreldrinu passi saman m.t.t. gagnagerðar (3.3). Oftast er tengt saman á heiltöludálkum (sem þurfa að vera af sömu stærð).

Dálkurinn í foreldrinu sem aðkomulykillinn vísar í þarf að hafa aðalkeyl eða einkvæman lykil. Væri svo ekki gæti gagnagrunnskerfið ekki verið visst um hvaða línu í foreldrinu vísunin í barninu ætti við.

Sjá má hvernig skilgreina mætti samband á milli taflna 5.2 og 5.3 sem hluta af `CREATE TABLE` skipunum á sýnidæmi 5.2.

## 5.4 Margar samtengdar töflur

Nú þegar við kunnum að tengja saman tvær töflur er ekkert sem hindrar okkur í að tengja saman eins margar töflur og við viljum.

Sýnidæmi 5.4 til 5.5 bæta töflum við gagnagrunninn “taeknisniskolinn” sem við bjuggum til fyrir löngu (sýnidæmi 2.1). Þær töflur passa við töflurnar sem búnar voru til í dæmi 5.2.

Hér höfum við ekki farið yfir hvernig vinna má með gögn úr mörgum töflum (velja úr mörgum töflum í einu). Slík gagnavinnsla er viðfangsefni kafla 6.

<sup>9</sup> e. *parent table* annars vegar og *child table* hins vegar

<sup>10</sup> Aðkomulyklar eru því önnur gerð af skorðu (constraint).

---

```

CREATE TABLE Fog
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  nafn VARCHAR(30)
);

CREATE TABLE Afangar
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  audkenni CHAR(7),
  fagNumer INTEGER NOT NULL,
  onn INTEGER,
  FOREIGN KEY (fagNumer) REFERENCES Fog(numer)
);

```

---

Sýnidæmi 5.2: Tengdar töflur. Aðkomulykillinn er skilgreindur í síðustu línu seinni *CREATE TABLE* skipunarinnar. Línan býr til aðkomulykil fyrir dálkinn *fagNumer* sem vísar í dálkinn *numer* í töflunni *Fog*.

---

```

CREATE TABLE Foreldri
(
  adallykillForeldris INTEGER NOT NULL PRIMARY KEY
);

CREATE TABLE Barn
(
  adallykillBarns INTEGER NOT NULL PRIMARY KEY,
  adkomulykill INTEGER NOT NULL,
  FOREIGN KEY (adkomulykill)
    REFERENCES Foreldri(adallykillForeldris)
);

```

---

Sýnidæmi 5.3: Meira “abstract” dæmi um tengdar töflur. Hér koma hlutverk taflanna og dálkanna fram sem heiti, til að sýna sambandið betur.

---

```

CREATE TABLE Starfsmenn
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  nafn VARCHAR(50),
  starfsheiti VARCHAR(20),
  netfang VARCHAR(20)
);

CREATE TABLE Nemendur
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  nafn VARCHAR(100),
  kennitala CHAR(11),
  innritun DATE,
  umsjonarkennaraNumer INTEGER NOT NULL,
  FOREIGN KEY (umsjonarkennaraNumer)
    REFERENCES Starfsmenn(numer)
);

```

---

Sýnidæmi 5.4: Okkar gömlu góðu starfsmanna- og nemendatöflur (töflur 2.1 og 4.1) uppfærðar til að vera tengdar saman. Hér höfum við bætt dálkinum *umsjonarkennari* við nemendatöfluna, sem inniheldur vísun í starfsmannatöfluna. Hver nemandi hefur sem sagt einn starfsmann sem er umsjónarkennari viðkomandi nemanda.

---

```

CREATE TABLE Hopar
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  nafnHops VARCHAR(10),
  hamarksFjoldi INTEGER,
  afangaNumer INTEGER NOT NULL,
  kennaraNumer INTEGER NOT NULL,
  FOREIGN KEY (afangaNumer)
    REFERENCES Afangar(numer),
  FOREIGN KEY (kennaraNumer)
    REFERENCES Starfsmenn(numer)
);

```

---

Sýnidæmi 5.5: Hver áfangi getur verið kenndur oftast en einu sinni á önn, nemendum er þá skipt upp í hópa. Hér er tafla sem geymt gæti upplýsingar um hópa. Hópur getur hér verið með nafn (í Tækniskólanum er þetta oftast bara tala) og hámarksfjölda nemenda sem eigin ein-kenni. Síðan vísum við í áfangatöfluna til að skrá upplýsingar um hvaða áfanga er hér verið að kenna, og starfsmannatöfluna til að skrá upplýsingar um hver kennir hópinn.

## 5.5 Tengingar

Samtenging á töflum getur táknað þrjár gerðir af samböndum. Gerðirnar eru *1-1 sambönd*, *1-N sambönd* og *N-N sambönd*. Lítum stuttlega á þær.

### 1-N: Einn tengdur í marga

Lítum fyrst á gerð 1 – Nþ Kalla mætti þá sambandsgerð “einn-í-marga”<sup>11</sup> samband (táknið *N* þýðir hér “margir”). Þetta er sú gerð af samböndum/tengingum sem við sjáum oftast.

<sup>11</sup> e. *one-to-many*

Skoðum dæmi 5.2 aftur, með það fyrir augum að hér sé verið að tengja “einn-í-marga”.

Ef við greinum sambandið á milli taflanna tveggja aðeins, þá sjáum við að hver áfangi tilheyrir nákvæmlega einu fagi. Enginn áfangi tilheyrir tveimur fögum. Hins vegar getur hvert fag samanstðið af mörgum áföngum. Öll sambönd á milli taflna þar sem hver lína í annarri töflunni getur tengst mörgum línur í hinna töflunni, en ekki öfugt, eru 1 – *N* sambönd.

Í 1 – *N* sambandi er aðkomulykillinn settur í “margir” töfluna - barnið í sambandinu.

### 1-1: Einn tengdur í einn

1 – 1 tengingar eru einfaldar tengingar. Við gætum kallað þær “einn-í-einn” tengingar. Ef 1 – 1 tenging er á milli taflna samsvarar hver lína í annarri töflunni nákvæmlega einni línu í hinna töflunni, og öfugt.

Það að vera með 1 – 1 tengingu á milli taflna er mjög svipað og að vera bara með eina töflu (hver lína í annarri töflunni er í raun bara framhald af línu í fyrri töflunni). Við sjáum þessa gerð tenginga oftast þegar gagnagrunnshönnuður sér að gott væri að skipta töflu með mörgum dálkum í nokkrar minni.

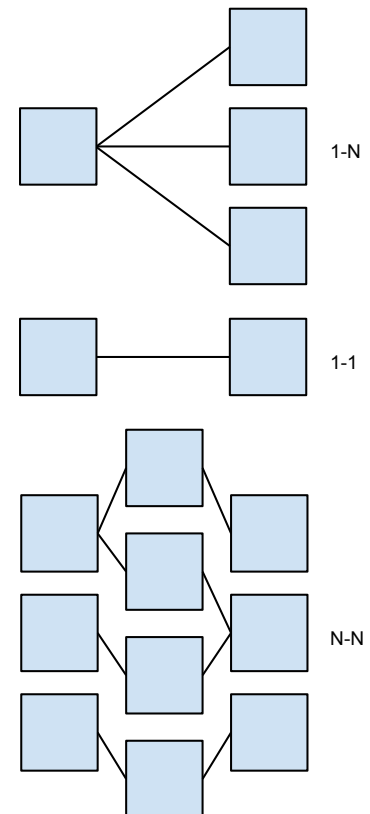
Í 1 – 1 sambandi er aðkomulykill settur frá einum einkvæmum lykli til annars.

### N-N: Margir tengdir í marga

Síðasta gerð tenginga er *N – N* gerðin (“margir-í-marga”).

Til að sjá dæmi um hvernig *N – N* tenging gæti komið upp getum við skoðað sambandið á milli hugtakanna “nemendur” og “hópar” (við

Mynd 5.2: Tengingar í sambandsgerðunum þremur.



þjuggum til töflur fyrir þessi hugtök í sýnidæmum 5.4 og 5.5). Hver nemandi getur tilheyrt mörgum hópum. Hver hópur inniheldur meira en einn nemanda.

Ekki er hægt að tákna þetta samband með aðkomulyklum eingöngu svo vel sé. Til að tákna slík sambönd bætum við við sérstakri töflu. Þessi tafla, sem við getum kallað tengitöflu eða millitöflu, hefur það hlutverk að halda utan um sambandið. Hver lína í slíkri töflu táknar eitt samband.

Sýnidæmi 5.6 sýnir hvernig búa mætti til tengitöflu sem skilgreinir  $N - N$  samband á milli nemenda og hópa.

---

```
CREATE TABLE HopSkraningar
(
  numer INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
  nemandaNumer INTEGER NOT NULL,
  hopNumer INTEGER NOT NULL,
  FOREIGN KEY (nemandaNumer) REFERENCES Nemendur (numer) , í hóp.
  FOREIGN KEY (hopNumer) REFERENCES Hopar (numer)
);
```

---

Sýnidæmi 5.6: Sambandið á milli hópa og nemenda táknað með tengitöflu. Taflan hefur tvo aðkomulykla. Hver lína í töflunni táknar eina skráningu nemanda

## 5.6 Að sjá yfirlit gagnagrunns í MySQL Workbench

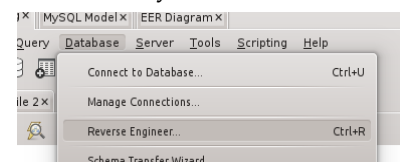
Eftir því sem gagnagrunnar stækka verður erfiðara að átta sig á tengingunum á milli taflnanna sem honum tilheyra.

MySQL Workbench getur sem betur fer aðstoðað okkur við að halda utan um gagnagrunna. Meðal annars getur hann búið til mynd af gagnagrunninum sem sýnir töflur gagnagrunnsins, dálka þeirra, lykla og sambönd.

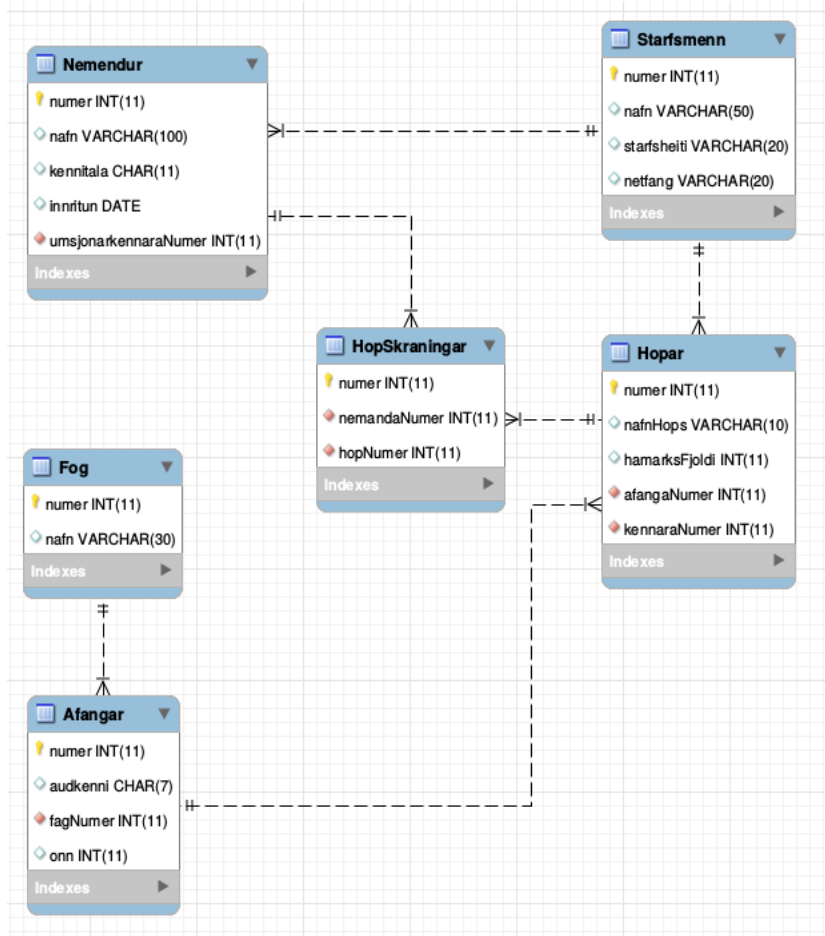
Til að búa til slíka mynd út frá gagnagrunni sem þegar er til má velja valkostinn “Reverse Engineer” undir “Database” í aðalvalmynd MySQL Workbench (sjá mynd 5.3) eða ýta á **CTRL+R** og fylgja leiðbeiningunum sem þar birtast.

Útkoman verður lík mynd 5.4.

Mynd 5.3: Reverse Engineer takkinn í MySQL Workbench.







Mynd 5.4: Yfirlitsmynd fyrir Tækniskólagagnagrunninn, búin til af MySQL Workbench.

### 5.7 (Ekki) meira um lykla

Lyklar eru stórt viðfangsefni sem ekki er hægt að snerta á nema að mjög litlu leyti í þessari bók. Við munum ekki kafa dýpra í þá hér, heldur munum við láta þá bíða bóka og námskeiða fyrir lengra komna.

### 5.8 Yfirlit

Í þessum kafla skoðuðum við lykla og tengingar á milli taflna.

- Það að setja lykil á dálk hjálpar gagnagrunnskerfi að vinna með hann á hagkvæman máta.
  - Einn lykill getur náð yfir marga dálka.
  - Hver tafla getur haft marga lykla.
- Lykill getur verið “einkvæmur” (unique). Engin tvö stök í einkvæmum lykli eru eins.
- Hver tafla getur verið með einn aðallykil, sem er einkvæmur lykill. Hann er notaður til að auðkenna hverja línu í töflunni fyrir sig.
- Aðkomulykill er sérstök gerð af lykli sem skilgreinir samband á milli tveggja taflna.
- Aðkomulyklar og einkvæmir lyklar eru skorður (constraints). Villa kemur upp sé reynt að setja inn gögn í dálk sem tilheyrir slíkum lykli ef gögnin uppfylla ekki skilyrðin sem lykillinn setur.
- Sambönd á milli taflna geta verið á þrjá vegu:  $1 - N$ ,  $1 - 1$  og  $N - N$ .
  - Í  $1 - N$  sambandi er aðkomulykillinn í “margir” hluta sambandsins (barnið).
  - Í  $1 - 1$  sambandi tengir aðkomulykillinn saman tvo einkvæma lykla.
  - Í  $N - N$  sambandi eru tveir aðkomulyklar í sérstakri tengitöflu.
- Hægt er að fá yfirlitsmynd af gagnagrunni með því að nota Reverse Engineer tólið í MySQL Workbench.

## 6

### *Að sameina gögn*

Við höfum komist að því hvernig búa má til töflur sem vísa hver í aðra. Við höfum líka komist að því hvernig ná má í upplýsingar úr einni töflu.

Nú skulum við líta á hvernig búa má til skipanir sem velja upplýsingar úr fleiri en einni töflu. Til þess þurfum við að læra hvernig stækka má FROM klausuna svo að hún geti meðhöndlað margar töflur.

#### 6.1 *Nöfn dálka*

Áður en lengra er haldið skulum við skoða betur dálitla ónákvæmni sem við höfum sætt okkur við hingað til.

Þegar við höfum vísað í dálk hefur okkur dugað að skrifa einfaldlega nafn hans. Við höfum ekki séð mikið að því að skrifa WHERE klausur á borð við WHERE numer = 11.

Þetta verður hins vegar erfiðara þegar við vinnum með margar töflur í einu. Hvað ef við erum að vinna með tvær töflur þar sem báðar töflurnar eru með dálk sem heitir numer?

Til að passa upp á að gagnagrunnskerfið viti hvaða dálk við eigum við þurfum við oft að taka fram í hvaða töflu dálkurinn sem við erum að nefna er. Þetta er gert með því að skipta út nafninu á dálkinum fyrir “fullt nafn” hans. Til að fá fullt nafn dálks skrifum við fyrst nafn töflunnar sem hann er í, svo punkt, svo nafn dálksins. Þannig má vísa í dálkinn a í töflunni A með því að skrifa A.a.

Notkun á fullu nafni dálks má sjá á sýnidæmi [6.1](#).

---

```

SELECT audkenni
FROM Afangar
WHERE numer = 1;

SELECT Afangar.audkenni
FROM Afangar
WHERE Afangar.numer = 1;

```

---

Sýnidæmi 6.1: Tvær *SELECT* skipanir sem gera það sama - velja auðkenni áfanga úr áfanga-töflunni þar sem raðnúmer línunnar er 1. Munurinn er sá að í seinni skipuninni er tekið fram í hvaða töflu dálkurinn *numer* er.

### Að endurnefna - AS

Það að vísa í löng heiti getur verið óþjálmt (eða jafnvel ómögulegt). Til að vinna með dálka og töflur undir öðru nafni má nota lykilorðið *AS*. Þetta býr til “aukanafn”<sup>1</sup> fyrir fyrirbrigðið sem unnið er með. Gamla nafninu er ekki breytt í gagnagrunnskerfinu, *AS* býr bara til nýtt nafn til að nota tímabundið.

<sup>1</sup> e. *alias*

Sjá má endurnefndan dálk á sýnidæmi 6.2.

---

```

SELECT Afangar.audkenni AS afangi
FROM Afangar;

```

---

Sýnidæmi 6.2: Endurnefning dálks. Dálkurinn *audkenni* mun birtast sem *afangi* í niðurstöðum þessarar skipunar.

## 6.2 Að velja úr meira en einni töflu - JOIN

Til að segja hvaðan upplýsingarnar sem við ætlum að vinna með í *SELECT* skipun eiga að koma notum við *FROM* klausu.

Hingað til hefur *FROM* klausan ekki innihaldið annað en orðið *FROM* og nafnið á töflunni. Þetta er villandi einfalt. Rifjum upp hvað við sögðum þegar við kynntumst *FROM* klausunni fyrst (undirkafla 4.1):

*FROM* klausa lýsir því hvaðan upplýsingar koma.

Þessi lýsing getur innihaldið meira en eina töflu. Almennt séð myndar *FROM* klausan sem sagt *mengi* upplýsinga sem hægt er að vinna með.

Til að skilgreina mengi sem tekur upplýsingar sínar úr tveimur eða fleiri töflum bætum við lykilorðinu *JOIN* (ásamt fleiri lykilorðum) við *FROM* klausuna.

## INNER JOIN

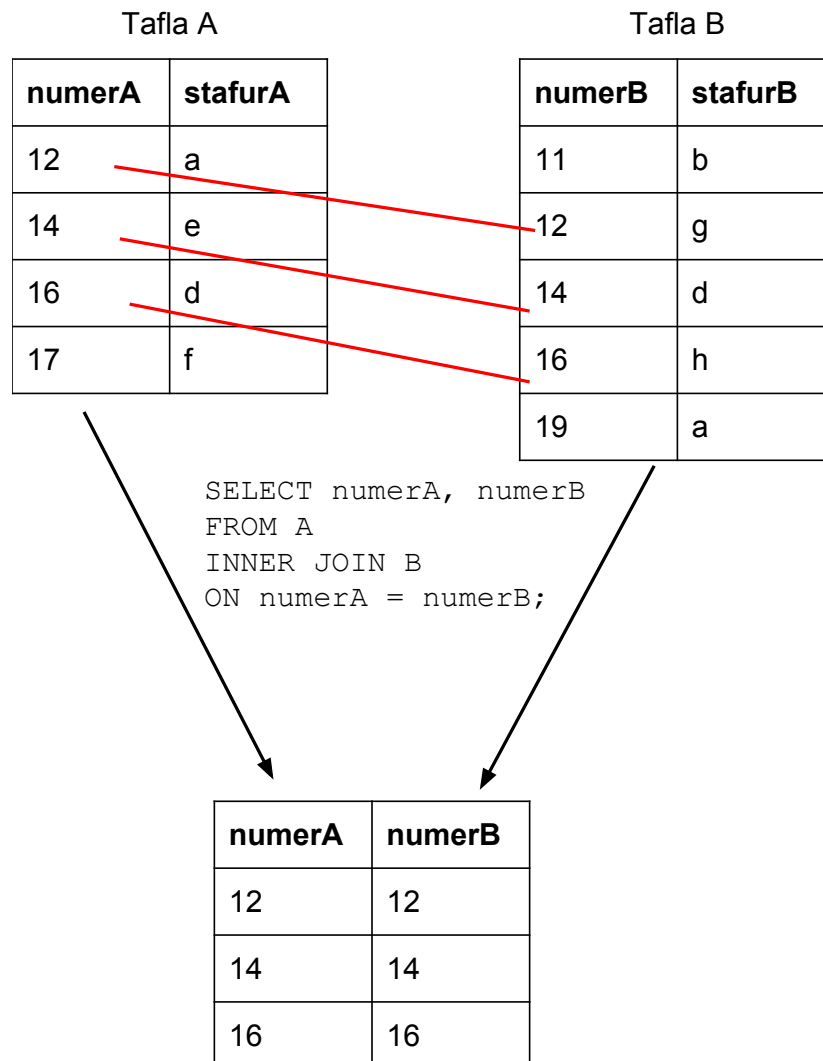
INNER JOIN er sú leið sem langmest er notuð til að ná upplýsingum úr mörgum töflum.

Til að skilja INNER JOIN betur skulum við rifja upp hvernig töflur voru tengdar saman með aðkomulyklum (undirkaflí 5.3). Aðkomulyklar tengja töflur saman með því að halda utan um gögn sem eru sameiginleg í tveimur töflum. INNER JOIN nýtir sér þessi sameiginlegu gögn til að smíða eitt stórt mengi sem vinna má með.

Lítum á mynd 6.1. Þar má sjá gildi í tveimur töflum borin saman. Þessi samanburður á sér stað í ON hluta skipunarinnar. Hann er líkur samanburði í WHERE klausu, en hefur það sérstaka hlutverk að bera saman línur í tveimur mismunandi töflum.<sup>2</sup>

<sup>2</sup> Venjulega er þetta jafnt-og samanburður, til að finna þær línur sem eru *eins* í báðum töflum.

Mynd 6.1: Sýnir hvernig *SELECT* skipun með *INNER JOIN* velur einungis þær línur úr töflunum *A* og *B* sem dálkarnir *numerA* og *numerB* eiga sameiginlegar.



Fyrir utan það að FROM klausan er stærri, þá virkar SELECT skipun sem velur úr fleiri en einni töflu alveg eins og SELECT skipun sem vinnur bara með eina töflu. Hægt er að velja hvaða dálk úr töflunum tveimur sem er.

Sjá má raunverulegri notkun á INNER JOIN á sýnidæmi 6.3.

---

```
SELECT Afangar.audkenni, Fog.nafn AS nafnFags
FROM Fog
INNER JOIN Afangar
ON Fog.numer = Afangar.fagNumer;
```

---

Sýnidæmi 6.3: *SELECT* skipun sem velur úr töflunum *Fog* og *Afangar*. Niðurstöðurnar má sjá á mynd 6.2.

#	audkenni	nafnFags
1	FOR1A3U	Forritun
2	FOR1B3U	Forritun
3	FOR2B2U	Forritun
4	VSH1A3U	Vefhönnun
5	VSH2A3U	Vefhönnun
6	VSH2B2U	Vefhönnun
7	GSÖ1G2U	Notkun Gagnasafna
8	GSÖ1F2U	Notkun Gagnasafna
9	GSÖ2B2U	Notkun Gagnasafna
10	TÆK1A1U	Tölvutækni
11	TÆK2A3U	Tölvutækni
12	TÆK2B2U	Tölvutækni
13	GRU2L4U	Lokaverkefni grunndeildar

Mynd 6.2: Niðurstaða *SELECT* skipunarinnar í dæmi 6.3.

Hægt er að tengja saman meira en tvær töflur með því að skrifa mörg JOIN í sömu FROM klausunni. Sjá sýnidæmi 6.4.

Þegar valið er úr mörgum töflum má líta á það sem svo að sífellt sé verið að stækka það mengi sem valið er úr, eina töflu í einu. Passa þarf upp á að tenging sé á milli allra taflanna sem taldar eru upp. Lína þarf að uppfylla öll skilyrðin í ON hlutum skipunarinnar til að geta birst í niðurstöðunum.

---

```

SELECT Hopar.nafnHops, Afangar.audkenni,
       Starfsmenn.nafn
FROM Afangar
INNER JOIN Hopar
ON Afangar.numer = Hopar.afangaNumer
INNER JOIN Starfsmenn
ON Hopar.kennaraNumer = Starfsmenn.numer;

```

---

Sýnidæmi 6.4: *SELECT* skipun sem velur nöfn hópa, hvaða áföngum hóparnir tilheyra og hver kennir þá, úr töflunum *Hopar*, *Afangar* og *Starfsmenn*.

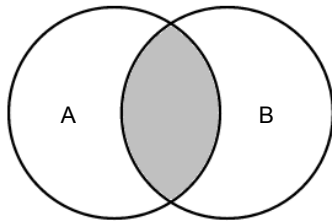
## ÖNNUR JOIN

Til að skilja JOIN á milli tveggja taflna er gott að hugsa um töflurnar sem verið er að tengja saman sem mengi. Það að finna INNER JOIN af töflunum er þá svipað því að finna sniðmengi taflanna - að finna allar línur sem einkennast af því að töflurnar tvær eigi stökin sameiginleg.

```

SELECT [dálkar]
FROM A
INNER JOIN B
ON A.a = B.b

```

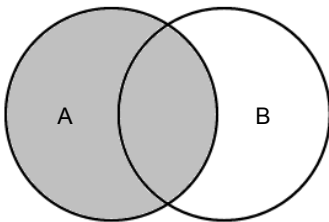


Mynd 6.3: Líta má á töflur sem mengi af upplýsingum.

```

SELECT [dálkar]
FROM A
LEFT OUTER JOIN B
ON A.a = B.b

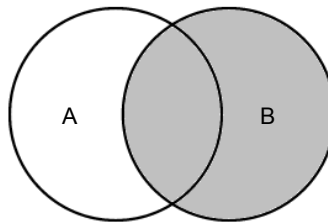
```



```

SELECT [dálkar]
FROM A
RIGHT OUTER JOIN B
ON A.a = B.b

```



Þessi mengjahugsunarháttur opnar fyrir möguleikann á fleiri gerðum af JOIN. Svokölluð OUTER JOIN geta fundið allar línur í einni töflu ásamt tilsvarendi línu í annarri töflu - ef tilsvarendi lína er til. Í MySQL eru til LEFT OUTER JOIN og RIGHT OUTER JOIN, sem virka á sama hátt fyrir utan það að þær virka á mismunandi töflur.

Sum gagnagrunnskerfi (en ekki MySQL) hafa líka svokallað `FULL OUTER JOIN`, sem velur allar línur úr tveimur töflum óháð því hvort að stökin eigi eitthvað sameiginlegt.

### 6.3 Undirfyrirspurnir

Þegar við vinnum með gagnagrunna getum við lent í því að þurfa að nota upplýsingarnar sem við fáum úr fyrirspurn til þess að geta klárað að skrifa aðra fyrirspurn.

Sem dæmi um slíkar aðstæður getum við litið lengst aftur á töflu 4.7, sem geymir einkunnir nemenda. Í kaflanum sem taflan var sett fram í (kafla 4.6) sáum við dæmi um hvernig við getum skrifað fyrirspurn til að finna meðaleinkunn nemenda í töflunni.

---

```
SELECT AVG(einkunn)
FROM Einkunnir;
```

---

Sýnidæmi 6.5: Dæmi 4.22 endurtekið, meðaleinkunn fundin.

En hvað ef við viljum skrifa fyrirspurn sem finnur nöfn þeirra nemenda sem eru hærri en meðaleinkunnin? Ekki dugur að keyra fyrri fyrirspurnina, skrifa niðurstöðuna niður og nota hana í annarri fyrirspurn. Við viljum ekki að fyrirspurnin okkar verði röng ef gögnin í töflunni breytist.

Til að sameina tvær fyrirspurnir af þessum toga breytum við fyrirspurninni sem við þurfum fyrst að fá upplýsingar úr í svokallaða undirfyrirspurn<sup>3</sup>. Við umlykjum þá fyrirspurn svigum og setjum hana á þann stað í hinna fyrirspurninni sem upplýsingarnar úr innri fyrirspurninni ættu að koma fyrir<sup>4</sup>.

Undirfyrirspurn má sjá á sýnidæmi 6.6.

---

```
SELECT nafn
FROM Einkunnir
WHERE einkunn > (
    SELECT AVG(einkunn)
    FROM Einkunnir
);
```

---

<sup>3</sup> e. *subquery*

<sup>4</sup> Ekki þarf að setja semíkommu á eftir undirfyrirspurn. Henni lýkur þegar sviganum sem umkringir hana er lokað.

Sýnidæmi 6.6: `SELECT` skipun sem notar undirfyrirspurn til að finna þá nemendur sem eru yfir meðaleinkunninni. Undirfyrirspurnin er í `WHERE` klausunni. Undirfyrirspurnin er alltaf keyrð á undan ytri fyrirspurninni.



### *Mismunandi hlutverk undirfyrirspurna*

Undirfyrirspurnir eiga það sameiginlegt að skila niðurstöðum sem ytri fyrirspurnin getur nýtt sér og að vera afmarkaðar af svigum. Þar fyrir utan geta þær verið af ýmsum toga.

- Undirfyrirspurnir geta skilað einu gildi, dálki af gildum eða heilli töflu af gildum.
- Undirfyrirspurnir geta valið gildi úr sömu töflu og ytri fyrirspurnin eða úr öðrum töflum.
- Undirfyrirspurnir geta komið fram á mörgum stöðum innan ytri fyrirspurnar, oftast í WHERE eða FROM klausu.

Undirfyrirspurnir geta haft sínar eigin undirfyrirspurnir, það er að segja, undirfyrirspurnir geta verið hver inni í annarri.<sup>5</sup>

Undirfyrirspurnir geta komið fyrir í SQL-skipunum sem ekki eru fyrirspurnir. Þar á meðal eru INSERT, UPDATE og DELETE skipanir.

<sup>5</sup> Ef undirfyrirspurnagryfjan er orðin mjög djúp getur verið ástæða til að stoppa og athuga hvort þær séu allar nauðsynlegar.

## 6.4 Yfirlit

Í þessum kafla kynntumst við nokkrum aðferðum til að sameina upplýsingar, sem geta verið úr mismunandi töflum.

- Til að vísa í ákveðinn dálk í ákveðinni töflu má fyrst skrifa nafn töflunnar, svo punkt, svo nafn dálksins.
- Búa má til tímabundin heiti með því að nota AS lykilorðið.
- Líta má á töflur sem mengi af upplýsingum.
- Það að framkvæma INNER JOIN á tvær töflur er líkt því að finna sniðmengi taflanna.
- Hægt er að setja fyrirspurn inn í aðrar SQL-skipanir með því að umlykja fyrirspurnina með svigum. Fyrirspurnin kallast þá undirfyrirspurn.

# 7

## Að uppfæra gagnagrunna

Fyrr í bókinni höfum við farið yfir fjórar grunnaðgerðir sem nauðsynlegar eru til gagnagrunnsvinnslu. Við kunnum að

- Búa til gagnagrunna og töflur (CREATE skipanir, kaflar 2.4 og 3.1)
- Setja gögn inn í töflur (INSERT skipunin, 3.2)
- Ná í gögn úr töflum (SELECT skipunin, kaflar 4 og 6 eins og þeir leggja sig)
- Eyða töflum og öllu sem í þeim er (DROP skipunin, kafli 3.8)

Þetta kemur okkur býsna langt. Þetta hefur hins vegar ekki gert okkur kleift að gera nokkrar breytingar á töflum eða þeim gögnum sem í þeim eru. Til þess þurfum við fleiri skipanir. Þær heita ALTER TABLE, UPDATE og DELETE.

### 7.1 DDL og DML

Áður en lengra er haldið skulum við staldra við og skoða þær skipanir sem við höfum þegar kynnst.

Þessar skipanir, CREATE, INSERT, SELECT og DROP skiptast í tvo flokka. Flokkarnir kallast *Data Definition Language* (DDL) og *Data Manipulation Language* (DML).<sup>1</sup>

DDL skipanir hafa áhrif á uppbyggingu gagnagrunnsins. Þær breyta, búa til og eyða gagnagrunnum, töflum og dálkum. CREATE skipanir og DROP skipunin eru DDL skipanir.

DML skipanir hafa áhrif á gögn í gagnagrunninum. Þær breyta, búa til, eyða og sýna innihald taflna. INSERT og SELECT eru DML skipanir.<sup>2</sup>

Mynd 7.1: Yfirlit yfir þær SQL-skipanir sem við höfum séð og flokkun þeirra í DDL og DML.

#### DDL skipanir

CREATE DATABASE  
CREATE TABLE  
DROP TABLE  
ALTER TABLE

#### DML skipanir

INSERT  
UPDATE  
DELETE  
SELECT

<sup>1</sup> Þessir flokkar hafa verið kallaðir *gagnaskilgreiningarmál* og *gagnameðferðarmál* á íslensku.

<sup>2</sup> SELECT skipunin er örlítið sérstök hvað DML skipanir varðar, þar sem hún hefur það ekki að aðalhlutverki að breyta gögnum. Þess vegna er oft fjallað um hana sérstaklega. Hún er DML skipun engu að síður.

Fleiri flokkar skipana eru til. Þær skipanir sem við förum yfir í þessari bók falla þó allar í þessa tvo.

## 7.2 Að breyta töflum

Hingað til höfum við ekki farið yfir leið til að breyta töflum. Það þýðir að í hvert skipti sem villa er gerð í töflu höfum við þurft að henda töflunni ásamt öllu því sem í henni er (`DROP TABLE`) og búa hana til upp á nýtt.

Þetta ferli gengur ekki mjög vel þegar um flókna gagnagrunna eða töflur með raunverulegum gögnum er að ræða. Við þurfum öflugra tól - við þurfum að geta breytt töflu eftir að hún hefur verið búin til.

Til þess að breyta töflu notum við skipunina `ALTER TABLE`. Við getum m.a. notað hana til þess að bæta við og eyða dálkum. Sjá sýnidæmi [7.1](#).

---

```
ALTER TABLE Tafla
ADD nyrDalkur INTEGER;

ALTER TABLE Tafla
DROP COLUMN gamallDalkur;
```

---

Sýnidæmi 7.1: Tvær `ALTER TABLE` skipanir. Sú fyrri bætir heiltöludálkinum *nyrDalkur* við töfluna *Tafla*. Sú seinni eyðir dálkinum *gamallaDalkur* úr sömu töflu.

`ALTER TABLE` breytir einingunum sem í gagnagrunninum eru, töflunum sjálfum. Þess vegna er `ALTER TABLE DDL`-skipun.

## 7.3 Að breyta gögnum

Fyrir getur komið að við þurfum að breyta töflum. Mun algengari en breytingar á töflum eru þó breytingar á gögnum.

Til að breyta gögnum má nota skipunina `UPDATE`. Hún setur ekki inn gögn sjálf (til þess notum við áfram `INSERT` skipanir), hún breytir einungis gögnum sem þegar eru til staðar í gagnagrunninum.

`UPDATE` skipun skiptist í þrjá aðalhluta<sup>3</sup>:

1. `UPDATE` lykilorðið og nafnið á töflu eða töflum, sem segir til um hvar gögnin er að finna.
2. `SET` klausa, sem telur upp þá dálka sem breyta á og nýju gildin sem eiga að fara í þá.

<sup>3</sup> Einnig er hægt að setja `ORDER BY` og `LIMIT` klausur í `UPDATE` skipanir. `ORDER BY` segir til um í hvaða röð `UPDATE` skipunin á að vinna, `LIMIT` setur takmark á það hversu mörgum línum skipunin má breyta.

3. WHERE klausa, sem setur skilyrði á þau gögn sem eiga að uppfærast. Eigi engin skilyrði að gilda um gögnin (sem sagt, ef uppfæra á allar viðeigandi línur) má sleppa WHERE klausnni.

Notkun UPDATE skipunarinnar má sjá á sýnidæmum 7.2 og 7.3.

---

```
UPDATE Nemendur
SET umsjonarkennaraNumer = 11
WHERE numer = 4;
```

---

Sýnidæmi 7.2: *UPDATE* skipun sem breytir nemendatöflunni. Hún skráir umsjonarkennara á nemanda númer 4. Umsjónarkennaranúmerið verður 11 eftir að skipunin hefur verið keyrð, óháð fyrra gildi.

---

```
UPDATE Nemendur
SET umsjonarkennaraNumer = 1;
```

---

Sýnidæmi 7.3: *UPDATE* skipun sem breytir *allri* nemendatöflunni með því að sleppa *WHERE* klausunni. Þessi skipun myndi skrá alla nemendur í umsjón hjá starfsmanni númer 1, Bjargeyju.

Gætum þess að rugla ekki saman ALTER og UPDATE. UPDATE breytir einungis gögnum, ekki töflum eða öðrum gagnagrunnshlutum. Hún er DML skipun.

## 7.4 Að eyða gögnum

Þegar eyða þarf gögnum án þess að hafa áhrif á töfluna eða töflurnar sem gögnin eru í má nota DELETE skipun. Henni svipar til UPDATE skipunarinnar, nema hvað engin SET klausa er til staðar - enda eru gögnin fjarlægð, ekki uppfærð.

---

```
DELETE FROM Afangar
WHERE audkenni = 'GSÖ1G2U';
```

---

Sýnidæmi 7.4: *DELETE* skipun sem eyðir áfanganum GSÖ1G2U úr áfangatöflunni. Notuð er *WHERE* klausa til að einangra áfangann líkt og í *UPDATE* skipuninni.

Sé *WHERE* klausunni sleppt í *DELETE* skipun er *öllum* línunum eytt. Þössum vandlega upp á að sú klausa sé til staðar og að hún sé rétt!

Líkt og *UPDATE* hefur *DELETE* einungis áhrif á gögn, ekki töflur eða gagnagrunna. Hún er DML skipun.

## 7.5 Viðhald gagnagrunna

Þegar gagnagrunnum og gögnum í þeim er breytt geta komið upp óvæntar aðstæður. Viðhald gagnagrunna er efni í bók út af fyrir sig, en við getum nefnt nokkur atriði stuttlega.

### *Aðkomulyklar og breytingar*

MySQL passar oftast upp á<sup>4</sup> að ekki sé hægt að breyta eða eyða gögnum sem aðkomulykill vísar á (gögn sem eru í “foreldrahlutverki” í aðkomulykilssambandi) svo að sambandið skemmist.

<sup>4</sup> Þetta er stillingaratriði sem hægt er að slökkva á.

Einfaldasta leiðin til að komast fram hjá þessu er að breyta eða eyða gögnunum í barninu á undan gögnunum í foreldrinu, svo að það “loforð” sem aðkomulykillinn gefur sé aldrei brotið.

Hægt er að setja upp aðkomulykla svo að slíkar breytingar séu sjálfvirkar. Þetta eru svokallaðar CASCADE aðgerðir.

### *Hreyfingar*

Þegar margar uppfærslur eða breytingar eru framkvæmdar í röð geta komið upp villur ef ferlið er truflað áður en því er lokið.

Til að tryggja það að breytingar sem eru háðar hver annarri séu framkvæmdar sem ein heild má skilgreina þær sem eina *hreyfingu*<sup>5</sup>. Líkja má SQL-hreyfingu við það þegar fjallgöngukappar binda sig saman í blindbyl; týnist þeir eru þeir þó í það minnsta saman. Hreyfingar eru mikilvægar til að tryggja það að gögn í gagnagrunnum skemmist ekki þegar villur koma upp við breytingar.

<sup>5</sup> e. *transaction*

## 7.6 Yfirlit

Í þessum kafla fórum við yfir nokkur atriði sem tengjast því að uppfæra gagnagrunna og hinar mismunandi skipanir sem því tengjast.

- SQL-skipanir sem við höfum farið yfir skiptast í tvo flokka, DDL og DML.
- DDL skipanir búa til, breyta og eyða gagnagrunnum og töflum.
  - CREATE DATABASE, CREATE TABLE, ALTER TABLE og DROP TABLE eru DDL-skipanir.
- DML-skipanir sýna, setja inn, breyta eða eyða *gögnum* í gagnagrunnum án þess að hafa áhrif á gagnagrunninn sjálfan.
  - SELECT, INSERT, UPDATE og DELETE eru DML-skipanir.
- ALTER TABLE skipunin bætir við og eyðir dálkum taflna.
- UPDATE skipunin breytir gögnum sem eru í töflum.
- DELETE skipunin eyðir gögnum sem eru í töflum.

## 8

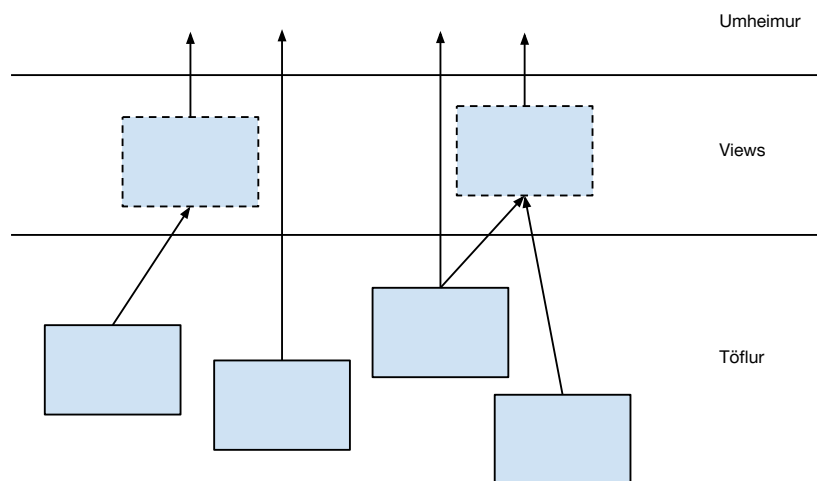
# Ítarefni

### 8.1 Views

Sú leið sem notuð er til að geyma gögn í gagnagrunnum er ekki alltaf sú leið sem hentar best til að vinna með gögnin.

Við getum til dæmis skoðað Tækniskólagagnagrunninn sem við bjuggum til í kafla 5 (sjá mynd 5.4). Sá gagnagrunnur inniheldur þó nokkuð margar töflur, oft þarf að nota JOIN (sjá undirkafla 6.2) á töflurnar sem í honum eru til að fá upplýsingarnar sem óskað er eftir. T.d. þarf að fara í gegnum þrjár töflur til að finna hvaða kennari kennir hvaða áfanga í Tækniskólagagnagrunninn (sjá sýnidæmi 6.4).

Það að þurfa sífellt að framkvæma svipaðar, en flóknar fyrirspurnir til að ná í gögn getur verið afskaplega þreytandi. Einfaldara væri ef við værum alltaf með akkúrat þær upplýsingar sem við þurfum, á því sniði sem við þurfum.



Mynd 8.1: Gagnagrunnur með “Views”

Til að auðvelda líf okkar í þessum aðstæðum getum við búið til svokölluð *View*. View er nokkurs konar “sýndartafla” sem skilgreint er af niðurstöðum fyrirspurnar.

Til að búa til View notum við DDL-skipunina<sup>1</sup> `CREATE VIEW`. Einföld `CREATE VIEW` skipun er byggð upp á eftirfarandi hátt: Fyrst kemur nafn skipunarinnar (`CREATE VIEW`), svo nafnið sem við ætlum að gefa view-inu, svo lykilorðið `AS` og að lokum fyrirspurnin (`SELECT`) sem view-ið á að byggjast á. Sjá sýnidæmi 8.1

<sup>1</sup> sjá DDL og DML í kafla 7

---

```
CREATE VIEW kennararAfanga AS
SELECT Afangar.audkenni AS afangi,
      Starfsmenn.nafn AS kennari
FROM Afangar
INNER JOIN Hopar
ON Afangar.numer = Hopar.afangaNumer
INNER JOIN Starfsmenn
ON Hopar.kennaraNumer = Starfsmenn.numer;
```

---

Sýnidæmi 8.1: `CREATE VIEW` skipun. Þessi skipun býr til sýndartöflu sem einfaldar aðgang að upplýsingum um hvaða kennari kennir hvaða áfanga.

Þegar view-ið hefur verið búið til má senda á það fyrirspurnir eins og hverja aðra töflu í gagnagrunninum. Sjá sýnidæmi 8.2.

---

```
SELECT kennari
FROM kennararAfanga
WHERE afangi = 'FOR1A3U';
```

---

Sýnidæmi 8.2: Kennarar sem kenna *FOR1A3U* fundnir með því að senda fyrirspurn á view-ið sem búið var til í sýnidæmi 8.1.

View hafa nokkra kosti aðra en að minnka flækjustig algengra aðgerða. T.d. er hægt að nota view til að koma (e.t.v. tímabundið) í stað töflu sem hefur verið breytt, svo niðurstöður fyrirspurna sem hafa verið skrifaðar breytist ekki þó að gagnagrunnurinn hafi verið uppfærður. Einnig er hægt að nota view til að stýra aðgangi að gagnagrunnum. T.d. er hægt að gefa notanda gagnagrunns aðgang að töflu sem inniheldur trúnaðarupplýsingar án þess að notandinn komist í þær upplýsingar með því að búa til view á töfluna sem inniheldur þær upplýsingar sem notandinn þarf, og gefa notandanum svo aðgang að view-inu en fela töfluna.



## 8.2 Yfirlit yfir helstu gagnagrunnskerfi

Gríðarmörg SQL-gagnagrunnskerfi eru til. Í þessum undirkafla verða þau gagnagrunnskerfi sem höfundur telur líklegast að nemendur Tækniskólans muni rekast á í náninni framtíð kynnt.

Grundvallaratriði SQL, sem farið er yfir í þessari bók, hafa miðast við notkun MySQL (8.2). Hugmyndirnar á bak við þessi grundvallaratriði er sú sama í öllum gagnagrunnskerfum sem byggja á SQL, en *útfærslan* er ekki endilega sú sama. Við skulum ekki búast við því að sýnidæmi bókarinnar keyri óbreytt í öllum gagnagrunnskerfum.

### MySQL

MySQL<sup>2</sup> er gagnagrunnskerfi í mikilli notkun, sérstaklega við vefsíðugerð.

Þökk sé útbreiðslunni er tiltölulega auðvelt að finna leiðbeiningar um notkun MySQL og uppsetningu MySQL-servera. Hægt er að vinna með MySQL í gegnum fjölmörg forritunarmál. MySQL er opið<sup>3</sup> og ókeypis.

MySQL hefur skilist að í nokkra hluta síðan það var fyrst gefið út. Upprunalegir höfundar kerfisins vinna nú við afbrigði sem heitir MariaDB<sup>4</sup>.

### PostgreSQL

PostgreSQL<sup>5</sup> er gagnagrunnskerfi sem leggur mikla áherslu á að fylgja stöðlum og bjóða upp á “réttu” gagnavinnslu.

PostgreSQL er vinsælt meðal gagnagrunnssérfræðinga, m.a. vegna þess hversu mikla stjórn gagnagrunnstjórinn hefur yfir virkni kerfisins. Kennt er á PostgreSQL gagnasafnsfræðiáföngum Tækniskólans.

Fyrir utan það að styðja “venjulegar” SQL skipanir, þá býður PostgreSQL upp á forritunarmál, kallað PL/pgSQL (Procedural Language/PostgreSQL), til að auðvelda ýmsar aðgerðir. PL/pgSQL býður meðal annars upp á lykkjur og önnur töl sem kunnugleg eru úr forritunarmálum á borð við C#.

Mynd 8.2: MySQL

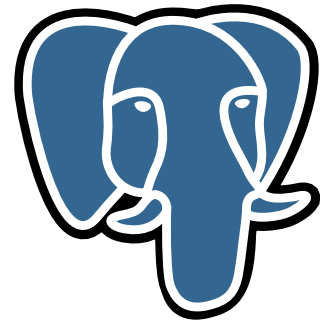


<sup>2</sup> <http://www.mysql.com/>

<sup>3</sup> e. *open source*

<sup>4</sup> <https://mariadb.org/>

Mynd 8.3: PostgreSQL



<sup>5</sup> <http://www.postgresql.org/>

### SQLite

SQLite<sup>6</sup> er ólíkt flestum gagnagrunnskerfum að því leyti að ekki þarf að setja upp eiginlegt kerfi á tölvunni sem á að hýsa gagnagrunninn, allt forritið er ein skrá.

Smæðarinnar vegna vantar SQLite ýmsa eiginleika sem stærri gagnagrunnskerfi bjóða upp á, en það hentar sérstaklega vel til að nota sem hluta af stærri kerfum. SQLite má finna “undir húddinu” á mörgum forritum sem þurfa að geyma gögn.

### Microsoft SQL Server

SQL Server<sup>7</sup> er gagnagrunnskerfi gefið út af Microsoft. Það er sniðið til að passa vel til keyrslu á Windows-vélum. SQL Server er helsti keppinautur Oracle gagnagrunnskerfisins þegar kemur að stórum gagnagrunnum.

Sú útgáfa af SQL sem notuð er til samskipta við SQL Server heitir Transact-SQL. T-SQL styður lykkjur og breytur.

### Oracle Database

Gagnagrunnskerfi Oracle<sup>8</sup> er mest notaða gagnagrunnskerfi í heimi í dag. Það hefur verið í þróun áratugum saman og knýr marga af heimsins stærstu gagnagrunnum.

Útvíkkun Oracle á SQL til að styðja lykkjur og önnur algeng forritunaratriði er kallað PL/SQL (Procedural Language/Structured Query Language).

## 8.3 Venslalíkanið

SQL byggir á föstum stærðfræðilegum grunni. Sá grunnur er kallaður *venslalíkanið*<sup>9</sup>. Því var fyrst lýst af tölvunarfræðingnum Edgar Codd um 1970<sup>10</sup>.

Venslalíkanið lýsir því hvernig líta má á gögn sem stök í mengjum og hvernig nota má þekktar<sup>11</sup> stærðfræðiaðgerðir til að nálgast þær.

SQL er *útfærsla* á venslalíkaninu. Uppbygging þess leiðir til þess að þegar við tölum um SQL notum við sjaldnast sama orðaforða og er notaður í stærðfræðinni sem það byggir á. Engu að síður er skilningur á venslalíkaninu og þeim orðaforða sem þar kemur við sögu mjög gagnlegur öllum sem nota SQL.

Mynd 8.4: SQLite



<sup>6</sup><http://sqlite.org/>

Mynd 8.5: SQL Server



<sup>7</sup><http://www.microsoft.com/sqlserver>

Mynd 8.6: Oracle Database



<sup>8</sup><http://www.oracle.com/us/products/database/overview/>

<sup>9</sup> e. *Relational model*

<sup>10</sup> Aðalgreinin sem lýsir því heitir “*A Relational Model of Data for Large Shared Data Banks*”. Áhugasamir geta fundið hana á netinu.

<sup>11</sup> Þær eru í það minnsta þekktar flestum stærðfræðingum.

Ítarleg umfjöllun um venslalíkanið er viðfangsefni fyrir háskólanám-skeið, ekki þessa bók. Engu að síður skulum við skoða mikilvægasta hugtak líkansins, *vensl*, og hvernig það passar við það sem við höfum séð af SQL í þessari bók.

## Mengi

Áður en lengra er haldið skulum við vera viss um að við höfum hugmynd um hvað *mengi*<sup>12</sup> er. Þetta eru sömu mengin og við höfum kynnst í grunnskólastærðfræði.

<sup>12</sup> e. *set*

Mengi er safn/hópur af fyrirbærum. “Fyrirbærin” í mengi geta verið nær hvað sem er, t.d. tölur, orð, eða önnur mengi. “Heiltölur”, “Íslendingar” og “Rauðir hlutir” eru allt mengi.

Tvö atriði einkenna mengi:

- Röð hluta í mengi skiptir ekki máli.
- Aldrei er um endurtekningar að ræða í mengi. Hlutur er annaðhvort í mengi eða hann er það ekki.

Þegar hlutur er í mengi er talað um að hluturinn sé *stak* í menginu.

Mengjum er oft gefið nafn sem er einn stór bókstafur. T.d. er algengt að kalla mengi heiltalna  $N$ .

Þetta gerir okkur mögulegt að kynnast fleiri hugtökum. Skoðum tvö mengi,  $A$  og  $B$ . Það skiptir okkur ekki máli hvað er í mengjunum.

- Séu öll stök í mengi  $A$  líka stök í mengi  $B$  er sagt að  $A$  sé *hlutmengi*  $B$ .
- Séu einhver stök í mengi  $A$  sem líka eru í mengi  $B$  er sagt að þau stök séu í *sniðmengi*  $A$  og  $B$ .
- Mengi sem inniheldur öll stök úr mengjum  $A$  og  $B$  er kallað *sammengi*  $A$  og  $B$ .

## Vensl

Við skulum sjá fyrir okkur mörg mengi af upplýsingum. Köllum þau öll  $S$  og gefum þeim númer, svo við getum kallað þau  $S_1, S_2$  og svo framvegis.

Skoðum nú annað mengi og köllum það  $R$ . Mengið  $R$  er *vensl*<sup>13</sup> á mengin  $S$  ef  $R$  er mengi af línum<sup>14</sup> þar sem að fyrsta gildið í línunni er úr  $S_1$ , annað stakið úr  $S_2$ , og svo framvegis.

<sup>13</sup> e. *relation*

<sup>14</sup> e. *tuples*

Til að gera venslin nothæf getum við ákveðið að hvert mengi  $S$  standi fyrir einhvern ákveðinn *eiginleika*<sup>15</sup> sem gögn geta haft. Þá stendur hver lína í venslunum fyrir einn “hlut” sem einkennist af þeim eiginleikum sem við skilgreindum.

<sup>15</sup> e. *attribute*

Hvernig tengist þetta þeim hugtökum sem við höfum lært úr SQL?

- Lína í SQL-töflu samsvarar (e. *tuple*) í venslum.
- Dálkur í SQL-töflu samsvarar eiginleika (e. *attribute*) í venslum.
- SQL-tafla samsvarar að mestu leyti venslum.<sup>16</sup>
- Niðurstöður fyrirspurna og view-a (sjá undirkafla 8.1) samsvara líka að mestu leiti venslum.

<sup>16</sup> Vensl og SQL-töflur greinast að á nokkrum atriðum. T.d. eru vensl alvöru mengi, sem ekki geta verið með endurtekningar. SQL-tafla sem ekki er með aðallykil getur innihaldið endurtekin gildi.

Það að skilja venslalíkanið gerir sum atriði sem viðkoma SQL skýrari, t.d. samband view-a og taflna. Einnig getum við nú séð að þegar rætt er um venslaðar töflur er átt við töflur sem hægt er að búa til vensl á.

## 8.4 Að tengjast gagnagrunni með PHP

Við höfum eytt miklum tíma í að skoða gagnagrunna sem sjálfstætt fyrirbrigði.

Í þessum undirkafla skoðum við loksins hvernig tengja má MySQL-gagnagrunn við PHP-forritskóða. Slíkar tengingar eru teknar fyrir vegna þess hve algengar<sup>17</sup> þær eru í vefforritun.

Útskýringarnar gera ráð fyrir skilningi á ýmsum atriðum:

- Hugtökum í vefsíðum - HTML, Javascript
- Keyrslu PHP-skripta
- Grundvallarmálfræði PHP
- IP-tölum
- Föllum, lykkjum og fylkjum<sup>18</sup>

<sup>17</sup> Linux, Apache, PHP og MySQL mynda saman “pakka” sem oft er notaður sem ein heild. Pakkinn er nefndur eftir skammstöfun sinni, LAMP. Hann er í gríðarmikilli notkun.

<sup>18</sup> e. *functions, loops og arrays*

### Hlutverk gagnagrunna í vefsíðum

Vefsíða sem byggð er upp á hefðbundinn hátt skiptist gróflega í tvo hluta - client og server.

Client-hlutinn er sá hluti sem keyrður er á tölvu notandans. Í client-hluta eru HTML-tög túlkuð og Javascript-kóði keyrður, m.a.. Venjulega fer þessi vinna fram í vafra<sup>19</sup> notandans.

<sup>19</sup> e. *browser*, t.d. Google Chrome, Firefox og Internet Explorer

Server-hlutinn er margskiptur. Viðfangsefni okkar, PHP og MySQL, tilheyra þessum hluta. Oft keyra PHP og MySQL á sömu tölvu, sem er þá einfaldlega nefnd "serverinn".

Hlutverk gagnagrunnsins í þessari uppbyggingu er, eðli hans samkvæmt, það að halda utan um upplýsingar. PHP-hluti serversins sér um að eiga samskipti við gagnagrunninn og miðla upplýsingunum áfram til clientsins. Notandinn og tölva hans eiga aldrei bein samskipti við MySQL-serverinn.

### Uppsetning tengingar með PDO

Gerum ráð fyrir að við séum með PHP-skriptu sem keyrir á server. Til að tengjast MySQL-gagnagrunni þarf hún eftirfarandi upplýsingar:

- Tengingarupplýsingar: Gagnagrunnsgerðina (hjá okkur alltaf MySQL), nafn gagnagrunnsins og staðsetningu hans
- Notandanafn MySQL-þjónsins
- Lykilorð notandans.

Klasann [PDO](#) má svo nota til þess að mynda tenginguna sjálfa.<sup>20</sup>

---

```
<?php
```

```
$source = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'notandanafn';
$password = 'lykilorð';

try {
    $dbh = new PDO($source, $user, $password);
} catch (PDOException $e) {
    echo 'Tenging mistókst: ' . $e->getMessage();
}
```

```
?>
```

---

<http://www.php.net/manual/en/class.pdo.php>

<sup>20</sup> Fleiri leiðir eru færar til að mynda tengingar af þessum toga, til dæmis *mysqli* viðbótin. Þá PHP-viðbót sem einfaldlega heitir *mysqli* skal þó alls ekki nota - hún er úreld.

Sýnidæmi 8.3: Tenging við gagnagrunn með PDO. Þessi kóði er settur í skrá sem heitir *dbcon.php*.

Dæmi um hvernig öll skriptan gæti litið út má sjá á sýnidæmi 8.3. Skoðum það sýnidæmi nánar.

Fyrsta breytan sem er skilgreind í skriptunni (*\$source*) inniheldur tengingarupplýsingarnar. Þar kemur fyrst fram orðið *mysql*, aðskilið frá gagnagrunnsnafninu og staðsetningu gagnagrunnsins með tvípunkti.

Sé ætlunin að skrifa skriptu af þessum toga til nota í eigin forriti þyrfti að skipta út testdb fyrir nafn gagnagrunnsins sem nota skal. Sé MySQL-serverinn ekki að keyra á sömu tölvu og PHP-skriptan þarf einnig að breyta IP-tölunni í IP-tölu tölvunnar sem MySQL-serverinn keyrir á.

Næstu tvær breytur innihalda notandanafn og lykilorð fyrir MySQL-serverinn. Þetta er sama notandanafn og lykilorð og notað var til að tengjast gagnagrunninum í kafla 2.

Pegar upplýsingarnar hafa verið geymdar er loks “reynt” að búa til tengingu með PDO. Takist það eru tengingarupplýsingarnar geymdar í “database handle” breytu, nefnd \$dbh í sýnidæminu. Þessa breytu má síðan nýta til að eiga samskipti við gagnagrunninn.

Takist ekki að koma tengingunni á prentar skriptan út strenginn “Tenging mistókst” ásamt villuskilaboðunum sem berast.

### *Að sækja gögn með PDO*

Pegar tengingin er komin upp er hægt að nota hana til þess að ná í gögn úr viðkomandi gagnagrunni.

Gefum okkur að highscore-taflan í sýnidæmi 3.6 sé til staðar í gagnagrunninum.<sup>21</sup> Rifjum upp að þetta er einföld tafla með tveimur dálkum: Annar dálkurinn heitir “player” og inniheldur þriggja stafa skammstöfun á nafni spilara, hinn dálkurinn heitir “score” og inniheldur stigafjölda. Athugum hvernig við getum sótt þessi gögn úr gagnagrunninum.

Byrjum á að búa til php-breytu sem inniheldur fyrirspurnina sem við viljum framkvæma. Það má gera á eftirfarandi hátt:

```
$sql = 'SELECT player, score FROM HighScores';
```

Breytan \$sql inniheldur þá streng sem skilgreinir fyrirspurnina.

Næst þarf að *keyra* fyrirspurnina. Það má gera með “query” fallinu sem skilgreint er af PDO. Notum það svona:

```
$result = $dbh->query($sql);
```

Breytan \$result inniheldur þá niðurstöðu fyrirspurnarinnar.

Niðurstöðurnar koma þó ekki á mjög vingjarnlegu sniði frá PDO-klasanum. Til að vinna með niðurstöðurnar er best að setja þær í fylki<sup>22</sup>. Til þess getum við notað fallið “fetch”.

<sup>21</sup> Ef hún er ekki til staðar má keyra sýnidæmið til að setja töfluna upp.

<sup>22</sup> e. *array*

Fetch “les” eina línu af niðurstöðunum í `$result` í hvert skipti sem það er gert. Við getum því notað lykkju til að ná í allar niðurstöðurnar:

```
while ($row = $result->fetch()) {
    $highscores[] = array($row['player'], $row['score']);
}
```

Fylkið `$highscores` mun innihalda allar upplýsingarnar úr dálkunum “player” og “score” þegar keyrslu lykkjunnar lýkur.

Að lokum mun php-skriptan okkar líta út eitthvað á borð við það sem sjá má í sýnidæmi 8.4.

---

```
<?php

try {
    $sql = 'SELECT player, score FROM HighScores';
    $result = $dbh->query($sql);
} catch (PDOException $ex) {
    echo 'Error fetching scores: ' . $ex->getMessage();
}

while ($row = $result->fetch()) {
    $highscores[] = array($row['player'], $row['score']);
}

?>
```

---

Sýnidæmi 8.4: PDO notað til að sækja upplýsingar úr gagnagrunni og geyma þær í fylkinu `$highscore`. Athugum að breytan `$dbh` þarf að vera skilgreind þegar þessi skripta er keyrð, sjá undirkafla 8.4. Vistum skriptuna í skrá sem við köllum *query.php*.

### Að setja saman HTML og PHP

Nú höfum við séð hvernig tengjast má gagnagrunni með PHP (sýnidæmi 8.3) og hvernig sækja má upplýsingar í gegnum tenginguna (sýnidæmi 8.4).

Búum nú til vefsíðu sem notar þessar skriptur, `dbcon.php` og `query.php`. Gerum það með einni skrá í viðbót - köllum hana `index.html.php`. Við veljum nafnið *index* af því það er það nafn sem flestir vefþjórnar leita að sem upphafssíðu. Við gefum því viðaukann `.html` af því að þessi skrá mun innihalda HTML-tög að miklum meirihluta. Við endum nafnið á `.php` af því að þetta verður eftir allt saman PHP-skripta sem við þurfum að keyra.

Byrjum á að búa til “tóma” HTML-skrá sem vafri getur opnað. Hún getur litið út eins og sýnidæmi 8.5.

---

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Titill vefsíðu</title>
  </head>
  <body>
  </body>
</html>

```

---

Sýnidæmi 8.5: HTML beina-  
grind. Þetta er tóm síða.

Bætum nú PHP-kóða við tómu skrána svo úr verði sýnidæmi 8.6.

Byrjum á að láta `index.html.php` innihalda skripturnar sem við bjuggum til með *include* skipunum.

Þegar þessum skriptum hefur verið bætt við veit `index.html.php` af öllum breytum sem skilgreindar voru í skriptunum, þar á meðal fylkinu sem við geymdum niðurstöður fyrirspurnarinnar í, `$highscores`.

Nú getum við búið til HTML-töflu. Fyrst skrifum við þá hluta sem ekki eru háðir gögnunum í gagnagrunninum beint, síðan getum við ítrað<sup>23</sup> í gegnum `$highscores` fylkið til að búa til raðir HTML-töflunnar. Niðurstaðan er sýnidæmi 8.6.

<sup>23</sup> e. *iterated*

Að lokum fáum við “heimasíðu” sem sýnir upplýsingarnar. Sú heimasíða gæti e.t.v. litið betur út, en hún gerir það sem hún á að gera - hún er beintengd gagnagrunninum og uppfærast þegar honum er breytt. Svona aðferðir liggja á bakvið mjög margar heimasíður.



```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Titill vefsíðu</title>
  </head>
  <body>
    <?php
    include "dbcon.php";
    include "query.php";
    ?>
    <table border="1">

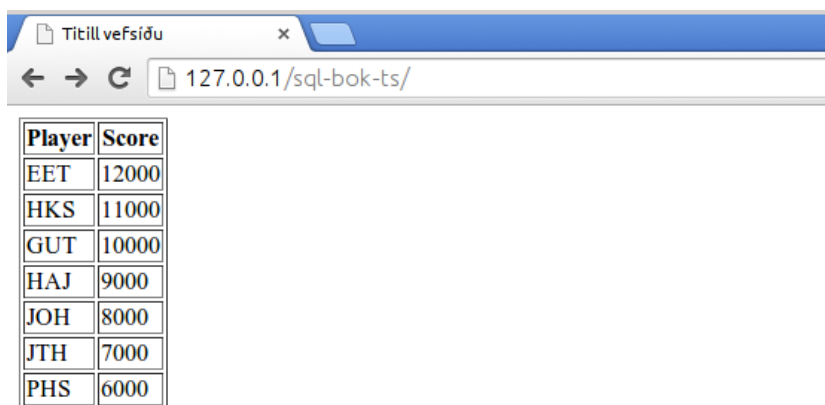
      <tr>
        <th>Player</th>
        <th>Score</th>
      </tr>

      <?php
      foreach($highscores as $entry){
        echo '<tr><td>'.$entry[0].</td><td>'.$entry[1].</td></tr>';
      }
      ?>

    </table>
  </body>
</html>

```

Sýnidæmi 8.6: PHP og HTML notað saman til að mynda HTML-töflu sem inniheldur upplýsingarnar úr “HighScores” SQL-töflunni.



Player	Score
EET	12000
HKS	11000
GUT	10000
HAJ	9000
JOH	8000
JTH	7000
PHS	6000

Mynd 8.7: Highscore “heimasíðan”, sem fær gögn sín úr gagnagrunni.

## 9

# Um þessa bók

### 9.1 Um útgáfu

Útgáfa bókarinnar er styrkt af Rannís, Próunarsjóði námsgagna. Út-lutunarárið er 2014, heiti verkefnisins er “Notkun gagnasafna fyrir framhaldsskólanema”.

### 9.2 Tæknileg atriði

Bókin er skrifuð í  $\text{\LaTeX}$ . Kóðinn er öllum aðgengilegur á [Github síðu höfundar](https://github.com/Ernir/sql-bok-ts)<sup>1</sup>.

<sup>1</sup><https://github.com/Ernir/sql-bok-ts>

### 9.3 Leyfi

Þessi bók er gefin út undir [Creative Commons BY-NC-SA 4.0](http://creativecommons.org/licenses/by-nc-sa/4.0/)<sup>2</sup> leyfi.

<sup>2</sup><http://creativecommons.org/licenses/by-nc-sa/4.0/>

Í stuttu máli veitir leyfi þetta rétt til að:

- Deila bókinni að vild
- Aðlaga bókina, breyta henni eða bæta

Svo fremi sem

- Uppruni afleiddu bókarinnar kemur skýrt fram
- Bókin og/eða afleidda bókin eru ekki nýtt í gróðaskyni
- Þetta leyfi fylgir afleiddu bókinni óbreytt

Ekki má bæta við öðrum hamlandi leyfum eða tæknilegum viðbótum til að hindra notkun eða dreifingu afleiddrar bókar.

#### 9.4 Þessi útgáfa

Þetta er útgáfa 1.0.0 af bókinni, gefin út 20. ágúst 2014.

#### 9.5 Þakkir

Þessi bók væri ekki til án Hrefnu Karítasar Sigurjónsdóttur og Sigurðs R. Ragnarssonar. Þakka ykkur kærlega.