

Tartalom

Bevezetés.....	1
Beállítások (Workbench WB).....	2
Adatbázis készítés.....	4
Adatbázis szerkezete, kapcsolatok.....	6
Feladatok.....	8
Önálló, megoldás nélküli feladatok.....	25
Megoldások.....	25

Bevezetés

Jelen jegyzettel az adatbázisok világához szeretnék egy kis útmutatót adni. Nem célom hosszú szövegek írása inkább könnyebb-nehezebb fejtörőkön keresztül szeretném megmutatni az SQL nyelvet. Ennek eszköze az ingyenesen (jogi szöveg elolvasandó) használható méltán híres Mysql adatbázis motor lesz. A kód írására a Mysq Workbench nevű programot fogom használni.

Persze nem elég egy adatbázis motor kell még hozzá adat is. Ehhez az internetről letölthető Employees adatbázist fogom használni. A jegyzethez mellékeltem az eredeti fájlokat (itt is van jogi bla-bla). Mivel az adatok (dátumok) elég őskoriak a jegyzet írásakor (az Úr 2020. éve) ezért azokat megváltoztattam. Az adatbázis Sql scriptekből készíthető el. A táblák és a bennük lévő adatok az employees_full.sql futtatásával készíthetők el. Minden egyes tábláról készült egy script mely a tábla és az adatok visszaállítását szolgálja. Erre azért van szükség, hogy bátran meg tud írni a feladatok megoldást. Abban az esetben, ha valami nem sikerül könnyű lesz az eredeti adatokat visszanyerni és lehet újra próbálkozni. Igen a feladatoka neked kell megoldanod, neked kell megtalálni a választ. Ugyanúgy ahogy azt majd az ÉLETBEN is. Sok feladatnak megtalálod a megoldását a jegyzet végén. Nem javaslom ennek a szekciónak a felkeresését legalábbis azelőtt nem, hogy lenne egy működő megoldásod mely saját kútfőből érkezett. Természetesen a kedvenc kereső motorod még minid a barátod. Egyes feladatoknál vannak tippek is, hogy merre érdemes elindulni. A megoldások esetében nem célom a leghatékonyabb kódot megírni inkább az adatbázis motorban lévő lehetőségeket

szeretném megmutatni. Sok-sok beépített funkciót is fogunk használni. Benézzünk a tárolt eljárások, illetve a tranzakciók világába is a lekérdezések és az adatmanipuláló műveletek mellett. A feladatok nem nehézségi sorrendben vannak nyugodtan át lehet ugrani egyet-egyet későbbi megoldás céljából.

Az adatbázisban igen nagy mennyiségű adat található (több mint 4 millió rekord). Köszönet érte a készítőknak. Ez azt is jelenti, hogy egy-egy művelet hosszabb ideig is tarthat főleg azok melyekben a fizetés tábla is szerepel. Ennek fényében be kell majd állítani a Workbench egyes tulajdonságait. Ezzel fogunk kezdeni, ezek után elkészítjük az adatbázist, megismerkedünk annak szerkezetével a kapcsolatokkal, majd jöhet a legizgalmasabb rész a feladatok megoldása. A megoldás részt tényleg csak akkor nézd meg ha már két napja gondolkozol a helyes kódon és nem jutsz eredményre. Abból nem tanulsz, ha megnézed a kódot és lemásolód azt. Sokkal több tudásra tudsz szert tenni a saját ötleteidből megvalósított kóddal. Próbálkozni lehet bátran hiszen az adatokat bármikor vissza lehet állítani az eredeti állapotukba. Sok beszédnek sok az alja. Vágjunk is bele.

Jó JOINT-ozást a Szerző.

Beállítások ([Workbench WB](#))

Feltételezem már telepítve van az adatbázis motor és a Workbench (WB) programban van egy kapcsolat. Ezekkel a jegyzet nem foglalkozik. Kezdjük a beállításokkal. Nyissuk meg a WB programot és csatlakozunk a futó Mysql szervezhez. Ezek után a Edit → Preferences → SQL Editor menüben állítsuk át a következőket:

MySQL Session

DBMS connection keep-alive interval (in seconds): Time interval between sending keep-alive messages to DBMS. Set to 0 to not send keep-alive messages.

DBMS connection read timeout interval (in seconds): The maximum amount of time the query can take to return data from the DBMS. Set 0 to skip the read timeout.

DBMS connection timeout interval (in seconds): Maximum time to wait before a connection attempt is aborted.

Other

Internal Workbench Schema: This schema will be used by MySQL Workbench to store information required for certain operations.

☐ Safe Updates (rejects UPDATES and DELETES with no restrictions)

OK Cancel

Ezzel beállítottuk, hogy egy-egy hosszabb művelt esetén ne szakadjon meg a kapcsolat. A „Safe Updates” kikapcsolásával tudunk olyan update-delete műveletet írni mely nem hivatkozik elsődleges kulcsra. Mivel nagy mennyiségű adattal dolgozunk és ezeket szeretnénk látni is állítsuk még át az SQLExecution alatt található limit pipát üresre:

Object Editors

SQL Execution

Administration

Modeling

Defaults

MySQL

Diagram

Appearance

Fonts & Colors

SSH

Others

☐ Continue SQL script execution on errors (by default)

☒ New connections use auto commit mode

Progress status update interval (in milliseconds):

SELECT Query Results

☐ Limit Rows

Limit Rows Count:

Max. Field Value Length to Display (in bytes):

☐ Treat BINARY/VARBINARY as nonbinary character string


☒ Confirm Data Changes

☒ Preserve Row Filter

Persze érdemes körülnézni a többi beállítás között is és a saját képünkre szabni a programot. Ezt már rád bízom kedves olvasó. Meg is vagyunk jöhetnek az adatok.

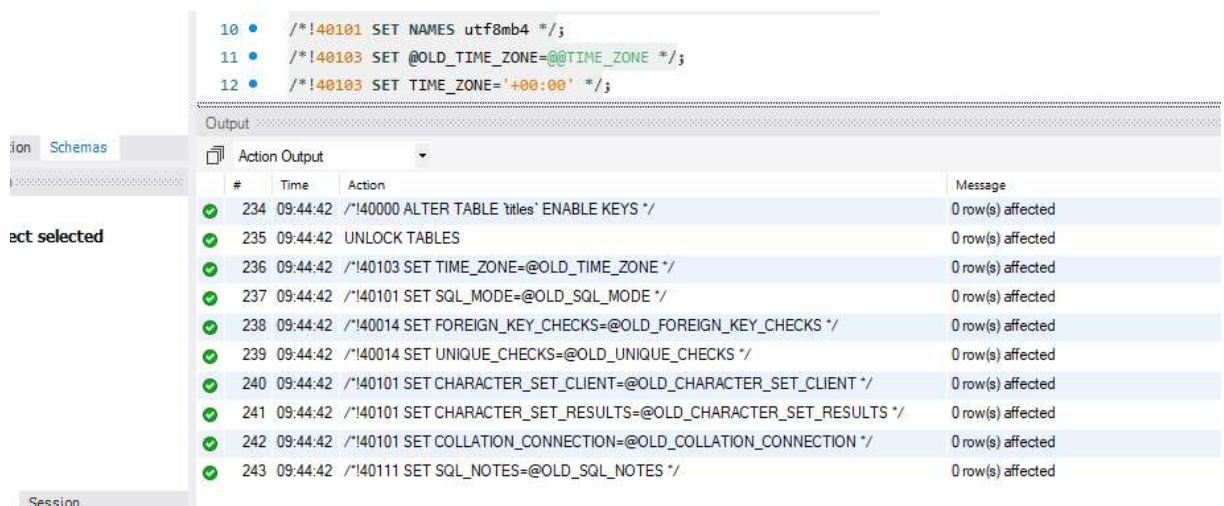
Adatbázis készítés

A mellékelt könyvtár szerkezetet gondolom már kicsomagoltad, ha mégsem akkor ezt érdemes megtenni, hogy könnyebb legyen az élet. Itt a gyökér könyvtárban találsz a „employees_full.sql” fájlt. Ezt kell megnyitni a WB programban. Két lehetőség is van:

- WB bal felső sarok Open Sql script file ikon 
- Megnyitod a fájlt egy notepad alkalmazásban majd bemásolod a tartalmát a WB egy új Sql script felületére.

Az első módszer választása esetén a WB fog egy kicsit nyervogni, hogy a fájl nagyon nagy. Nem kell kétségbe esni válasszuk a Open lehetőséget. Picit kell is várakozni míg a fájl tartalma betöltődik.

Ezek után válasszuk a lenti képen piros nyíllal jelzett ikont. Ez az egész script-t végre fogja hajtani. A zöld nyíl villám csak egy sort hajt végre a scriptünkből. Ez a sor az lesz, ahol a kurzor villog. Itt is van lehetőség több sor végrehajtására abban az esetben, ha kijelöljük a végrehajtandó sorokat. A gomb megnyomása után már kezdődhet is a móka-kacagás, illetve a várakozás, hogy az adatbázis elkészüljön. A folyamatot végig tudjuk követni az output ablakon melyet alul találunk az esetleges hibák is itt fognak megjelenni.

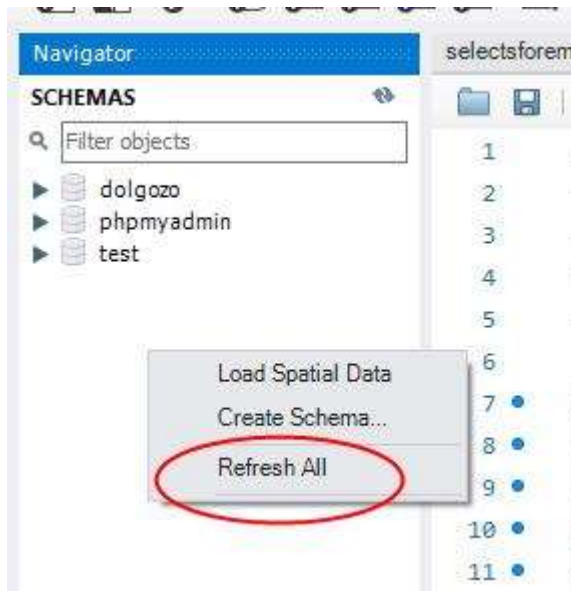


The screenshot shows the WinBox (WB) interface. The top panel displays the SQL script content with line numbers 10 to 12. The bottom panel shows the 'Output' window with a table of execution results.

#	Time	Action	Message
234	09:44:42	/*!40000 ALTER TABLE 'titles' ENABLE KEYS */	0 row(s) affected
235	09:44:42	UNLOCK TABLES	0 row(s) affected
236	09:44:42	/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */	0 row(s) affected
237	09:44:42	/*!40101 SET SQL_MODE=@OLD_SQL_MODE */	0 row(s) affected
238	09:44:42	/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */	0 row(s) affected
239	09:44:42	/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */	0 row(s) affected
240	09:44:42	/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */	0 row(s) affected
241	09:44:42	/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */	0 row(s) affected
242	09:44:42	/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */	0 row(s) affected
243	09:44:42	/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */	0 row(s) affected

Remélhetőleg valami hasonló képpel találkozol. Hiba esetén a Google a barátod 😊

A program bal oldalán találod a Navigator ablakot. Jobb klikk itt és válaszd a Refresh All menüt. A *Navigator* ablak alján a *schemas* tab-t válaszd.



Ezek után meg kell, hogy jelenjen az employees adatbázis a listában.

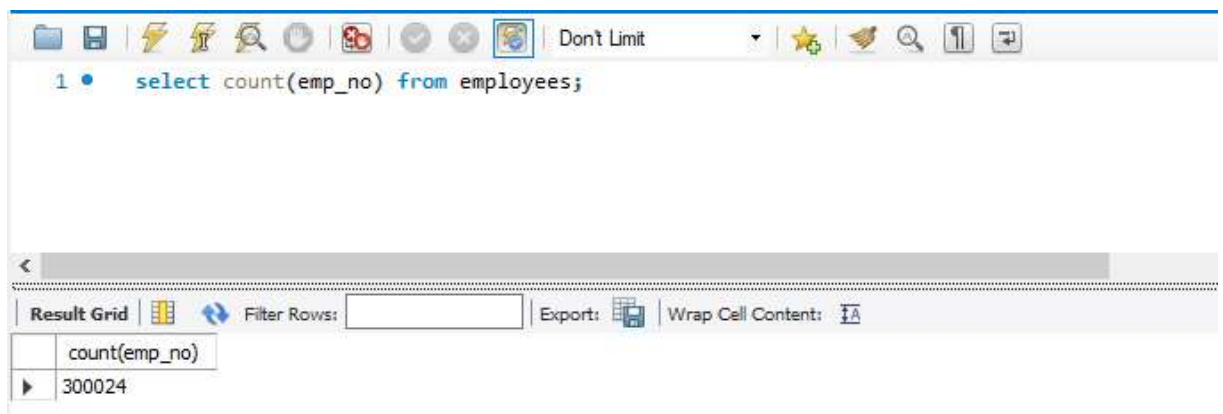


Következő lépésben jobb klikk az adatbázis nevéen és a set as default schema menüpont választása. Ekkor lesz ilyen szép vastag betűs az adatbázis neve, mint a képen. Erre azért volt szükség, hogy a scriptjeinkben ne kelljen a use database utasítást használni. Egy gyors

ellenőrzés az adatokra. Nyiss egy új scriptet. Ezt a bal felső sarokban tudod megtenni a New Sql ikon segítségével.



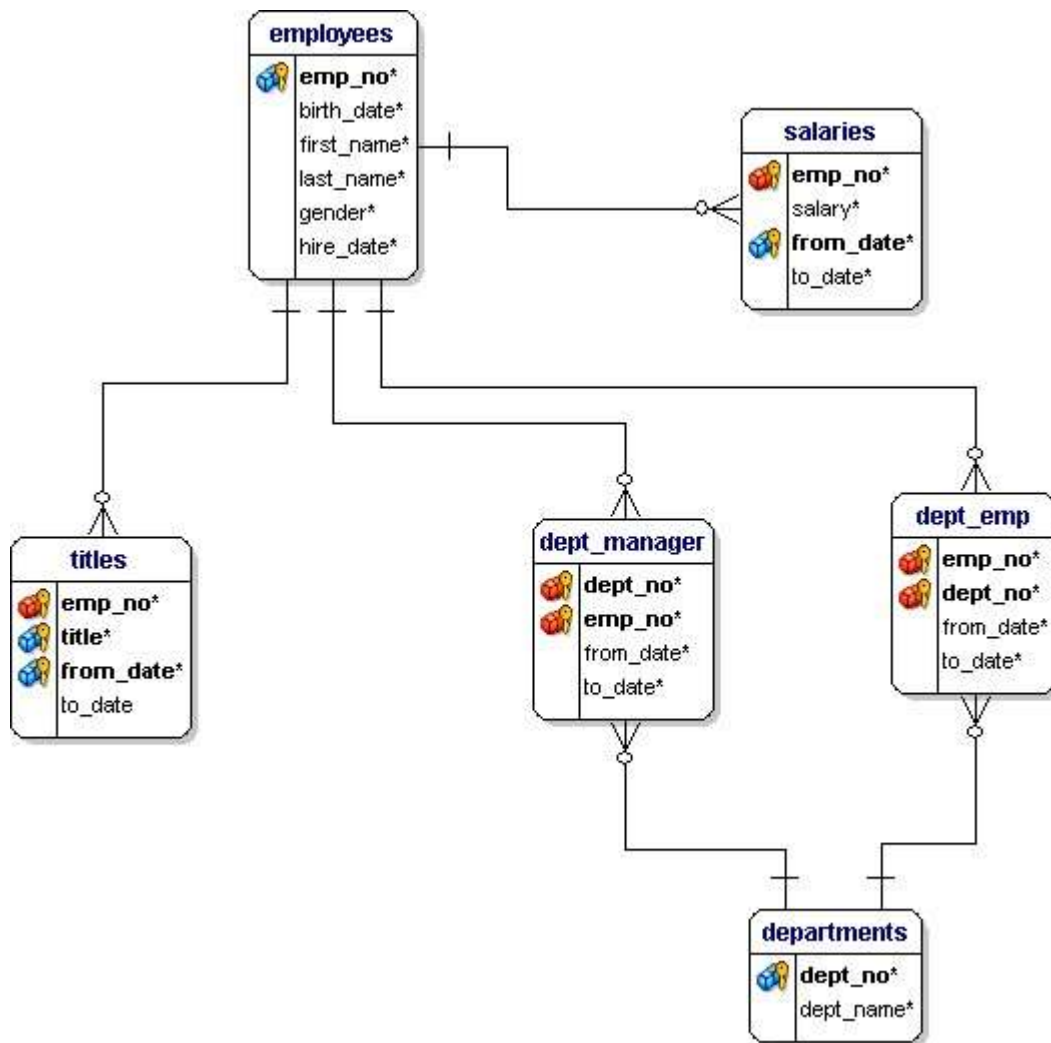
Az új felületre gépeld be a következő sort:



Majd valamelyik villám jellel futtasd a scriptet. A result grid-ben pont a 300 024 számot kell látnod, mint a képen. Nézzük milyen adatokkal szemben kell felvennünk a harcot.

[Adatbázis szerkezete, kapcsolatok](#)

Az adatbázisban összesen hat darab táblát találunk ezek közül egy az kapcsoló tábla (dept_emp) sok-sok kapcsolatot ír le az employess és a departments táblák között. A többi kapcsolat egy-sok vagy egy-egy típusú. Nézzünk egy ábrát az adatbázis tábláiról és a kapcsolatokról:



Az ábrán nagyon jól látszik, hogy a dolgozó tábla elsődleges kulcsa (emp_no) egy tábla kivételével mindenhol megjelenik idegen kulcsként. A kivétel a departments tábla mely a normalizálás miatt jött létre. Az adatbázis tervezője úgy döntött, hogy rekordot nem lehet törölni bár a szigorítások nincsenek beállítva. Ezért jelzi sok helyen a kezdeti és vég dátumát a rekord érvényességének. Ebből következően egy dolgozó akkor aktív vagy van állományban, ha a hozzá tartozó fizetések közül az egyiknek és csak az egyiknek a vég dátuma (to_date) nagyobb, mint a mindenkori pillanatnyi dátum. Ezt az adatok között a maximális dátum (9999-01-01) beállításával jelzik. A feladatoknál külön fogom jelezni mikor csak az állományban levő dolgozókra kell megírni a megoldást, egyébként az összes adatra kíváncsiak leszünk. Egy-egy kapcsolatként tekinthetünk a dolgozó -> titulus, dolgozó -> fizetés, dolgozó -> részleg, dolgozó -> részleg manager kapcsolatokat. Itt kell megemlíteni, hogy az adatbázis szintjén ezek is egy több kapcsolatként vannak megvalósítva a változások miatt. Ezért nem szabad elfelejteni a kezdő és vég dátumok karbantartását egy-egy változás esetén. Egy-egy kapcsolat még a részleg

manager -> részleg kapcsolat mely így is van fizikailag megvalósítva.

Tanulmányozd az ábrát nagyon figyelmesen, hogy megértsd a kapcsolatokat a rekordok között mert ezeket a lekérdezések update-k során sűrűn használni kell majd. Egyik táblában sem lehet null értékű mező. Az elsődleges kulcsokat mindig kézzel kell készíteni egyik sem generált (auto increment) érték. Ez az adatmanipuláló műveletek esetén fontos információ. Az adatbázisból nem törölünk rekordot, fizetésemelés esetén az éppen aktív fizetést deaktiváljuk majd egy új fizetést készítünk. Természetesen ezt tranzakcióban fogjuk tenni. Lesz olyan feladat, ahol az adatbázis filozófiájával szemben módosítani fogunk a meglévő adatokon de ez csak a példa kedvéért. Ezt a feladat szövegében fogom jelezni. Most már kezdhetjük a tényleges munkát.

Feladatok

1. Kérjük le külön lekérdezésben az egyes táblákban levő rekordok számát. Az oszlop neveket próbáljuk úgy kiírni ahogy az a képeken látszik.

The image shows six separate screenshots of a SQL query result grid, each displaying a single row with a count:

Query	Result
employee count	300024
department count	9
department employees count	331603
salary count	2844047
title count	443308
department manager count	24

2. Írasd ki az összes dolgozó vezeték, keresztnévét és a részleg nevét, ahol dolgozik, dolgozott. Az adatokat a részleg neve alapján rakd sorba.

The image shows a screenshot of a SQL query result grid with the following data:

	last_name	first_name	dept_name
▶	Sluis	Mary	Customer Service
	Lortz	Huan	Customer Service
	Tramer	Basil	Customer Service
	Billingsley	Breannnda	Customer Service
	Syrzycki	Jungsoon	Customer Service

Összesen 331603 sort kell kapj eredményként.

3. Jöhet egy kis statisztika? Persze, hogy jöhet szinte látom a kihívás utáni vágyat a szemedben. A részlegek szerinti dolgozók számára, az átlag fizetésre és az éves összes bér költségre vagyok kíváncsi. A fizetés táblában az éves bér van eltárolva. Az adatokat a részleg neve alapján rendezd és használd ember számára olvasható oszlop neveket, mint a képen.

Tipp: Group by. A lekérdezés hosszsan futhat.



The screenshot shows a 'Result Grid' interface with a table containing department statistics. The table has four columns: 'department name', 'employee count', 'average salary', and 'salary sum'. The data is sorted by department name. The interface includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' toggle.

department name	employee count	average salary	salary sum
Customer Service	223644	58770.3665	13143639841
Development	810026	59478.9012	48179456393
Finance	165285	70489.3649	11650834677
Human Resources	168490	55574.8794	9363811425
Marketing	100001	71012.0000	7101200000

Összesen 9 sor az eredmény.

4. Szeretnék minden adatot megkapni a részleg nevű táblából név szerint sorba rendezve.



The screenshot shows a 'Result Grid' interface with a table containing a list of departments. The table has two columns: 'dept_no' and 'dept_name'. The data is sorted by department name. The interface includes a 'Filter Rows' input field and an 'Export' button.

dept_no	dept_name
d009	Customer Service
d005	Development
d002	Finance
d003	Human Resources
d001	Marketing

Összesen 9 sor az eredmény.

5. Azokat a dolgozókat keresem, akik páratlan napon születtek. Nevük egy oszlopban ahogy a képen és a születési dátum, évek száma, illetve a napnak a neve, amelyen született. A rendezés a szül dátum alapján történik a legfiatalabb van elől dátum egyezés esetén a vezetéknev dönt. Tipp: A mysql dátum kezelő függvényei sokat tudnak segíteni a megoldásban.

Egy kis segítség: https://www.w3schools.com/sql/sql_ref_mysql.asp

Result Grid		Filter Rows:		Export:	Wra
	name	birth date	age	day of week	
▶	Angel Gopalakrishnan	1980-02-01	40	Friday	
	Anger Henk	1980-02-01	40	Friday	
	Anily Zsolt	1980-02-01	40	Friday	
	Auyong Stamatina	1980-02-01	40	Friday	
	Ada S...	1980-02-01	40	Friday	

Result 22 x

Összesen 152 752 sor az eredmény.

6. A törzsgárda tagok, azaz a legkorábban felvett (hire_date) dolgozók minden a dolgozó táblában lévő adata vezetéknév szerint sorba rendezve. A már nem aktív dolgozók is.

Tipp: Talán valamelyik aggregáló függvény segíthet.

Result Grid		Filter Rows:		Edit:		Export/Import:	
	emp_no	birth_date	first_name	last_name	gender	hire_date	
▶	110085	1974-10-28	Ebru	Alpin	M	2000-01-01	
	111692	1969-10-05	Tonny	Butterworth	F	M 00-01-01	
	110511	1972-07-08	DeForest	Hagimont	M	2000-01-01	
	111035	1977-02-24	Przemyslaw	Kaelbling	M	2000-01-01	
	110022	1971-09-12	Margareta	Markovitch	M	2000-01-01	

employees 25 x

Összesen 9 sor

7. Csoportosítsuk dolgozóinkat a születési évük alapján. Arra vagyunk kíváncsiak, hogy adott évben hány darab dolgozó született. A lekérdezés eredményében az összes születési évnek szerepelnie kell.

Result Grid			Filter Rows:	Ex
	year of birth	employee count		
▶	1967	21209		
	1968	22857		
	1969	23228		
	1970	23104		
	1971	23051		
	1972	22850		
	1973	23276		
	1974	23311		
	1975	23126		
	-----	-----		

Result 28 x

Összesen 14 sor.

8. Előző feladatot oldjuk meg úgy, hogy csak a páros évek adatai jelennek meg.

Result Grid			Filter Rows:	Expo
	year of birth	employee count		
▶	1968	22857		
	1970	23104		
	1972	22850		
	1974	23311		
	1976	23065		
	1978	23080		
	1980	1940		

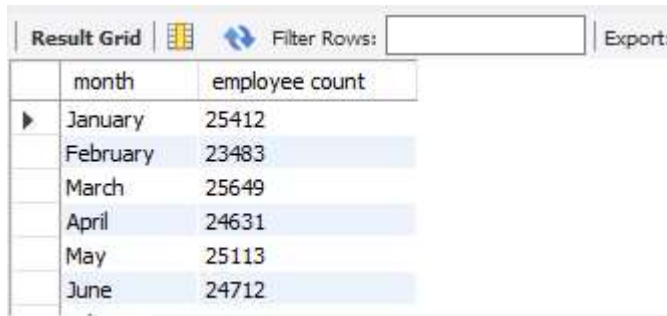
Összesen 7 sor.

9. Nézzük meg mennyire öreg vagy fiatal az állomány. A valaha volt és mostani dolgozók átlag életkorát keressük.

Result Grid		Filter Rows:
	average age	
▶	46.9181	

Összesen 1 sor.

10. Nézzük meg, hogy az egyes hónapokban 1-12 hány darab dolgozó született. Próbál meg a hónap nevét kiírtani és nem a számát.
Tipp: Megint a dátum kezelő metódusok között kell körülnézni.

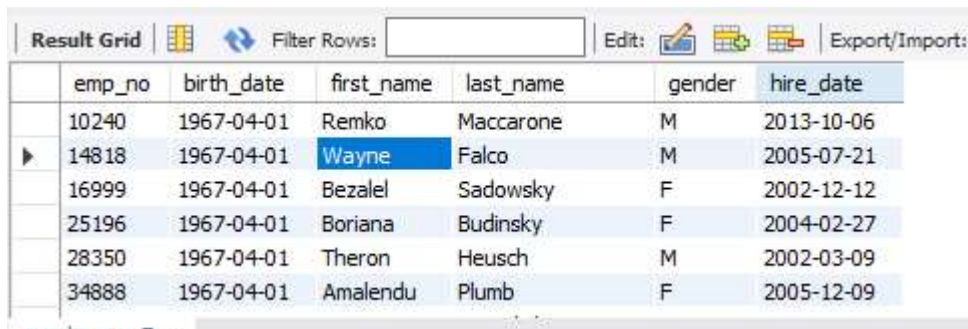


The screenshot shows a 'Result Grid' interface with a 'Filter Rows' input field and an 'Export' button. The table has two columns: 'month' and 'employee count'. The data is as follows:

month	employee count
January	25412
February	23483
March	25649
April	24631
May	25113
June	24712

Összesen 12 sor mivel minden hónapban született dolgozó

11. Azokat a dolgozókat szeretném látni, akiknek a múlt a mostani vagy jövő hónapban ünnepeljük a születés napját. Persze a születési dátum szerint rendezünk, idősebbeké az elsőbbség. Feltételezheted, hogy januárban és decemberben nem indítjuk el a lekérdezést.
Tipp: Dátum kezelés és between.



The screenshot shows a 'Result Grid' interface with a 'Filter Rows' input field, an 'Edit' button, and an 'Export/Import' button. The table has seven columns: 'emp_no', 'birth_date', 'first_name', 'last_name', 'gender', and 'hire_date'. The data is as follows:

emp_no	birth_date	first_name	last_name	gender	hire_date
10240	1967-04-01	Remko	Maccarone	M	2013-10-06
14818	1967-04-01	Wayne	Falco	M	2005-07-21
16999	1967-04-01	Bezalel	Sadowsky	F	2002-12-12
25196	1967-04-01	Boriana	Budinsky	F	2004-02-27
28350	1967-04-01	Theron	Heusch	M	2002-03-09
34888	1967-04-01	Amalendu	Plumb	F	2005-12-09

Összesen 74 456 sor.

12. Most írjuk meg a lekérdezésünket n napra. Azaz, ha az $n=8$ akkor azok a dolgozók, akiknek 8 nappal ezelőtt vagy a következő nyolc napban lesz a születésnapja. Itt sem kell attól tartani, hogy „átlógsz” egy másik évbe.

Result Grid						
	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	13305	1967-04-25	Bikash	Herath	M	2000-07-17
	13870	1967-04-25	Theirry	Kirkerud	F	2004-01-09
	13916	1967-04-25	Vidya	Wynblatt	M	2001-06-06
	17329	1967-04-25	Shaunak	Tyugu	M	2001-06-23
	22583	1967-04-25	Howell	Thorelli	M	2007-07-08
	24330	1967-04-25	Ebru	Piveteau	F	2004-06-21

Összesen 13 750 sor.

13. Most próbáljuk meg megvalósítani úgy, hogy akkor is jó eredményt adjon vissza, ha alul vagy felül „átlóg” egy másik évbe a napok száma.
Tipp: A tárolt eljárásokban lehet elágazást írni, illetve ezeknek lehet „inout” típusú paramétert adni.

Lenti lekérdezésben a kezdő hónap és nap '01-01' volt míg az n értéke 8:

Result Grid						
	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	12885	1967-12-25	Anwar	Collavizza	M	2005-09-11
	18503	1967-12-25	Mechthild	Marletta	M	2001-10-29
	23198	1967-12-25	Jayson	Pramanik	M	2009-01-23
	25207	1967-12-25	Basil	Broder	M	2009-04-23
	33936	1967-12-25	Khun	Bernini	M	2014-08-31
	37647	1967-12-25	Rosalyn	Baalen	F	2003-10-05

Result 26 employees 27 x

Összesen 13 919 sor.

14. Keressük meg a dolgozó táblában a legkisebb és a legnagyobb azonosítót (emp_no).

Result Grid		
	min(emp_no)	max(emp_no)
▶	10001	499999

Összesen 1 sor.

15. Az összes olyan dolgozót, akinek a fizetése nagyobb mint 30 000. Azokat is, akik nem aktívak.

Tipp: Sokáig fut, JOIN

emp_no	birth_date	first_name	last_name	gender	hire_date	salary
10001	1968-09-02	Georgi	Facello	M	2001-06-26	60117
10001	1968-09-02	Georgi	Facello	M	2001-06-26	62102
10001	1968-09-02	Georgi	Facello	M	2001-06-26	66074
10001	1968-09-02	Georgi	Facello	M	2001-06-26	66596
10001	1968-09-02	Georgi	Facello	M	2001-06-26	66961

Összesen 2844047 sor

16. Most csak az aktív állományt és a fizetés legyen nagyobb 80 000-nél.

emp_no	birth_date	first_name	last_name	gender	hire_date	salary
10001	1968-09-02	Georgi	Facello	M	2001-06-26	88958
10005	1970-01-21	Kyoichi	Maliniak	M	2004-09-12	94692
10007	1972-05-23	Tzvetan	Zielinski	F	2004-02-10	88070
10009	1967-04-19	Sumant	Peac	F	2000-02-18	94409
10010	1978-06-01	Duangkaew	Piveteau	F	2004-08-24	80324

Összesen 69786 sor.

17. Az aktív dolgozók átlag fizetése 2 tizedes jeggyel kiírva.

average salary
72.012,24

Összesen 1 sor.

18. Az átlagnál jobban kereső dolgozókat keressük. Az aktív állomány érdekel csak. A dolgozó táblából minden adat míg a fizetés táblából a havi fizetés és hogy mióta ennyi.

Tipp: A fizetés táblában az éves fizetés van eltárolva.

Result Grid								
Filter Rows: <input type="text"/>								
Export: <input type="button" value="Export"/> Wrap Cell Content: <input type="button" value="Wrap"/> Fetch rows: <input type="button" value="Fetch"/>								
	emp_no	birth_date	first_name	last_name	gender	hire_date	monthly salary	from_date
▶	13453	1970-11-30	Sugwoo	Lalonde	M	2010-01-09	5.318	2017-01-07
	61067	1972-04-13	Alois	Zultner	F	2007-08-18	5.318	2016-08-16
	210060	1973-12-12	Vitaly	Argence	M	2002-11-05	5.318	2016-11-01
	461198	1978-09-25	Debatosh	Akaboshi	M	2000-11-13	5.318	2016-11-09
	472464	1967-03-28	Nitsan	Zaiane	M	2013-11-12	5.318	2017-01-19
	86462	1973-05-04	Mantis	Champarnaud	F	2002-09-29	5.318	2017-01-25
	88500	1977-03-04	Rimli	Penttonen	F	2009-07-05	5.318	2017-03-02
	98936	1973-08-06	Tesuro	Merro	F	2004-02-08	5.318	2017-07-03
	21231	1975-10-01	Lubomir	Tempesti	F	2004-10-31	5.318	2016-10-28

Összesen 154 543 sor.

19.Az első száz dolgozó adatai, úgy, hogy vezetéknév szerint vannak rendezve.

Result Grid						
Filter Rows: <input type="text"/>						
Edit: <input type="button" value="Edit"/> Export/Imp: <input type="button" value="Export/Imp"/>						
	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	36585	1969-01-11	Menkae	Aamodt	F	2001-02-07
	103125	1976-04-25	Huican	Aamodt	M	2001-12-12
	107094	1972-10-25	Baoqiu	Aamodt	M	2007-02-02
	203802	1970-04-25	Berry	Aamodt	M	2004-11-03
	216963	1975-07-16	Alois	Aamodt	M	2010-08-24
	107608	1975-09-23	Gretta	Aamodt	M	2003-05-04
	239949	1977-08-21	Zhiwei	Aamodt	F	2005-11-26
	206967	1970-08-28	Duke	Aamodt	F	2008-11-16
	102734	1971-05-31	Woody	Aamodt	M	2010-09-23
	242185	1970-02-18	Kazuhito	Aamodt	M	2001-12-02

Összesen 100 sor. Nyilván.

20.A következő lekérdezésben a dolgozó azonosítót, a vezetéknévét és az aktuális titulusát szeretném látni. Nyilván csak az aktív állomány adataira vagyok kíváncsi. A sorrendet az azonosító határozza meg.

Tipp: Ehhez???

Result Grid			
Filter Rows:			
	emp_no	last_name	title
▶	10001	Facello	Senior Engineer
	10002	Simmel	Staff
	10003	Bamford	Senior Engineer
	10004	Koblick	Senior Engineer
	10005	Maliniak	Senior Staff
	10006	Preusig	Senior Engineer
	10007	Zielinski	Senior Staff
	10009	Peac	Senior Engineer
	10010	Piveteau	Engineer
	10012	Bridoland	Senior Engineer

Összesen 240 124 sor.

21.Keressük azokat a dolgozókat, akikhez esetlegesen nem tartozik egyetlen titulus sem. Az egész tábla adataira vonatkozóan, nem csak aza aktív állomány.

Result Grid		
Filter Rows:		
	emp_no	last_name
*	NULL	NULL

Összesen 0 sor.

22.Írj három lekérdezést, amiben az első csak az első 100 dolgozót kéri be, a második a következő százat, míg a harmadik a harmadik száz dolgozót. Elég kiíratni az azonosítót a vezeték és keresztnéveket. A rendezés ez utóbbi kettő alapján történik.

I.

Result Grid			
Filter Rows:			
	emp_no	last_name	first_name
▶	258641	Aamodt	Abdelkader
	258005	Aamodt	Adhemar
	455773	Aamodt	Aemilian
	436560	Aamodt	Alagu
	266651	Aamodt	Aleksander
	487598	Aamodt	Alexius
	216963	Aamodt	Alois
	15427	Aamodt	Aluzio
	100860	Aamodt	Amabile
	107070	Aamodt	Anestis

II.

	emp_no	last_name	first_name
▶	264133	Aamodt	Maha
	36577	Aamodt	Mahmut
	276963	Aamodt	Maik
	455119	Aamodt	Mani
	417540	Aamodt	Marek
	486049	Aamodt	Mariangiola
	479557	Aamodt	Marie
	43174	Aamodt	Mariusz
	46884	Aamodt	Marla
	439427	Aamodt	Masamitsu

III.

	emp_no	last_name	first_name
▶	105323	Aamodt	Younwoo
	100010	Aamodt	Youpyo
	239949	Aamodt	Zhiwei
	83170	Aamodt	Ziya
	40184	Aamodt	Zvonko
	93347	Acton	Abdulla
	215122	Acton	Adit
	230078	Acton	Aimee
	255705	Acton	Aimee
	277402	Acton	Alassane

Összesen 100 sor minden lekérdezés esetén.

23. Most azokat a dolgozókat keressük, akiknek a vezetékgneve tartalmazza az „sb” karakter sorozatot. Minden adata a dolgozó táblából. A karakter sorozat bárhol lehet a vezetéknévben. Rendezés a vezetéknév szerint.

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	87497	1978-11-08	Kish	Bratsberg	F	2004-09-15
	79208	1971-06-15	Tamiya	Bratsberg	F	2007-04-26
	78400	1976-12-04	Hilary	Bratsberg	F	2000-09-26
	78076	1973-02-27	Anestis	Bratsberg	M	2000-05-16
	452245	1968-12-07	Jaroslava	Bratsberg	M	2006-09-15
	406114	1971-11-13	Akemi	Bratsberg	M	2002-11-03
	406014	1972-08-13	Masanao	Bratsberg	F	2003-03-31

Összesen 715 sor.

24.Számoljuk meg az azonos vezeték nevű dolgozókat. Minden vezetéknévnek szerepelnie kell a megszámlálásban. Sorrend a darabszám szerint növekvő.

Result Grid			Filter Rows:
	last_name	count(emp_no)	
▶	Sadowsky	145	
	Merro	147	
	Georgatos	148	
	Zykh	148	
	Guardalben	148	
	Rosar	150	
	Dulli	151	

Összesen 1637 sor.

25.Aktív dolgozók összes adata. Dolgozó táblából minden, az aktív titulus és a pillanatnyi fizetés összege.

Tipp: JOIN

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

Fetch rows:

	emp_no	birth_date	first_name	last_name	gender	hire_date	salary	title
▶	10001	1968-09-02	Georgi	Facello	M	2001-06-26	88958	Senior Engineer
	10002	1979-06-02	Bezalel	Simmel	F	2000-11-21	72527	Staff
	10003	1974-12-03	Parto	Bamford	M	2001-08-28	43311	Senior Engineer
	10004	1969-05-01	Chirstian	Koblick	M	2001-12-01	74057	Senior Engineer
	10005	1970-01-21	Kyoichi	Maliniak	M	2004-09-12	94692	Senior Staff
	10006	1968-04-20	Anneke	Preusig	F	2004-06-02	59755	Senior Engineer
	10007	1972-05-23	Tzvetan	Zielinski	F	2004-02-10	88070	Senior Staff

Result 16 ×

Összesen 240 124 sor.

26.Az összes adat közül a férfiak és nők száma.

Result Grid			Filter Rows:
	gender	count(emp_no)	
▶	M	179973	
	F	120051	

Összesen 2 sor.

27. Most a fizetés táblában levő összes, aktív és nem aktív fizetések számát szeretnénk látni.

Result Grid			
	all	not active salaries	active salaries
▶	2844047	2603923	240124

Összesen 1 sor.

28. A 10001 azonosítójú dolgozó eddigi összes fizetése. to_date szerint növekvő sorrendben. Mindkét táblából minden adat

Tipp: JOIN

Result Grid									
	emp_no	birth_date	first_name	last_name	gender	hire_date	salary	from_date	to_date
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	60117	2001-06-26	2002-06-26
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	62102	2002-06-26	2003-06-25
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	66074	2003-06-26	2004-06-25
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	66596	2004-06-25	2005-06-25
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	66961	2005-06-25	2006-06-25
▶	10001	1968-09-02	Georgi	Facello	M	2001-06-26	71046	2006-06-25	2007-06-24
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	74333	2007-06-24	2008-06-24

Összesen 17 sor.

29. A pillanatnyi állomány utolsó fizetésemelésének a dátuma. Adatok, amiket meg kell mutatni a képről leolvashatók. Sorrend fizetés from_date és dolgozó vezetéknev. Az eredményben nem lehet olyan fizetés, ami már nem aktív.

Result Grid							
	emp_no	last_name	first_name	hire_date	salary	from_date	to_date
▶	421185	Adachi	Oddvar	2000-08-06	93179	2016-08-02	9999-01-01
	420493	Albarhamtoshy	Yuriy	2000-07-17	53292	2016-08-02	9999-01-01
	433409	Alpay	Byong	2000-08-06	69940	2016-08-02	9999-01-01
	13292	Alpin	Bokyoung	2010-05-17	68769	2016-08-02	9999-01-01
	422292	Ananiadou	Theron	2007-08-04	106697	2016-08-02	9999-01-01
	281252	Angel	Lech	2006-08-05	62326	2016-08-02	9999-01-01
	99613	Angelov	Ziyad	2001-08-06	118590	2016-08-02	9999-01-01

Összesen 240 124 sor.

30.Minden adat a dept_man táblából.

emp_no	dept_no	from_date	to_date
110022	d001	2000-01-01	2006-10-01
110039	d001	2006-10-01	9999-01-01
110085	d002	2000-01-01	2004-12-17
110114	d002	2004-12-17	9999-01-01
110183	d003	2000-01-01	2007-03-21
110228	d003	2007-03-21	9999-01-01
110303	d004	2000-01-01	2003-09-09

Összesen 24 sor.

31.A pillanatnyi részlegvezetők adatai. Kereszt és vezetéknév szépen egy oszlopban, részleg neve, és a dolgozó titulusa. Sorrend a részleg neve növekvő.

dept_name	title	name
Customer Service	Manager	Weedman Yuchang
Development	Manager	DasSarma Leon
Finance	Manager	Legleitner Isamu
Human Resources	Manager	Sigstam Karsten
Marketing	Manager	Minakawa Vishwani
Production	Manager	Ghazalie Oscar
Quality Management	Manager	Pesch Dung

Összesen 9 sor. Mivel pont annyi részleg van.

32.Részlegek szerinti kimutatást szeretnénk az összes éves, havi fizetésre, minimum, maximum fizetésre, az átlag fizetésre és az aktív dolgozók számára. Természetesen a múlt most nem érdekel.

dept_name	yearly salaries	monthly salaries	avg(s.salary)	max(s.salary)	min(s.salary)	count(s.salary)
Customer Service	1,182,134,209.00	98.511.184,08	67285.2302	144866	39373	17569
Development	4,153,249,050.00	346.104.087,50	67657.9196	144434	39036	61386
Finance	977,049,936.00	81.420.828,00	78559.9370	142395	39012	12437
Human Resources	824,464,664.00	68.705.388,67	63921.8998	141953	38936	12898
Marketing	1,188,233,434.00	99.019.452,83	80058.8488	145128	39821	14842
Production	3,616,319,369.00	301.359.947,42	67843.3020	138273	38623	53304
Quality Management	851,010,736.00	70.925.093,00	65441.0034	133103	39042	14546

Összesen 9 sor.

33. Tíz százalék fizetésemelés azoknak a dolgozóknak, akik a legrégebben kaptak fizetésemelést. Persze ezt is egy lekérdezés dönti el nem égetünk be dátumot a kódba. Itt most az update utasítást kell használni. Az adatbázisunknak nem ez a filozófiája, de majd visszaállítjuk az adatokat. Eredetileg deaktiválni kéne a régi fizetést és egy új fizetést kéne rögzíteni.

Tipp: Abban az esetben, ha a következő hiba üzenetet kapod akkor még nem állítottad át biztonságos módot.

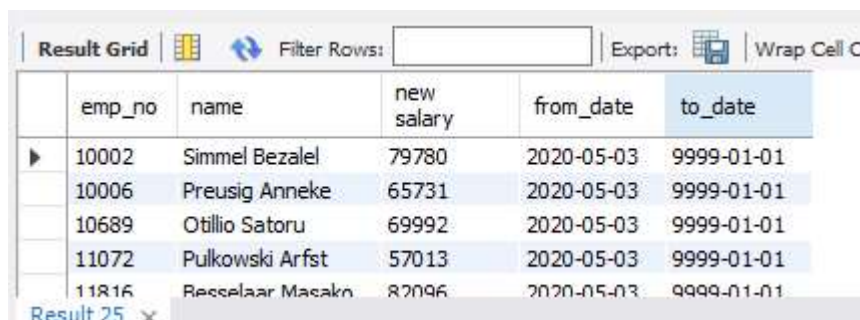
„Error Code: 1175. You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column”

Nézd meg a beállítások fejezetben hogyan kell ezt csinálni.

Összesen 650 sor érintve.

34. Listázzuk ki az előző feladatban megadott dolgozókat. Adatok a képen.

Tipp: Mire van szükséged, hogy megtaláld ezeket az adatokat?



The screenshot shows a database result grid with the following columns: emp_no, name, new salary, from_date, and to_date. The data is as follows:

emp_no	name	new salary	from_date	to_date
10002	Simmel Bezael	79780	2020-05-03	9999-01-01
10006	Preusig Anneke	65731	2020-05-03	9999-01-01
10689	Otilio Satoru	69992	2020-05-03	9999-01-01
11072	Pulkowski Arfst	57013	2020-05-03	9999-01-01
11816	Besselaar Masako	87096	2020-05-03	9999-01-01

Összesen 650 sor. Még szép, hogy annyi amennyi az update-ben is volt.

35. Válasz egy dolgozót, aki nem manager és aktív persze egy select segítségével. Majd bocsátsd el az állományból.

Tipp: Gondolkozz el mit, miket kell ehhez módosítani.

A választás eredménye (nálam), ez persze lehet más is:

Result Grid
@emp_no
10001

Egy gyors ellenőrző select a választásra:

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	emp_no	birth_date	first_name	last_name	gender	hire_date	emp_no	salary	from_date	to_date
	10001	1968-09-02	Georgi	Facello	M	2001-06-26	10001	88958	2017-06-22	9999-01-01

Az elbocsátás után a dolgozó fizetései. Az utolsó bejegyzés érdekel. A bekarikázott a to_date.

10001	1968-09-02	Georgi	Facello	M	2001-06-26	10001	85112	2015-06-22	2016-06-22
10001	1968-09-02	Georgi	Facello	M	2001-06-26	10001	85097	2016-06-22	2017-06-22
10001	1968-09-02	Georgi	Facello	M	2001-06-26	10001	88958	2017-06-22	2020-05-03

Összesen 3 sor érintett.

36. Mivel fenti szomorú eset többször is előfordul egy cég és egy dolgozó életében írj tárolt eljárást. A bemenő paraméter a dolgozó azonosító legyen, aki kilép. Az eljárás 0 értéket írjon ki, ha a művelet nem sikeres és 1-es értéket, ha minden rendben zajlott.

```

set @emp_no=(select e.emp_no from employees e inner join salaries s
on e.emp_no=s.emp_no
where s.to_date > curdate()
and e.emp_no not in (select emp_no from dept_manager) limit 1);
-- létező id
call dismiss_employee(@emp_no);

```

Eredménye:

Result Grid	Filter Rows:
result	
1	

```

} -- nem létező id
• call dismiss_employee(8);
.

```

Eredménye:

Result Grid		Filter
	result	
▶	0	

37. Következő eljárás sokkal kedvesebb számunkra. Olyan eljárást kell írni mely megemeli egy dolgozó fizetését. A paraméterei: a dolgozó azonosító, emelés mértéke százalékban egész szám, az emelés kezdetének a dátuma, és egy inout számláló melybe eggyel nagyobb érték kerül, ha sikeres volt a művelet.

```
2 • set @count = 0;
3 • call raise_salary(10005, 10, '2020-05-05', @count);
4 • select @count;
_
```

Result Grid		Filter
	@count	
▶	1	

Teszt a futtatás után:

```
6 • select * from salaries where emp_no=10005;
_
```

Result Grid		Filter Rows:	Ext
emp_no	salary	from_date	to_date
10005	90392	2013-09-10	2014-09-10
10005	90531	2014-09-10	2015-09-09
10005	91453	2015-09-09	2016-09-09
10005	94692	2016-09-09	2020-05-04
10005	104161	2020-05-05	9999-01-01

38. Persze a felhasználónak semmi sem jó. A következőben olyan eljárást kell írni mely egy feltételnek a where utáni részét várja paraméterként (sima szöveg varchar(500)) és ennek alapján több dolgozónak is megemeli a fizetését. A végén megmondja, hogy hány dolgozó volt érintett az

emelésben. Ezt egy inout változóban teszi. A paraméterek: where feltétel a where szócska utáni rész varchar(500), emelés mértéke százalékban int, emelés kezdetének a dátuma dátum, inout számláló

```
3 • set where_text=
4   'to_date > curdate() and from_date=(select min(from_date) from salaries where to_date > curdate())';
5 • set @count=0;
6 • call raise_salaries(where_text, 10, '2020-05-04', @count);
7 • select @count;
```

Itt azok a dolgozók kapnak fizetésemelést, akik már régen kaptak a többiekhez képest. Futtatás előtt állítsuk vissza a salaries táblát a backup fájlból. Visszaállítás után az érintett sorok száma.

Result Grid	
@count	
▶	650

A lekérdezés a 2016-08-02 from_date értékkel rendelkező dolgozóknak emeli meg a fizetését. Abban az esetben, ha újra lefuttatod az eljárást a sorok száma: 669 lesz.

39.Írj tárolt eljárást mellyel fel lehet vinni egy új dolgozót minden adatával együtt. Az adatokat az eljárás paramétereként kérd be a részleg legyen szöveg, ami, ha nem létezik a táblában nem folytatja a műveletet. Az új dolgozó azonosítót az eljárás állítsa elő. Ez legyen a visszatérési érték, amennyiben nem sikeres a művelet 0 értékkel térjen vissza az eljárás. Paraméterek: last_name, first_name, birth_date, gender, hire_date, deployment, salary, title, out new_id. A paraméterek típusa ugyanaz mint a megfelelő táblában levő típus.

```
2 • set @new_id = 0;
3 • call hire_employee('Kowasaki', 'Alan', '1995-05-05', 'M', '2020-05-05', 'development', 104250, 'Software developer', @new_id);
4 • select @new_id;
5   -- check
6 • select e.*, s.salary, t.title from employees e inner join salaries s
7   on e.emp_no=s.emp_no inner join titles t on e.emp_no=t.emp_no where s.to_date >= curdate()
8   and t.to_date >= curdate() and e.emp_no= @new_id;
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	emp_no	birth_date	first_name	last_name	gender	hire_date	salary	title
▶	500000	1995-05-05	Alan	Kowasaki	M	2020-05-05	104250	Software developer

Önálló, megoldás nélküli feladatok

40. Írj tárolt eljárást mely elbocsátja egy adott részleg vezetőjét és a paraméterben megadott azonosítójú dolgozót állítja be az új managernek. Használd bátran a már megírt eljárásokat, ha azok tudnak segíteni. Jelezd a művelet sikerét vagy bukását.
41. Írj tárolt eljárást mely törli az adatbázisból a tíz évnél régebbi adatokat. Persze csak azokat melyek már nem aktívak. Ez a feladat megint csak a gyakorlás végett került be, ebben az adatszerkezetben nem akarunk törölni.
42. Készíts egy táblát logs néven. Ebben a táblában jelenjen meg minden módosítás melyet a tárolt eljárások visznek végre. Legyen benne egy timestamp ami a módosítás idejét tartalmazza. A módosítást végző bejelentkezett felhasználó, (system változóból kiolvasható), a tábla neve, a módosítás típusa (insert, update, delete), és az érintett elsődleges kulcs.
43. Alakítsd át tárolt eljárásaidat, úgy, hogy vezessék az előbb elkészített logs táblát.
44. Írj egyszerű programot, ahonnan eléred az adatokat egy általad választott nyelven. Vigyázz, hogy ne olvass be egyszerre túl sok adatot. Az adatbázis műveleteket végezd a háttérben. Használd a programból a tárolt eljárásokat.
45. Végül. Találj ki egyéb lekérdezéseket a rendelkezésre álló adatok alapján. Írj minél több ilyet. Abban az esetben, ha támad valami remek ötleted küld el nekem is hátha bekerül a jegyzetbe. Jó szórakozást.

Megoldások

1. Ez egy nagyon egyszerű feladat túl sok magyarázat nem is tartozik hozzá. Mindig az elsődleges kulcsot érdemes számolni vagy egy olyan oszlopot mely nem tartalmazhat null értéket. A megoldáshoz a count() aggregáló függvényt használhatjuk az oszlopok átnevezéséhez pedig az „as” szócskát. Mivel szünet karaktert is használtunk az oszlop nevében ezért a szöveget aposztrófok közé kell tenni. Álljon itt az egyik tábla lekérdezése ez alapján a többi már könnyen megírható.

```
3
4 • select count(dept_no) as 'department count' from departments;
5
```

2. Mondtam, hogy nem nehézségi sorrendben vannak a feladatok. Itt már 3 táblát össze kell kapcsolni. Akkor sikerülhetett a feladat megoldása, ha elég figyelmesen tanulmányoztad az adatbázisban levő kapcsolatokat. A három tábla, amiket használni kell az a dolgozó, részleg és az ezekhez tartozó részleg_dolgozó kapcsolótábla. A feladatot itt két összekapcsolás segítségével oldottam meg. Jó megoldás a táblák felsorolása és egy where feltétel megadása is.

```
-
2 • select e.last_name, e.first_name, d.dept_name from employees e inner join dept_emp de
3   on e.emp_no=de.emp_no inner join departments d on de.dept_no=d.dept_no order by d.dept_name;
4
```

Figyeljük meg a táblanév rövidítések használatát, nem vagyunk gépirok. Ezt ott lehet megadni, ahol a táblát deklarálod a lekérdezésben de a rövid névre már előbb is lehet hivatkozni. Itt nem kell használni az „as” szócskát, mint az oszlop nevek megváltoztatása esetén. A kapcsolatok feltételének megadása az „on” szócska után történik és jelen esetben az elsődleges kulcsot köti össze az idegen kulccsal mindhárom tábla esetén. Itt most minden rekordot lekértünk nem csak az aktuális bejegyzéseket. Egy dolgozó többször is előfordulhat, ha részleget váltott. Vannak is ilyen dolgozók hiszen több sort kaptunk a lekérdezés eredményeként, mint ahány dolgozónk van.

3. Mint azt a tippben is láttad itt már csoportosítást is kellett használni a több táblás lekérdezésre. Fel kell használni az átlag, megszámlálás, összegzés aggregáló függvényeket. A részleg neve kell, hogy szerepeljen a group by után hiszen azt ki szeretnénk írni.


```

3 • select d.dept_name as 'department name', count(e.emp_no) as 'employee count',
4     avg(s.salary) as 'average salary', sum(s.salary) as 'salary sum'
5 from employees e inner join dept_emp de
6 on e.emp_no=de.emp_no inner join departments d on de.dept_no=d.dept_no
7 inner join salaries s on s.emp_no=e.emp_no group by d.dept_name order by d.dept_name;

```

Ebben az esetben négy táblát kellett összekapcsolni. Ezek között ott van a fizetés tábla is mely miatt a lekérdezés hosszan fut, bármilyen kevés sort is ad vissza.

4. UNCSI... Megint egy nagyon könnyű feladat. Egy sima kiválasztás egy rendezéssel, sehol egy JOINt.

```

3 • select * from departments order by dept_name;
4

```

5. A feladat nem túl nehéz. Minden problémánkra beépített függvények adtak megoldást. Én a megoldásomban bevezettem egy segéd változót, amiben az aktuális évet tároltam. Ezzel csak áttekinthetőbbé tettem a kódom. Hogy lehet ilyet csinálni?

```

2 • set @year_now=year(curdate());
3 • select @year_now;

```

Result Grid		Filter Rows:
	@year_now	
▶	2020	

A select utasítás csak az eredmény kiíratás miatt kellett. A year() metódus kiveszi az évet egy dátumból míg a curdate() visszaadja a mai dátumot és csak a dátumot. Abban az esetben ha az időre is szükség van, mondjuk egy log bejegyzéshez akkor a now() metódust használhatod. A felhasználó (programozó) által bevezetett változók neve a @ karakterrel kell, hogy kezdődjenek. A system változó nevei két darab kukaccal

kezdődnek. Innen kezdve már gyerekjáték az egész. Jöhet a végleges kód:

```
2 • set @year_now=year(curdate());
3
4 • select concat(last_name, " ", first_name) as 'name', birth_date as 'birth date',
5   @year_now - year(birth_date) as 'age', dayname(birth_date) as 'day of week' from employees
6   where day(birth_date) % 2 = 1 order by birth_date desc, last_name;
7
```

A két név összefűzésére a mysql concat() metódusát használtam. A kort a két év különbségéből számoltam, persze ebben a hónap nincs benne reméljük most nem sértődik meg egyik dolgozóm sem 😊. A nap nevének a kiírását megint csak a rendszerre bízta. A feltételben pedig csak a nap paritását kellett megvizsgálnom. A születési dátumot csökkenő sorrendbe kellett rendezni, hogy a legfiatalabb dolgozó kerüljön előre. Ne feledjük, hogy a változó csak akkor kap értéket ha lefuttatjuk azt a sort a lekérdezés előtt, egyébként null érték lesz benne.

6. Megint egy nem túl nehéz feladat. Ez az az eset amikor a lekérdezésben lekérdezést használhatunk. Szükségünk van a legkisebb dátumra a dolgozó táblából (ez lesz az al lekérdezés) és ennek az eredményét kell összevetni a felvétel dátumával. Természetesen itt is használhatunk változót a minimum dátum eltárolására.

Változó nélkül:

```
4
5 • select * from employees where hire_date=(select min(hire_date) from employees) order by last_name;
```

Ebben az esetben a belső lekérdezés egyetlen egy sorral tért vissza. A zárójel a szintaktika miatt szükséges a belső lekérdezéshez.

Változóval:

```
~
3 • select min(hire_date) into @min_date from employees;
4
5 • select * from employees where hire_date= @min_date order by last_name;
6
```

Látható, hogy a változót nem is kellett deklarálnom, hanem röptében hoztam létre a

select-into utasításban. Innen kezdve ez a változó létezik és ha nem feleljük el futtatni a sort akkor ugyanazt az eredményt kapjuk, mint az előbb. A változóinkban nem tudjuk egy több soros vagy oszlopos lekérdezés eredményét tárolni mindig csak egyetlen értéket.

7. Újra itt van újra itt van a Plain Old Group By (POGB). Bizony ezt a feladatot is a POGB segítségével oldhatjuk meg. Annyira egyszerű, hogy szinte szóra sem érdemes.

```
3 • select year(birth_date) as 'year of birth', count(emp_no) as 'employee count'
4   from employees group by year(birth_date);
5
```

Szerintem nincs mit magyarázni.

8. Abban az esetben, ha a having záradékkal próbáltad az elkészült lekérdezést kiegészíteni akkor hibát kaptál. Ez a kiértékelési sorrend miatt van. A POGB előtt kellett egy where záradékot írni. Így talán gyorsabb is a kiértékelés mert csak azokat az adatokat kell csoportosítani melyek majd szerepelnek a végeredményben.

```
3 • select year(birth_date) as 'year of birth', count(emp_no) as 'employee count'
4   from employees where year(birth_date) % 2 = 0 group by year(birth_date);
5
```

Ennyi.

9. Ez sem igényel túl sok magyarázatot.

```
3 • select avg(year(curdate())-year(birth_date)) as 'average age' from employees;
4
```

Csak egy kis számolgatás. Az átlag metódus paramétere két metódus eredményének a különbsége.

10. Group by miért vagy te group by. A monthname() metódus fogja visszaadni a hónap nevét.

```

3 • select monthname(birth_date) as 'month', count(emp_no) as 'employee count'
4   from employees group by month(birth_date) order by month(birth_date);

```

Ez biztos nem tartott egy percnél tovább, viszont nem is ér pontot 😞.

11. Megint a változókat használtam fel. Egy-egy változóba kiszámoltam az előző és következő hónapot majd ez vettem össze a születési dátum hónapjával.

```

3 • set @start_month = month(SUBDATE(curdate(), INTERVAL 1 month)),
4     @end_month = month(adddate(curdate(), interval 1 month));
5
6 • select * from employees where
7   month(birth_date) between @start_month and @end_month order by birth_date;
8







```

Subdate() és adddate() metódusok pont azt csinálják amit a nevük is ígér. Persze nekem most csak a hónap kell a kiszámított dátumból. Azért nem indítjuk el a szélső hónapokban a lekérdezést mert akkor a between alsó korlátja nagyobb lesz, mint a felső. Az pedig nem fog eredményt adni, mint a mellékelt ábra is mutatja.

```

3 • set @start_month = month(SUBDATE('2020-01-15', INTERVAL 1 month)),
4     @end_month = month(adddate('2020-01-15', interval 1 month));
5
6 • select * from employees where
7   year(birth_date) between @start_month and @end_month order by birth_date;
8
9

```

Result Grid						
Filter Rows: <input type="text"/>						
Edit:   						
Export/Import:  						
Wrap Cell Content: 						
	emp_no	birth_date	first_name	last_name	gender	hire_date
*	NULL	NULL	NULL	NULL	NULL	NULL

12. Megint változókat kell bevezetni, illetve újabb dátum kezelő függvényekkel ismerkedhetünk meg.

```

16 • set @start_day_of_year = dayofyear(subdate(curdate(), interval 2 day)),
17    @end_day_of_year = dayofyear(adddate(curdate(), interval 2 day));
18
19 • select * from employees where birth_date between makedate(year(birth_date), @start_day_of_year) and
20    makedate(year(birth_date), @end_day_of_year) order by birth_date;
21

```

A `dayofyear()` metódus visszaadja egy dátumból a napnak a sorszámát. Az index 0-tól indul. Ezekkel gyorsan megkerestem a kezdő, illetve a záró napnak a sorszámát. A lekérdezésben pedig ezen két változó segítségével „legyártom” a kezdő és vég dátumot, amibe a dolgozó születési évét használom fel. Ehhez a `makedate()` metódust hívtam segítségül. Ennek két attribútuma egy évszám és egy nap sorszám. Innen kezdve a dolgozó akkor felel meg a feltételnek, ha a születésnapja ezen két dátum közé (inclusive) esik. Persze itt is az a gond, hogy abban az esetben, ha mondjuk jan. 01-én szeretnénk elindítani a lekérdezést (bár akkor mindenki alszik), akkor nem kapunk eredményt. Ez persze nem azért van mert a szülő nők és a nőgyógyászok is aludtak minden év januárjában és nem születtek gyermekek, hanem az előbb felsorolt feladat megoldásában tárgyalt okok miatt.

13. Azt vettük észre, hogy abban az esetben, ha az intervallum eleje „átlóg” egy másik évbe akkor bizony az alsó korlát nagyobb lesz, mint a felső korlát. Első gondolat egy sima csere lehetne, azaz, ha a kezdő nap száma nagyobb mint a záró napé cseréljük meg a két változót. Persze hamar beláthatjuk, hogy ez a megoldás ugyan adna eredményt, de nem azt, amit várunk. Pont azokat a dolgozókat zárjuk ki, akiknek a születésnapjára kíváncsiak vagyunk. Tagadjuk le a feltételt? Ekkor a normál eset nem fog jól működni. Írjuk meg kétszer ugyanazt a lekérdezést A és B esetre? Ilyet nem teszünk. Ennyi időnk és energiánk nincs is. Egyébként is, aki majd futtatja a lekérdezést az elkezd számolgatni, hogy A vagy B eset áll fenn? Inkább hívjuk segítségül a tárolt eljárásokat melyekben el tudjuk helyezni az üzleti logikát. Ezek amúgy is előre le vannak fordítva, így a lekérdezés közben a motornak ezzel már nem kell bíbelődnie. Bontsuk kétfelé az időszakaszt, amit vizsgálni szeretnénk. Legyen egy év végi része és egy év eleji része. Ez akkor is működhet, ha a szakasz eleje és vége ugyanabban az évben van. Ekkor az első rész vége megegyezhet a második rész elejével. Mindenki látja már, hogy négy darab változóra lesz szükségem a

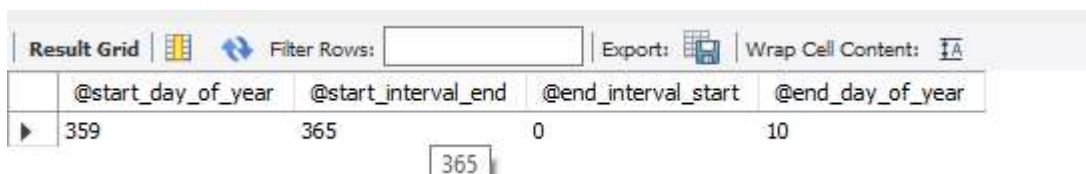
feltétel megírásához. Nézzük először ezeknek a kezdeti értékeit. A mai nap dátum legyen 2020-01-02 és n=8.

```
2 • set @start_day_of_year = dayofyear(subdate('2020-01-02', 8)),
3   @end_day_of_year = dayofyear(adddate('2020-01-02', 8)),
4   @start_interval_end=dayofyear('2020-01-02'),
5   @end_interval_start= dayofyear('2020-01-02');
6
```

Erre már tudunk írni egy lekérdezést, aminek a feltétel valahogy így nézne ki:

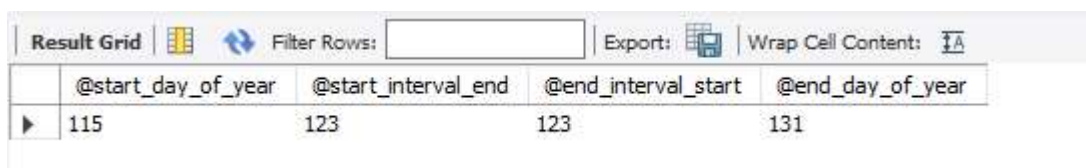
```
19
20 where (dayofyear( birth_date) between @start_day_of_year and @start_interval_end)
21 or (dayofyear(birth_date) between @end_interval_start and @end_day_of_year) order by birth_date;
22
```

Fenti képen csak a lekérdezés feltételét látjuk. Ez a sima esetben működhet is viszont a speciális „átlógó” esetben még mindig nem jó. A második esetben a fent megadott adatok esetén a változóink értékének valahogy így kéne kinéznie.



	@start_day_of_year	@start_interval_end	@end_interval_start	@end_day_of_year
▶	359	365	0	10

Erre már helyes eredményt kapunk az előbb látott feltétellel. Persze ha normál az eset akkor maradhatnak az előre beállított értékek. Legyen a dátum most 2020-05-02 az n=8. Ekkor ezt szeretnénk látni:



	@start_day_of_year	@start_interval_end	@end_interval_start	@end_day_of_year
▶	115	123	123	131

Most jöhet a tárolt eljárás, ami ezen négy értéket beállítja. A kezdeti értékeket is lehetne itt beállítani, de azt most az egyszerűség kedvéért meghagyom a lekérdezés elején. Úgy érezzük van egy kis problémánk. Hiszen a tárolt eljárások azok olyanok mint, a függvények. Egyetlen értéket tudnak előállítani. Az SQL-ben pedig nincs tömb, lista szerű

adatszerkezet, amiben elhelyezhetnénk a mi négy darab előállítandó értékünket. Most jön a jó hír. A tárolt eljárások bemenő paramétereit meg lehet jelölni az in, out vagy az inout szócskával. Az in ami a default érték azaz nem kell kiírni azt jelenti, hogy a paraméter bemenő típusú lesz. Ez a megszokott viselkedés. Az out azt ígéri az eljárás használójának (aki meghívja azt), hogy ha add egy változót paraméterként akkor az valamiféle értéket fog kapni mire az eljárás végez. Ennek a változónak nem kell, hogy legyen kezdő értéke. Míg a legutolsó módosító szó az inout azt várja el, hogy a kapott paraméternek legyen kezdő értéke és fenntartja magának a jogot, hogy ezt az értéket megváltoztathassa, ha arra van szükség. Itt azért gondolkozz el, hogy Te melyiket használnád a fenti három közül?

Helyes válasz persze, hogy az utolsót.

Mindjárt lehet két ilyen paramétere is az eljárásnak:

```
(start_day int, inout end_day int, start_interval_end int, inout end_interval_start int)
```

Persze itt csak a formális paraméter listát látjuk. Azok az attribútumok kaptak inout jelzőt melyeknek esetlegesen meg kell változni.

Hogyan lehet tárolt eljárást írni?

A navigátor ablakban az adatbázisod szerkezetében találsz egy Stored Procedures könyvtárat. Erre jobb klikk és Create Stored Procedure menüpont.



Ezek után már szerkesztheted is az eljárásodat.

```
new_procedure

1 • CREATE PROCEDURE `new_procedure` ()
2 BEGIN
3
4 END
5
```

A new_procedure-t írd át egy általad választott névre. Én a következő módon írtam meg a fejléctet.

```
3 CREATE PROCEDURE `day_set_for_birthday_select`
4 (start_day int, inout end_day int, start_interval_end int, inout end_interval_start int)
5 BEGIN
6
7 END
8
```

A paraméter listát már láttuk, az eljárás neve meg persze bármi lehet. Az eljárás törzse sem túl bonyolult. Hiszen csak érték cserét kell végrehajtani benne bizonyos feltétel alapján.

```
BEGIN
  if start_day > end_day
  then
    set start_interval_end=365;
    set end_interval_start=0;
  end if;
END
```

A teljes kód:

```

1 • CREATE PROCEDURE `day_set_for_birthday_select`
2   (start_day int, inout end_day int, start_interval_end int, inout end_interval_start int)
3   BEGIN
4     if start_day > end_day
5     then
6       set start_interval_end=365;
7       set end_interval_start=0;
8     end if;
9   END
10

```

Miután kész vagy már csak az Apply gombot kell megnyomnod és már használhatod is az eljárásodat. Teszteljük le! Azon a felületen, ahol a lekérdezést írod a következőt géped be.

```

2 • set @start_day_of_year = dayofyear(subdate('2020-01-02', 8 )),
3   @end_day_of_year = dayofyear(adddate('2020-01-02', 8 )),
4   @start_interval_end=dayofyear('2020-01-02'),
5   @end_interval_start= dayofyear('2020-01-02');
6
7 • call day_set_for_birthday_select
8   (@start_day_of_year, @end_day_of_year, @start_interval_end, @end_interval_start);
9
10 • select @start_day_of_year, @start_interval_end, @end_interval_start, @end_day_of_year;
11

```

Ennek az eredményét már fentebb láttuk, illetve most Te magad is láthatod. Több dátummal is teszteld le. Mindig minden dátumot változtass meg ugyanarra az értékre. Most már jöhet a lekérdezés, aminek a szövegét fenti sorok alá fogom írni.

```

• select * from employees where (dayofyear( birth_date) between @start_day_of_year and @start_interval_end)
or (dayofyear(birth_date) between @end_interval_start and @end_day_of_year) order by birth_date;

```

A teljes kód:

```

2 • set @start_day_of_year = dayofyear(subdate('2020-01-02', 8 )),
3   @end_day_of_year = dayofyear(adddate('2020-01-02', 8 )),
4   @start_interval_end=dayofyear('2020-01-02'),
5   @end_interval_start= dayofyear('2020-01-02');
6
7 • call day_set_for_birthday_select
8   (@start_day_of_year, @end_day_of_year, @start_interval_end, @end_interval_start);
9
10 • select @start_day_of_year, @start_interval_end, @end_interval_start, @end_day_of_year;
11
12 • select * from employees where (dayofyear( birth_date) between @start_day_of_year and @start_interval_end)
13   or (dayofyear(birth_date) between @end_interval_start and @end_day_of_year) order by birth_date;
14

```

Nem is olyan bonyolult. Ugye? Mint azt láttad az eljárást a call szócskával lehet működni bírni. Persze a napok kiírását az első select utasítást ki lehet hagyni és akkor csak egy result grid-t kapsz eredményként. Az eljárásod szövegén később a navigator ablakban az eljárás nevének végrehajtott jobb klikk és alter procedure menüpont segítségével tudsz változtatni.

14.

```

2 • select min(emp_no), max(emp_no) from employees;
3

```

15. Egy sima inner join megoldja a problémánkat

```

2 • select e.*, s.salary from employees e inner join salaries s
3   on e.emp_no=s.emp_no where s.salary> 30000;
4

```

16. Itt szinte semmi nem változott. A feltételt kellett kiegészíteni azzal, hogy a fizetés táblában a to_date legyen nagyobb vagy egyenlő a mai napnál.

```

2 • select e.*, s.salary from employees e inner join salaries s
3   on e.emp_no=s.emp_no where s.salary> 80000 and s.to_date >= curdate();
4

```

17.

```

2 • select format(avg(salary), 2, 'hu_HU') as 'average salary'
3   from salaries where to_date >= curdate();
4

```

A formázás a format() metódus segítségével történik melynek harmadik paramétere opcionális és a lokalizációt adja meg.

18.Ez megint egy olyan feladat amit mostanra a kisujjadból ráztál ki. Nyilván egy JOIN illetve egy röptében elvégzett osztás ami megoldja a problémát.

```

select e.*, format(s.salary/12, 0, 'hu_HU') as 'monthly salary', s.from_date
from employees e inner join salaries s on e.emp_no=s.emp_no
where s.salary > (select avg(salary) from salaries) and s.to_date >= curdate()
order by s.salary;

```

19.Amennyiben még nem ismerted akkor most megismerkedhettél a limit utasítással. Ennek van egy másik változata is melyet majd szintén fogunk használni.

```

2 • select * from employees order by last_name limit 100;

```

20.Egyszerű, ezért csak a megoldás áll itt magyarázat nélkül.

```

select e.emp_no, e.last_name, t.title
from employees e inner join titles t on e.emp_no=t.emp_no
where t.to_date >= curdate()
order by e.emp_no;

```

21.A megoldásban az in utasítást használtam egy al lekérdezés segítségével.

```

• select e.emp_no, e.last_name from employees e
  where e.emp_no not in (select emp_no from titles);

```


Nem kaptunk eredményt, nincs a keresésének megfelelő dolgozó. Persze ha kiveszük a not szócskát...

22. Az előbb megismert limit utasítás egy másik változatát használtam. Ebben meg lehet adni azt is, hogy mennyi sort szeretnénk bekérni.

```
4 • select emp_no, last_name, first_name
5   |from employees order by last_name, first_name limit 0, 100;
6 • select emp_no, last_name, first_name
7   from employees order by last_name, first_name limit 100, 100;
8 • select emp_no, last_name, first_name
9   from employees order by last_name, first_name limit 200, 100;
```

Az első paraméter a kezdő sor indexe. Itt is nullától kezdődik a számolás. Ez jól jöhet akkor, ha egy programban nem akarsz egyszerre nagy mennyiségű adatot beolvasni. Ekkor ezt a módszert, mint egy buffert tudod használni.

23. Nagyon könnyű. Régi barátunk a like és a százalék jel oldja meg a problémánkat.

```
select * from employees where last_name like '%sb%' order by last_name;
```

24. Megint egy semmiség. Group by és hurrá.

```
• select last_name, count(emp_no) from employees
  group by last_name order by count(emp_no);
```

25.

```
• select e.*, s.salary, t.title from employees e inner join salaries s
  on e.emp_no=s.emp_no inner join titles t on e.emp_no=t.emp_no
 where s.to_date >= curdate() and t.to_date >= curdate();
```

Nem hiszem, hogy túl sokat hozzá tudnék fűzni.

26. Persze, hogy a Group by.

- `select gender, count(emp_no)
from employees group by gender;`

27. Picikét talán nehezebb, mint az előző feladat. Nyilván hamar rájöttél, hogy az egyik számot el lehet tárolni egy változóban és akkor mér egyszerű a dolog, mint a pofon.

```
3 • set @count= (select count(emp_no) from salaries);
4
5 • select @count as 'all', count(emp_no) as 'not active salaries',
6   @count-count(emp_no) as 'active salaries' from salaries where to_date < curdate();
7
```

Használhattuk volna a select-into utasítást is változó értékének a megadására.

28.

- `select e.*, s.salary, s.from_date, s.to_date
from employees e inner join salaries s on e.emp_no=s.emp_no
where e.emp_no=10001 order by s.to_date;`

29. Természetesen JOIN. Annyira kell csak figyelni, hogy a fizetés táblában a to_date legalább a mai napi dátum legyen vagy annál nagyobb.

- `select e.emp_no, e.last_name, e.first_name, e.hire_date, s.salary, s.from_date, s.to_date
from employees e inner join salaries s on e.emp_no =s.emp_no
where s.to_date >=curdate() order by s.from_date, e.last_name;`

30. Nem vicceltem. Kicsit tanulmányozd a tábla adatait felépítését mert fogjuk használni.

- 2 • `select * from dept_manager;`

HA-HA-HA!

31. Annyi a különlegesség, hogy most négy táblát kellett összekapcsolni.

Egyébként meg arra kellett figyelni, hogy a dolgozó legyen aktív. Ez akkor is teljesül ha éppen Ő a részleg vezetője. Ezen adatot pedig a dept_man tábla to_date oszlopából lehet kiolvasni.

```
2 • select d.dept_name, t.title, concat(e.last_name, " ", e.first_name) as 'name'
3   from employees e inner join dept_manager dm on e.emp_no=dm.emp_no inner join
4   departments d on d.dept_no= dm.dept_no inner join titles t on e.emp_no= t.emp_no
5   where dm.to_date >= curdate()
6   and t.to_date >= curdate() order by d.dept_name;
```

Hosszú de nem túl bonyolult.

32. Megint egy hosszú kód néhány JOIN-al és egy Group by-al.

```
2 • select d.dept_name, format(sum(s.salary), 2) as 'yearly salaries',
3   format(sum(s.salary)/12, 2, 'hu_HU') as 'monthly salaries',
4   avg(s.salary), max(s.salary), min(s.salary), count(s.salary) from departments d
5   inner join dept_emp de on d.dept_no=de.dept_no
6   inner join employees e on e.emp_no=de.emp_no inner join
7   salaries s on s.emp_no=e.emp_no where s.to_date >=curdate()
8   and de.to_date >= curdate() group by d.dept_name;
```

Van mit gépelni.

33. Egy update utasítást kell írni. Ennek a feltétele kell, hogy kiválogassa a megadottaknak megfelelő dolgozókat. A fizetés táblában kell módosítani az összeget és a from_date mező értékét. Arra is figyelni kell, hogy csak az aktív dolgozók fizetését szabad változtatni.

```
• update salaries set salary=salary*1.10, from_date=curdate()
  where to_date > curdate() and from_date=(select min(from_date)
  from salaries where to_date >= curdate());
```

34. Egy egyszerű JOIN a fizetés és dolgozó tábla között, ahol azokat az adatokat keresem melyeknek a from_date oszlopa a maximum értéket képviseli a fizetés táblában. Ehhez vagy egy változó, vagy egy belső select szükséges. Én ez utóbbi megoldást választottam.

```

2 • select e.emp_no, concat(e.last_name, " ", e.first_name) as 'name',
3   s.salary as 'new salary', s.from_date, s.to_date
4   from employees e inner join salaries s on e.emp_no=s.emp_no
5   where s.from_date = (select max(from_date) from salaries where to_date>=curdate());

```

35. Biztos teljesen világos, hogy ez nem egy művelet. Haladjunk lépésről lépésre. Kezdjük a választással. Egy aktív dolgozóra van szükségünk, aki nem manager. Ennek megfelelő select utasítást kell írni és ennek eredményét be kell tenni egy változóba.

```

• set @emp_no=(select e.emp_no from employees e inner join salaries s
  on e.emp_no=s.emp_no where s.to_date > curdate()
  and e.emp_no not in (select emp_no from dept_manager) limit 1);
• select @emp_no;

```

Mivel az employee id-ra volt szükségem ezért egy JOIN-nal összefűztem a két táblát. Megoldható lenne e nélkül is? Egy belső select segítségével pedig kiszűrtem az esetleges managert. Mivel egy változóban kellett tárolnom az értéket ezért a limit 1, hogy csak egy találatom legyen. A következő lépés, hogy az érintett táblákat módosítom. Mennyi tábláról is van szó pontosan?

Igen 3 darab a fizetés a titulus, és a részleg. Ezek már sima update utasítások.

```

• update salaries set to_date = curdate() where to_date>=curdate() and emp_no = @emp_no;
• update titles set to_date = curdate() where to_date>=curdate() and emp_no = @emp_no;
• update dept_emp set to_date = curdate() where to_date>=curdate() and emp_no = @emp_no;

```

Jogos, ha azt kérded, hogy mi van, ha valamelyik nem sikerül a 3 módosítás közül? Igen ezt egy tranzakcióban kellett volna csinálni. Ezt majd a következő feladatban valósítjuk meg.

Miért is van erre szükség?

Gondold csak el mi történik, ha az első update lefut, de a másik kettő már nem. Lesz egy dolgozónk, akinek nem lesz fizetése viszont lesz neki aktív titulusa és a részleg is nyilván fogja tartani, mint aktív dolgozót.

36. Mivel a feladata kicsit összetettebb haladjunk lépésről lépésre. Kezdjük annak az ellenőrzésével, hogy a dolgozó aktív státuszban van. Abban az esetben, ha nem akkor 0-t kell kiírni. Ez lesz az eljárásunk első lépése.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `dismiss_employee`(emp_id int)
BEGIN
  declare result int default 0;
  if exists (select e.emp_no from employees e inner join salaries s on e.emp_no=s.emp_no
  where s.to_date > curdate() and e.emp_no=emp_id)
  then
    set result = 1;
  end if;
  select result;
END
```

Bevezettem egy result nevű változót melynek a default értéke 0. Ezek után felhasználtam az exists vizsgálatot egy elágazásban. Most a teszt miatt ha a megadott id aktív dolgozót takar a result értékét átállítom 1-re. Az elágazás után kiválasztom a result nevű változóm. Ez lesz az eljárás eredménye, ez nem keverendő össze a visszatérési értékkel. Ezen a ponton tesztelhetjük az eljárást.

Nézzük a következő lépést. Szükség lesz egy hibakezelőre. Ezt szintén deklarálni kell és meg kell neki adni az akciót melyet végre kell, hogy hajtson akkor, ha hiba keletkezett. Ez a deklaráció az utolsó kell, hogy legyen ezért az előbb deklarált result sora után írjuk be a következőket:

```
3   declare result int default 0;
4   declare exit Handler for sqlexception
5   begin
6     rollback;
7     set result = 0;
8   end;
```

Azt írtuk, hogyha hiba van akkor állítsa vissza az eddigi változásokat és állítsa a result változó értékét nullára majd lépjen ki. Akit mélyebben is érdekel a dolog az olvassa el a dokumentációt.

Most jöhet a tranzakció melynek a rollback utasítását már a kivétel kezelőben láttuk. Ennek az elágazás törzsébe kell kerülnie.

```

then
  start transaction;
  -- ide kell beilleszteni a három tábla update utasítását.
  set result = 1;
  commit;
end if;

```

Innen kezdve már egyszerű a dolog a megjegyzés helyére beillesztjük az előző feladatban megírt 3 db update utasítást. Abban a szerencsés esetben, ha a kód eljut a commit sorig akkor mindhárom tábla módosítása sikeres volt és a commit el is menti ezeket a változásokat. Hiba esetén pedig a handler visszaforgatja a tranzakció által a hiba keletkezéséig megtörtént módosításokat. Most már nem maradhatunk „áldatlan” állapotban 😊.

A teljes kód:

```

CREATE DEFINER=`root`@`localhost` PROCEDURE `dismiss_employee`(emp_id int)
BEGIN
  declare result int default 0;
  declare exit Handler for sqlexception
begin
  rollback;
  set result = 0;
end;

if exists (select e.emp_no from employees e inner join salaries s on e.emp_no=s.emp_no
where s.to_date > curdate() and e.emp_no=emp_id)
then
  start transaction;
  update salaries set to_date = curdate() where to_date>=curdate() and emp_no = @emp_no;
  update titles set to_date = curdate() where to_date>=curdate() and emp_no = @emp_no;
  update dept_emp set to_date = curdate() where to_date>=curdate() and emp_no = @emp_no;
  set result = 1;
  commit;
end if;
select result;
END

```

Ennyi! Mehet a tömeges elbocsátás 😞.

37. Megint egy tárolt eljárást kell írni és ebben is tranzakciót kell használni, nem tehetjük meg szegény dolgozóval, hogy a régi fizetését deaktiváljuk

míg az újat nem visszük be a rendszerbe. Nézzük részleteiben az eljárás készítésének a lépéseit. Miután megírtad a fejléct a következő módon:

```
2
3 CREATE PROCEDURE `raise_salary`
4 (emp_id int, percentage int, start_date date, inout count int)
5 BEGIN
6
7 END
```

kezdhetjük a változók beállítását. Rögtön szükségünk lesz egy szorzóra, amivel a régi fizetést meg tudjuk emelni. Ehhez némi egyszerű számításra van szükség. Persze jó érzékkel rögtön arra gondoltál, hogy milyen jó lenne, ha ez kikerülhetne egy külön eljárásba. Jó hír: nem csak tárolt eljárást, de függvényt is tudsz írni. Jobb klikk a navigátor ablakban található functions menüre, majd create function.

```
CREATE FUNCTION `new_function` ()
RETURNS INTEGER
BEGIN
RETURN 1;
END
```

Láthatod, hogy az új függvény visszatérési típusa integer, ami nekünk nem lesz jó. A nevét is szeretnénk megváltoztatni és egy paraméterre is szüksége lesz.

```
3 CREATE FUNCTION `set_multiplier` (percentage int)
4 RETURNS DOUBLE
```

A változtatás után a fejléc a fenti módon néz ki. Persze függvényednek akármilyen nevet adhatsz a Te gyermeked.
Jöhet a lényeg:

```

3 CREATE FUNCTION `set_multiplier` (percentage int)
4 RETURNS DOUBLE
5 BEGIN
6 RETURN 1 + percentage / 100;
7 END

```

A kód nyilván nem túl bonyolult. Persze ezzel pont ez volt a célunk, hogy a kódunk olvashatóbb és érthetőbb legyen. Apply gomb és jöhet a teszt. Ott, ahol a lekérdezést írod:

```
12 • select set_multiplier(12);
```

és az eredmény:

Result Grid	
	set_multiplier(12)
▶	1.12

Most már deklarálhatjuk a tárolt eljárásban a változóinkat:

```

CREATE PROCEDURE `raise_salary`
(emp_id int, percentage int, start_date date, inout count int)
BEGIN
  declare multiplier real default set_multiplier(percentage);
  declare new_salary int;
  declare old_end_date date default subdate(start_date, 1);
  declare exit Handler for sqlexception
begin
  rollback;
end;
END

```

A szorzó egyértelműen a most megírt függvénytől kap kezdő értéket. Az új fizetés nem kap egyelőre, értéket. A harmadik változónk az előző fizetés vég napját állítja be. Ilyet már sokszor láttunk ezért nem nagyon magyarázom. Ezek után már csak a kivételt kezelő kódot kellett megírni, ezt is láttuk már, annyi a különbség az előzőhöz képest, hogy most nem

kell a számláló értékét állítanom. Abban az esetben, ha hiba van úgy hagyom ahogy kaptam. Itt most csak a tranzakciót forgatom vissza. A következő sorban beállíthatjuk az új fizetés értékét.

```
rollback;
end;
set new_salary = round((select salary*multiplier from salaries where emp_no=emp_id and to_date > curdate()));
```

Az itt előállított értéket fel lehet használni az ellenőrzésre is. Az ellenőrzés után már csak a tranzakció és az update, insert utasítások hiányoznak.

```
14  if (new_salary is not null)
15  then
16    start transaction;
17    update salaries set to_date=old_end_date where emp_no=emp_id and to_date > curdate();
18    insert salaries values(emp_id, new_salary, start_date, '9999-01-01');
19    set count=count+1;
20    commit;
21  end if;
22  END
```

Kész is vagyunk. Apply és mehet a teszt.

```
3  CREATE PROCEDURE `raise_salary`
4  (emp_id int, percentage int, start_date date, inout count int)
5  BEGIN
6    declare multiplier real default set_multiplier(percentage);
7    declare new_salary int;
8    declare old_end_date date default subdate(start_date, 1);
9    declare exit Handler for sqlexception
10   begin
11     rollback;
12   end;
13   set new_salary = round((select salary*multiplier from salaries where emp_no=emp_id and to_date > curdate()));
14   if (new_salary is not null)
15   then
16     start transaction;
17     update salaries set to_date=old_end_date where emp_no=emp_id and to_date > curdate();
18     insert salaries values(emp_id, new_salary, start_date, '9999-01-01');
19     set count=count+1;
20     commit;
21   end if;
22   END
```

38. Ez egy olyan feladat melyet az életben nem biztos, hogy akarsz alkalmazni. Azért vettem mégis ide, hogy meg tudjunk ismerkedni az execute utasítással. Ezen kívül megismerhetjük még a cursort is. A legnagyobb baj az, hogy egy sima szöveget kapunk paraméterként melyből egy select utasítást kell gyártani és ezt le is kell futtatni. Egyébként ez a „hibás” része az elgondolásnak. Fogalmad sincs mit fogsz lefuttatni akár kaphatsz drop table utasítást is a másik programozótól. Ezért a gyakorlatban nem biztos, hogy akarsz ilyet írni. Nézzük megint a lépéseket, amivel elkészítjük az eljárásunkat. Első lépés a változók deklarálása.

```
3 CREATE PROCEDURE `raise_salaries`  
4 (where_text varchar(500), percentage int, start_date date, inout count int)  
5 BEGIN  
6     DECLARE selected_emp_no int;  
7     DECLARE done BOOLEAN DEFAULT FALSE;  
8     DECLARE cur CURSOR FOR SELECT emp_no FROM vw_emp_nos;  
9     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
0  
1     END  
-
```

A legelső változót a leválogatás eredményének bejárásakor fogom felhasználni. Mivel több fizetést szeretnék megemelni ezért ebbe fogom mindig eltárolni az éppen aktuális dolgozó azonosítóját. A második egy sima bool amivel majd megállítjuk a ciklusunkat. A következő a cursor amibe egy view-nak (vw_emp_nos) fog az eredménye bekerülni. Ezt nézetet fogjuk létrehozni a kapott feltétel segítségével és ezt fogjuk bejárni a cursorral. A continue handler akkor lendül működésbe, ha a cursor-ban nincs adat. Ekkor a ciklusom el sem fog indulni nem lesz mit bejárni. Nézzük, hogy tudjuk a where_text paraméterből elkészíteni a select utasítást és futtatni azt. Ezek a sorok a deklarációk után jönnek.

```

8      drop view if exists vw_emp_nos;
9      SET @select = concat('CREATE VIEW vw_emp_nos as SELECT emp_no FROM salaries where ', where_text, ');');
10     PREPARE statement FROM @select;
11     EXECUTE statement;
12     DEALLOCATE PREPARE statement;
13

```

Első körben eldobom az előző futtatáskor elkészült view-t. Elvileg csak akkor marad meg ha valami baj volt a futtatás alatt ez csak biztonsági intézkedés. Ezek után kiegészítem a kapott paramétert a create view utasítással mely a select alapján fogja leválogatni az emelésben részt vevő dolgozók azonosítóját. A következő három sor az elkészült select futtatását végzi el. Ezek végére elkészül a view. A következő lépésben már csak be kell járni a cursort. Miért került a view tartalma a cursorba? Figyeld meg a cursor deklarációs sorát és rá fogsz jönni. Mehet a bejárás:

```

14     OPEN cur;
15     read_loop: LOOP
16         FETCH cur INTO selected_emp_no;
17         IF done THEN
18             LEAVE read_loop;
19         end if;
20         call raise_salary(selected_emp_no, percentage, start_date, count);
21     END LOOP read_loop;
22     CLOSE cur;
23     DROP VIEW vw_emp_nos;

```

Ezzel tulajdonképpen végeztünk is. Legelőször megnyitom az előzőekben elkészült cursort. Egy ciklusban elkezdem bejárni. Minden egyes lépés esetén a selected_emp_no változóba berakom az aktuális azonosítót. Abban az esetben, ha nincs már sor a cursoban kilépek a ciklusból. Ez az elágazás igaz ágának a dolga. Még a ciklusban meghívom az előző feladatban megírt fizetésemelés eljárásomat a kiválasztott dolgozó azonosítóval. A count változóm értéke azért növekszik mert a fizetésemelés eljárás minden lépésben eggyel növeli a kapott inout változó értékét. A végén csak be kell zárni a cursort illetve el kell dobni az ideiglenesen elkészített view-t. A teljes kód.


```

3 CREATE PROCEDURE `raise_salaries`
4 (where_text varchar(500), percentage int, start_date date, inout count int)
5 BEGIN
6     DECLARE selected_emp_no int;
7     DECLARE done BOOLEAN DEFAULT FALSE;
8     DECLARE cur CURSOR FOR SELECT emp_no FROM vw_emp_nos;
9     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
10    DROP VIEW IF EXISTS vw_emp_nos;
11    SET @select = concat('CREATE VIEW vw_emp_nos as SELECT emp_no FROM salaries where ', where_text, '');
12    PREPARE statement FROM @select;
13    EXECUTE statement;
14    DEALLOCATE PREPARE statement;
15    OPEN cur;
16    read_loop: LOOP
17        FETCH cur INTO selected_emp_no;
18        IF done THEN
19            LEAVE read_loop;
20        end if;
21        call raise_salary(selected_emp_no, percentage, start_date, count);
22    END LOOP read_loop;
23    CLOSE cur;
24    DROP VIEW vw_emp_nos;
25 END

```

39. Könnyű észrevenni, hogy ebben az esetben nem egy, hanem mindjárt négy táblába is bele kell írni. Rögtön a tranzakció írás jut az eszünkbe. Nézzük megint lépésenként az eljárásunkat. Fejléc.

```

4 CREATE DEFINER=`root`@`localhost` PROCEDURE `hire_employee`(last_name varchar(16),
5 first_name varchar(14),
6 birth_date date, gender enum('M', 'F'), hire_date date, dept_name varchar(40),
7 salary int(11), title varchar(50), out new_id int(11))
8 BEGIN

```

Semmi extra, csak a megfelelő típusú és elengedhetetlen paraméterek. A következő jól megszokott lépés a változók bevezetése.

```

9      declare dept_no char(4) default
10      (select dept_no from departments d where d.dept_name=dept_name);
11      declare end_date date default '9999-01-01';
12      declare exit Handler for sqlexception
13  begin
14      set new_id=0;
15      rollback;
16  end;

```

Az end_date változót csak olvashatóság és kényelmi okból vezettem be. A dept_no változó kezdeti értékébe megpróbálom beolvasni egy létező részleg azonosítóját. Ezt fogom ellenőrizni. Abban az esetben, ha a részleg létezik semmi akadálya, hogy eltároljuk a dolgozónkat. Jöhet az ellenőrzés és a tárolás természetesen tranzakcióban.

```

17      set new_id=0;
18      if (dept_no is not null)
19  then
20      set new_id=(select max(emp_no) from employees)+1;
21      start transaction;
22      insert into employees values(new_id, birth_date, first_name,
23      last_name, gender, hire_date);
24      insert into salaries values(new_id, salary, hire_date, end_date);
25      insert into dept_emp values(new_id, dept_no, hire_date, end_date);
26      insert into titles values(new_id, title, hire_date, end_date);
27      commit;
28  end if;
29  END

```

Első körben az új azonosítót állítom elő, úgy, hogy kiolvasom a legutolsó értéket a dolgozó táblából majd megnövelem azt eggyel. Kezdődhet a tranzakció amiben négy sima insert utasítás van. Szerintem nem igényelnek magyarázatot. A legvégén pedig a commit mely siker esetén kiírja a változásokat az adatbázisba. Nem is bonyolult.

A kód egyben:

```
4 CREATE DEFINER=`root`@`localhost` PROCEDURE `hire_employee`(last_name varchar(16),
5 first_name varchar(14),
6 birth_date date, gender enum('M', 'F'), hire_date date, dept_name varchar(40),
7 salary int(11), title varchar(50), out new_id int(11))
8 BEGIN
9     declare dept_no char(4) default
10     (select dept_no from departments d where d.dept_name=dept_name);
11     declare end_date date default '9999-01-01';
12     declare exit Handler for sqlexception
13     begin
14         set new_id=0;
15         rollback;
16     end;
17     set new_id=0;
18     if (dept_no is not null)
19     then
20         set new_id=(select max(emp_no) from employees)+1;
21         start transaction;
22         insert into employees values(new_id, birth_date, first_name,
23         last_name, gender, hire_date);
24         insert into salaries values(new_id, salary, hire_date, end_date);
25         insert into dept_emp values(new_id, dept_no, hire_date, end_date);
26         insert into titles values(new_id, title, hire_date, end_date);
27         commit;
28     end if;
29 END
```