Adatbáziskezelés

# Adatbázis

Az **adatbázis** olyan számítógépen tárolt és rendezett adathalmaz, melynek elemei összetartoznak. Az adatbázisok célja, hogy az adatokat hosszútávon, tartósan lehessen tárolni, illetve, hogy a tartalma gyorsan kereshető legyen.

**Relációs adatbázis**nak nevezzük azokat az adatbázisokat, amelyekben az adatok a relációs adatmodell elvén vannak szervezve. A relációs adatmodell legfontosabb eleme a matematika reláció fogalma, mely a Descartes-szorzat részhalmazát jelenti. Egy relációs adatbázis esetében az adatok a gyors hozzáférhetőség érdekében kapcsolatokkal vannak meghatározva.

Ezek kezelésére adatbázis-kezelőket használunk. Az **SQL** (ami az angol Structured Query Language szóra utal) egy relációsadatbázis-kezelő lekérdezési nyelv. Bár nem tekintjük programozási nyelvnek, de mégis hasonlóan kell elképzelni, itt is vannak utasítások, amelyek segítségével műveleteket végezhetünk. Létrehozhatunk adatbázisokat, adatokat adhatunk az adatbázishoz, törölhetünk belőle, módosíthatjuk az egyes adatokat, lekérdezhetjük azokat, vagy rendezhetjük. Természetesen ez nem minden.

Jellemző feladatok:

* adatbázisokat létrehozni és törölni
* táblákat létrehozni az adatbázisban
* lekérdezéseket végrehajtani egy adatbázisban
* adatokat beszúrni vagy törölni egy adatbázisból
* már meglévő adatokat frissíteni az adatbázisban

## Példa 1

Képzeljünk el egy online játékot, ahol valós ellenfelek ellen játszhatunk. Ebben a játékban lehet pontot, pénzt szerezni. A pénzt el lehet költeni tárgyakra, míg a pont alapján egy ranglétrát állít fel a rendszer. Mivel online játékról beszélünk, az adatokat egy olyan helyen kell tárolni, amihez a felhasználók nem férnek hozzá, hiszen, ha a saját telefonjukon, számítógépükön tárolnánk a pénzük mennyiségét, akkor felhasználók át tudnák írni, ezzel több pénzt szerezni maguknak. Ezért azokat az adatokat, amelyekhez nem szeretnénk, ha hozzáférnének a játékosok, azokat egy adatbázisba fogjuk tárolni.

## Példa 2

Egy közösségi oldalon ugyebár létre lehet hozni profilokat, tartalmakat lehet feltölteni, ismerősöket keresni. A profilokhoz tartozó adatokat, azaz például a felhasználónevet, a jelszót, az email címet, a születési dátumot adatbázisban szokás tárolni, csak úgy, mint a közösség médiában jelenlévő kapcsolatokat, viszonyokat.

## Fogalmak

* Egyed: Egyednek nevezzük azt, aminek az adatait tároljuk az adatbázisban.
  + Például a közösségi oldalas példánál egy profil.
* Attribútum: Az egyed egy jellemzője.
  + Az előző példánál maradva egy profil egy adata, például a vezetéknév, vagy telefonszám.
  + Magyarul tulajdonság
* Tábla: A logikailag összetartozó adatokat egy táblában tároljuk.
  + Például egy táblában tároljuk a regisztrált felhasználókat.
* Record: Rekordnak nevezzük az adatbázis egy sorát.
  + Azaz, ha van egy adatbázisunk, amelyben személyek adatait tároljuk (például név és kor), akkor minden felvitt személy külön rekordnak tekinthető.
* Az adatbázis egy oszlopa, azaz egy egyed attribútuma.
  + Például az előző fogalomnál említett példa esetében külön mezőnek tekinthető a név és a kor is. Gyakran csak oszlopként hivatkozunk rá.

# Az SQL utasításai

* Adatdefiníciós nyelv (Data Definition Language – DDL)
  + adatbázisok sémájának definiálására használják
  + jellemző utasításai: CREATE, ALTER, DROP
* Adatlekérdező nyelv (Data Query Language – DQL)
  + adatok lekérdezéséhez, lehívásához használják
  + jellemző utasításai: SELECT, FROM, WHERE, GROUP BY, HAVING, ORDER BY
* Adatmanipulációs nyelv (Data Manipulation Language – DML)
  + adatok létrehozására, módosítására, törlésére használják
  + jellemző utasításai: INSERT, UPDATE, DELETE
* Adatelérést vezérlő nyelv (Data Control Language – DCL)
  + jogosultságok adása
  + jellemző utasítások: GRANT, REVOKE

## Adatdefiníciós nyelv

Az adatdefiníciós nyelv olyan utasításokat tartalmaz, amelyek az adatbázis elemekkel kapcsolatos műveleteket végeznek, másképpen kifejezve az adatbázisok sémáját határozhatjuk meg segítségükkel. Például adatbázist, táblát, felhasználókat hoz létre, módosít vagy töröl.

* CREATE: Adatbázis objektum létrehozását szolgálja, leggyakrabban táblákat, nézeteket, idexeket hozunk létre vele.
* ALTER: A már létező adatbázis objektumokat tudjuk módosítani vele.
* DROP: Már létező adatbázis objektumokat tudunk törölni vele. Törli magát az adatbázis objektumot és a tartalmát is.
* TRUNCATE: Ha a DROP utasítást használjuk egy adatbázistáblára, akkor az törli az adatbázis tartalmát és magát az adatbázist is, míg a TRUNCATE csak az adatbázis tartalmát törli, de az adatbázist meghagyja.

## Adatlekérdező nyelv

Az adatlekérdező nyelvre sokan csak szimplán lekérdező nyelvként hivatkoznak, angolul Data Query Language-nek (DQL) hívják.

Mint ahogy a neve is utal rá az adatlekérdező nyelv az adatok lekérdezésére használható. Az adatbázisban tárolt adatokat ezen utasítások segítségével kérhetjük le, rendezhetjük sorba vagy csoportosíthatjuk.

* SELECT: A lekérdezés fő utasítása, itt azokat az oszlopokat kell megadnunk, amelyek eredményét látni szeretnénk.
* FROM: A FROM kulcsszó után kell megadnunk azokat a táblákat, amelyekből szeretnénk adatokat lekérdezni.
* WHERE: Segítségével szűrési feltételeket határozhatunk meg. Csak azok a rekordok kerülnek kilistázásra, amelyek kielégítik a feltételeket.
* GROUP BY: Sorok csoportosítását, összevonását érhetjük el vele.
* HAVING: Segítségével a csoportosítás utáni eredményhalmazon fogalmazhatunk meg újabb szűrési feltételeket.
* ORDER BY: A lekérdezés eredményét rendezhetjük, állíthatjuk sorba valamilyen szempont alapján.

## Adatmanipulációs nyelv

Az adatmanipulációs nyelv utasításai az adatok hozzáadását, módosítását, törlését valósítják meg.

* INSERT INTO: Ezzel az utasítással adatokat adhatunk hozzá egy táblához.
* UPDATE: Módosításokat, frissítéseket hajthatunk végre a már létező adatokon.
* DELETE FROM: Segítségével törölhetjük a feltételeknek megfelelő adatokat, rekordokat.

## Adatelérést vezérlő nyelv

Magyarul adatelérést vezérlő nyelvnek, vagy adatvezérlő nyelvnek, angolul pedig Data Control Language-nek (DCL) nevezik. Ezen nyelvi utasítások a jogosultságok kezelését, szabályozását valósítják meg.

* GRANT: Különböző jogosultságokat adhatunk vele a felhasználóknak.
* REVOKE: Jogosultságokat vehetünk el vele felhasználóktól.

# Infrastruktúra (legalábbis, amit mi használunk)

MySQL Server: Ez valósítja meg az adatbázis kezelését. Az SQL szabványnak megfelelően biztosít szolgáltatásokat.

MySQL Connector: A python programokban (ezért kell telepíteni a venv-be, mint modul) ez illeszkedik a szolgáltatásokhoz, amiket a szerver nyújt. Lehetőséget ad arra, hogy python kódból kommunikáljunk a MySQL szerverünkkel. Ez egy driver.

Python: Itt tudjuk megírni a logikát, amiben aztán igénybe tudjuk venni az adatbázis szolgáltatásokat.

### MySQL Server telepítése

Angol nyelvű leírások:

<https://realpython.com/python-mysql/#installing-mysql-server>

<https://dev.mysql.com/doc/refman/5.7/en/installing.html>

Windows install:

<https://dev.mysql.com/downloads/installer/>

Mac install:

<https://dev.mysql.com/doc/refman/5.7/en/macos-installation.html>

Szükséges majd a hostname, username és a password a későbbiekben, mert az adatbázisok eléréséhez meg kell majd adni.

### MySQL Connector telepítése

* Környezetbe: pip install mysql-connector-python
* Kódba: import mysql.connector

## Csatlakozás MySQL szerverhez

A MySQL-kiszolgálóval való kommunikáció első lépése a kapcsolat létrehozása. Ehhez a mysql.connector modulban a connect() függvényt kell használnod. Ez a függvény paramétereket vár, mint például a hoszt, felhasználó és jelszó, és visszaad egy MySQLConnection objektumot. Az felhasználótól megkapott hitelesítő adatokat bemenetként kaphatod, majd átadhatod azokat a connect() függvénynek.

from getpass import getpass

from mysql.connector import connect, Error

try:

with connect(

host="localhost",

user=input("Enter username: "),

password=getpass("Enter password: "),

) as connection:

print(connection)

except Error as e:

print(e)

A fenti kód a megadott bejelentkezési hitelesítő adatokat használja fel a MySQL-kiszolgálóval való kapcsolat létrehozásához. Visszaad egy MySQLConnection objektumot, amit a connection változóban tárolunk. Innen ezt a változót tudjuk használni a MySQL-kiszolgálóhoz való hozzáféréshez.

A fenti kódban néhány fontos dologra kell figyelni:

* Mindig kezelni kell azokat a kivételeket, amelyek a MySQL-kiszolgálóhoz való kapcsolat létrehozása közben felmerülhetnek. Erre használjuk a *try ... except* blokkot, hogy elkapd és kiírd az esetlegesen előforduló kivételeket.
* Mindig be kell zárnod a kapcsolatot, miután végeztél az adatbázis elérésével. Az használaton kívüli nyitott kapcsolatok váratlan hibákhoz és teljesítményproblémákhoz vezethetnek. A fenti kód a *with* segítségével kontextuskezelőt használ, amely elrejti a kapcsolat bezárásának folyamatát.
* Sosem szabad beégetned a bejelentkezési hitelesítő adataidat, azaz a felhasználónevedet és jelszavadat, közvetlenül egy Python scriptbe. Ez rossz gyakorlat a telepítéshez és komoly biztonsági fenyegetést jelent. A fenti kód bekéri a felhasználótól a bejelentkezési hitelesítő adatokat. A beépített getpass modult használja a jelszó elrejtésére. Habár ez jobb megoldás, mint beégetni, vannak más, biztonságosabb módok érzékeny információk tárolására, például környezeti változók használata (mint pl a chrome, amikor megjegyzi a bejelentkezéseidet).

Most már létrehoztunk egy kapcsolatot a Python program és a MySQL-kiszolgál között, de még mindig létre kell hoznod egy új adatbázist, vagy csatlakoznod kell egy meglévő adatbázishoz a szerveren belül.

## Adatbázis létrehozása

Az előző részben létrehoztunk egy kapcsolatot a MySQL-kiszolgálóval. Ahhoz, hogy létrehozzunk egy új adatbázist, a következő SQL utasítást kell végrehajtani:

CREATE DATABASE books\_db;

A fenti utasítás létrehoz egy új adatbázist a books\_db névvel.

Megjegyzés: A MySQL-ben kötelező pontosvesszőt (;) tenni egy utasítás végére, ami a **lekérdezés** befejezését jelöli. Azonban a MySQL Connector/Python automatikusan hozzáfűzi a pontosvesszőt a lekérdezéseid végére, így nincs szükség rá a Python kódodban.

Egy SQL lekérdezés végrehajtásához Pythonban szükséged lesz egy kurzorra (cursor), ami elrejti az adatbázisrekordokhoz való hozzáférést. A MySQL Connector/Python biztosítja a MySQLCursor osztályt, amely objektumokat példányosít, és képes MySQL lekérdezések végrehajtására Pythonban. A MySQLCursor osztály egy példánya is kurzornak nevezhető.

A kurzorobjektumok egy MySQLConnection objektumot használnak a kommunikációhoz a MySQL-kiszolgálóval. Kurzort létrehozni a connection változód .cursor() metódusával tudsz:

cursor = connection.cursor()

A fenti kód egy MySQLCursor osztály példányát adja vissza neked.

Egy végrehajtandó lekérdezést a cursor.execute() metódusba kell szöveges formában elküldeni. Ebben az adott esetben a CREATE DATABASE lekérdezést fogod elküldeni a cursor.execute()-nak:

from getpass import getpass

from mysql.connector import connect, Error

try:

with connect(

host="localhost",

user=input("Enter username: "),

password=getpass("Enter password: "),

) as connection:

create\_db\_query = "CREATE DATABASE online\_movie\_rating"

with connection.cursor() as cursor:

cursor.execute(create\_db\_query)

except Error as e:

print(e)

A fenti kód végrehajtása után egy új adatbázis lesz online\_movie\_rating nével a MySQL-kiszolgálón.

A CREATE DATABASE lekérdezés egy sztring formájában van tárolva a create\_db\_query változóban, majd átadódik a cursor.execute()-nak végrehajtásra. A kód egy kontextuskezelőt használ a kurzorobjektummal a bezárási folyamat kezeléséhez.

Itt hibát kaphatsz, ha már létezik egy azonos nevű adatbázis a szervereden. Ezt megerősítheted azzal, hogy megjeleníted az összes adatbázis nevét a szervereden. Az előzőleg használt MySQLConnection objektummal végrehajtod a SHOW DATABASES utasítást:

show\_db\_query = "SHOW DATABASES"

with connection.cursor() as cursor:

cursor.execute(show\_db\_query)

for db in cursor:

print(db)

A fenti kód kiírja az összes adatbázis nevét, amely jelenleg a MySQL-kiszolgálódon található. A SHOW DATABASES parancs néhány olyan adatbázist is kiír, amelyet te nem hoztál létre a szervereden, mint például az information\_schema, performance\_schema, stb. Ezek az adatbázisok automatikusan generálódnak a MySQL-kiszolgáló által, és hozzáférést biztosítanak különböző adatbázis-metaadatokhoz és MySQL-kiszolgáló beállításokhoz.

Ebben a részben létrehoztunk egy új adatbázist a CREATE DATABASE utasítás végrehajtásával. A következő részben megnézzük, hogyan csatlakozhatsz egy már létező adatbázishoz.

## Csatlakozás adatbázishoz

Az előző részben létrehoztunk egy új adatbázist online\_movie\_rating néven. Azonban még nem csatlakoztunk hozzá. Sok helyzetben már rendelkezel egy meglévő MySQL adatbázissal, amelyhez kapcsolódni szeretnél a Python alkalmazásoddal.

from getpass import getpass

from mysql.connector import connect, Error

try:

with connect(

host="localhost",

user=input("Enter username: "),

password=getpass("Enter password: "),

database="online\_movie\_rating",

) as connection:

print(connection)

except Error as e:

print(e)

Ezt megtehetjük ugyanazzal a connect() függvénnyel, amit korábban használtunk, egy további paraméterrel, amit database-nek neveznek:

A fenti kód nagyon hasonló a korábban használt kapcsolatszkripthez. Az egyetlen változás itt egy további database paraméter, amelynek segítségével az adatbázis neve átadásra kerül a connect() függvénynek. Miután végrehajtod ezt a szkriptet, csatlakozva leszel az online\_movie\_rating adatbázishoz.

## Adattáblák létrehozása, változtatása és törlése

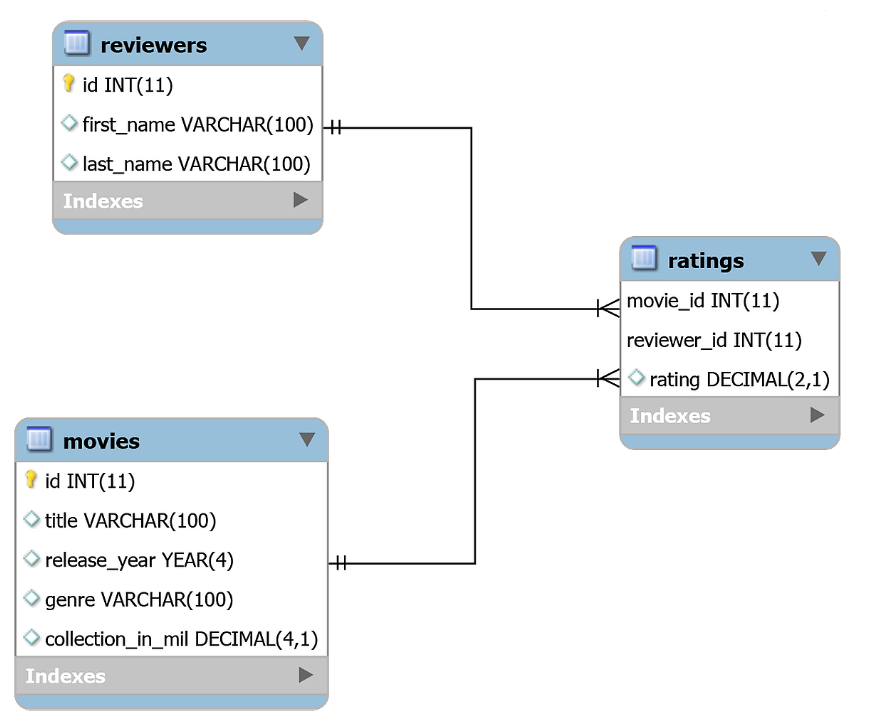
Most végzünk néhány alapvető DDL (adatleíró nyelv) lekérdezést Pythonnal, mint például a CREATE, DROP és ALTER. Áttekintő feladat lesz a MySQL adatbázisról, amire alapul majd a későbbi rész.

Elkezdünk összerakni egy online filmértékelő rendszer adatbázis-sémáját. Az adatbázis három táblát fog tartalmazni:

* movies tartalmazza a filmek általános információit, és a következő attribútumokkal rendelkezik:
  + id
  + title
  + release\_year
  + genre
  + collection\_in\_mil
* reviewers tartalmazza azoknak az embereknek az információit, akik véleményeket vagy értékeléseket küldtek be, és a következő attribútumokkal rendelkezik:
  + id
  + first\_name
  + last\_name
* ratings tartalmazza a beküldött értékelések információit, és a következő attribútumokkal rendelkezik:
  + movie\_id (idegen kulcs)
  + reviewer\_id (idegen kulcs)
  + rating

Egy valóságban használt filmértékelő rendszer, mint például az IMDb, sok más attribútumot is tárolna, mint például e-maileket, a film szereplőlistáit, stb. Ha szeretnéd, hozzáadhatsz további táblákat és attribútumokat az adatbázishoz. De ezek a három tábla elegendőek lesznek a feladathoz.

Az alábbi kép bemutatja az adatbázis sémáját:



Az adatbázisban található táblák kapcsolatban állnak egymással. A movies és reviewers táblák között many-to-many kapcsolat lesz, mivel egy filmhez több értékelő is rendelkezhet, és egy értékelő több filmet is értékelhet. A ratings tábla kapcsolatot teremt a movies tábla és a reviewers tábla között.

### Create table

Új tábla létrehozására az MySQL-ben a CREATE TABLE utasítást használjuk. Az alábbi MySQL query létrehozza a movies táblát az online\_movie\_rating adatbázisban:

CREATE TABLE movies(

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100),

release\_year YEAR(4),

genre VARCHAR(100),

collection\_in\_mil INT

);

Korábbi SQL utasítások alapján, a fenti lekérdezés nagy része érthető lehet. Azonban van néhány különbség a MySQL szintaxisban, amit érdemes tisztázni.

Például, a MySQL-nek számos adattípusa van, beleértve az INT, BIGINT, YEAR stb. Továbbá, a MySQL az AUTO\_INCREMENT kulcsszót használja, amikor egy oszlop értéke automatikusan növekszik az új rekordok beszúrásakor. Ezeket itt nem részletezzük, de bátorítok arra, hogy nézzetek utána.

Az új tábla létrehozásához ezt a lekérdezést kell átadnod a cursor.execute()-nak, amely elfogad egy MySQL lekérdezést, majd végrehajtja a lekérdezést a kapcsolódott MySQL adatbázison:

create\_movies\_table\_query = """

CREATE TABLE movies(

id INT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(100),

release\_year YEAR(4),

genre VARCHAR(100),

collection\_in\_mil INT

)

"""

with connection.cursor() as cursor:

cursor.execute(create\_movies\_table\_query)

connection.commit()

Most már rendelkezel a movies táblával az adatbázisodban. A create\_movies\_table\_query-t átadod a cursor.execute()-nak, amely elvégzi a szükséges végrehajtást.

Megjegyzés: A connection változó az a MySQLConnection objektumra utal, amelyet a kapcsolódás során kaptál vissza az adatbázishoz.

Ezenkívül vedd észre a connection.commit() utasítást a kód végén. Alapértelmezés szerint a MySQL csatlakozód nem végzi el az autocommit tranzakciókat. A MySQL-ben a tranzakcióban megadott módosítások csak akkor történnek meg, ha az utolsó sorban COMMIT parancsot használsz. Minden tranzakció után hívd meg ezt a módszert a tényleges tábla módosításához.

Ahogy a movies táblánál tettük, végrehajtjuk a következő szkriptet is a reviewers tábla létrehozásához:

create\_reviewers\_table\_query = """

CREATE TABLE reviewers (

id INT AUTO\_INCREMENT PRIMARY KEY,

first\_name VARCHAR(100),

last\_name VARCHAR(100)

)

"""

with connection.cursor() as cursor:

cursor.execute(create\_reviewers\_table\_query)

connection.commit()

Amennyiben igény van rá, hozzáadhatsz több információt egy értékelőről, például az e-mail címüket vagy demográfiai adatokat. De a first\_name és last\_name jelenleg elegendő nekünk.

Végül létrehozhatod a ratings táblát a következő szkript segítségével:

create\_ratings\_table\_query = """

CREATE TABLE ratings (

movie\_id INT,

reviewer\_id INT,

rating DECIMAL(2,1),

FOREIGN KEY(movie\_id) REFERENCES movies(id),

FOREIGN KEY(reviewer\_id) REFERENCES reviewers(id),

PRIMARY KEY(movie\_id, reviewer\_id)

)

"""

with connection.cursor() as cursor:

cursor.execute(create\_ratings\_table\_query)

connection.commit()

A külső kulcs (foreign key) kapcsolatok implementációja a MySQL-ben kissé eltér a szabványos SQL-től és korlátozottabb. A MySQL-ben mind a szülő, mind a gyermek táblának ugyanazt a storage engine-t kell használnia.

Egy storage engine az az alapvető szoftverkomponens, amelyet az adatbázis-kezelő rendszer használ az SQL műveletek végrehajtására. A MySQL-ben két különböző típusú storage engine létezik:

* Tranzakcionális storage engine-ek tranzakció biztonságosak, és lehetővé teszik a tranzakciók visszavonását egyszerű parancsok, például a rollback használatával. Sok népszerű MySQL motor, például az InnoDB és az NDB, ebbe a kategóriába tartozik.
* Nem tranzakcionális tárolómotorok bonyolult kézi kódot igényelnek a műveletek visszavonásához. Az MyISAM, MEMORY és sok más MySQL motor nem tranzakcionális.

Az InnoDB a MySQL alapértelmezett és a legnépszerűbb storage engine.je. Segít fenntartani az adatintegritást a foreign key [megszorítások](https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html) támogatásával. Ez azt jelenti, hogy bármely CRUD műveletet egy foreign key-en ellenőriznek annak érdekében, hogy ne vezessen inkonzisztenciákhoz különböző táblák között.

Továbbá, vegyük észre, hogy a ratings tábla a movie\_id és reviewer\_id oszlopokat, mindkettőt foreign key-ként, együttesen használja elsődleges kulcsként. Ez a lépés biztosítja, hogy egy értékelő ne tudja kétszer értékelni ugyanazt a filmet.

Lehetőséged van arra, hogy ugyanazt a cursort többször is használd (execute). Ebben az esetben az összes végrehajtás egy [atomi](https://en.wikipedia.org/wiki/Atomicity_(database_systems)) tranzakcióvá válik, nem pedig különálló tranzakciókká. Például végrehajthatod az összes CREATE TABLE utasítást ugyanazzal a cursorral, majd csak egyszer adod ki a tranzakció véglegesítését:

with connection.cursor() as cursor:

cursor.execute(create\_movies\_table\_query)

cursor.execute(create\_reviewers\_table\_query)

cursor.execute(create\_ratings\_table\_query)

connection.commit()

A fenti kód először végrehajtja az összes CREATE utasítást. Ezután COMMIT parancsot küld a MySQL szervernek, amely véglegesíti a tranzakciót. A .rollback() metódust is használhatod a ROLLBACK parancs küldéséhez a MySQL szervernek, amely visszavonja az összes adatmódosítást a tranzakcióból.

# Describe, alter és drop

## Describe

Most, hogy létrehoztuk az összes táblát, megtekintheted a sémaikat az alábbi SQL utasítással:

DESCRIBE <table\_name>;

A cursor objektumból eredményeket lekérni cursor.fetchall() módszerrel lehetséges. Ez a módszer az utoljára végrehajtott utasítás összes sorát lekéri. Feltételezve, hogy már rendelkezel a MySQLConnection objektummal a connection változóban, ki tudod íratni az összes eredményt, amit a cursor.fetchall() lekért:

show\_table\_query = "DESCRIBE movies"

with connection.cursor() as cursor:

cursor.execute(show\_table\_query)

# Fetch rows from last executed query

result = cursor.fetchall()

for row in result:

print(row)

Miután végrehajtjuk a fenti kódot, egy táblázatot kell kapnunk, amely információkat tartalmaz a movies tábla minden oszlopáról. Minden oszlop esetén részletes információkat kapsz, például az oszlop adattípusát, hogy az oszlop elsődleges kulcs-e vagy sem, és így tovább.

## Alter

A movies táblában van egy collection\_in\_mil nevű oszlop, amely a filmek jegybevételét tartalmazza millió dollárban. A következő MySQL utasítást írhatod, hogy módosítsd a collection\_in\_mil attribútum adattípusát INT-ről DECIMAL-ra:

ALTER TABLE movies MODIFY COLUMN collection\_in\_mil DECIMAL(4,1);

A DECIMAL(4,1) azt jelenti, hogy egy decimális szám, amelynek legfeljebb 4 számjegye lehet, amelyből 1 tizedesjegy, például 120.1, 3.4, 38.0, stb. Az ALTER TABLE utasítás végrehajtása után megtekintheted az aktualizált táblasémát a DESCRIBE használatával:

alter\_table\_query = """

ALTER TABLE movies

MODIFY COLUMN collection\_in\_mil DECIMAL(4,1)

"""

show\_table\_query = "DESCRIBE movies"

with connection.cursor() as cursor:

cursor.execute(alter\_table\_query)

cursor.execute(show\_table\_query)

# Fetch rows from last executed query

result = cursor.fetchall()

print("Movie Table Schema after alteration:")

for row in result:

print(row)

Ahogy az eredményben látható, a collection\_in\_mil attribútum most DECIMAL(4,1) típusú. Figyeld meg, hogy a fenti kódban kétszer hívod meg a cursor.execute() metódust. De a cursor.fetchall() csak az utoljára végrehajtott lekérdezés sorait kéri le, amely a show\_table\_query.

## Delete

Egy tábla törléséhez a DROP TABLE utasítást kell végrehajtanod MySQL-ben. A tábla törlése visszafordíthatatlan folyamat.

Ha végrehajtod az alábbi kódot, akkor a későbbi szakaszokban újra ki kell adnod a CREATE TABLE query-t a ratings tábla használatához.

A ratings tábla törléséhez küldd el a drop\_table\_query-t a cursor.execute()-nek:

drop\_table\_query = "DROP TABLE ratings"

with connection.cursor() as cursor:

cursor.execute(drop\_table\_query)

Ha végrehajtod a fenti kódot, törlöd a ratings táblát.

# Recordok beillesztése

Az eddigiekben három táblát hoztunk létre az adatbázisban: movies, reviewers és ratings. Most feltöltjük ezeket a táblákat adatokkal. Két különböző módot fogunk áttekinteni a rekordok beszúrására a MySQL Connector for Python segítségével.

Az első módszer, a .execute() jól működik, ha kevés rekordról van szó, és a rekordokat kézzel lehet megadni. A második módszer, a .executemany(), népszerűbb és jobban alkalmazható valós világbeli helyzetekben.

## .execute()

Az első megközelítés ugyanazt a cursor.execute() módszert használja, amit eddig is használtunk. Létrehozol egy INSERT INTO query-t string formájában, majd átadod a cursor.execute()-nak. Ezt a módszert használhatod adatok beszúrásához a movies táblába.

Emlékeztetésül a movies táblának öt attribútuma van:

1. id
2. title
3. release\_year
4. genre
5. collection\_in\_mil

Nem kell adatot hozzáadnod az id-hoz, mivel az AUTO\_INCREMENT automatikusan kiszámolja az id-t számodra. Az alábbi script beszúrja a rekordokat a movies táblába:

insert\_movies\_query = """

INSERT INTO movies (title, release\_year, genre, collection\_in\_mil)

VALUES

("Forrest Gump", 1994, "Drama", 330.2),

("3 Idiots", 2009, "Drama", 2.4),

("Eternal Sunshine of the Spotless Mind", 2004, "Drama", 34.5),

("Good Will Hunting", 1997, "Drama", 138.1),

("Skyfall", 2012, "Action", 304.6),

("Gladiator", 2000, "Action", 188.7),

("Black", 2005, "Drama", 3.0),

("Titanic", 1997, "Romance", 659.2),

("The Shawshank Redemption", 1994, "Drama",28.4),

("Home Alone", 1990, "Comedy", 286.9)

"""

with connection.cursor() as cursor:

cursor.execute(insert\_movies\_query)

connection.commit()

A movies tábla most már tíz rekordot tartalmaz. A kód a végén meghívja a connection.commit()-ot. Nagyon fontos, hogy minden módosítás után meghívd a .commit()-ot a táblában.

## .executemany()

Az előző megközelítés alkalmazható, amikor a rekordok száma viszonylag kicsi, és ezeket a rekordokat közvetlenül a kódba tudod írni. De ez ritkán igaz. Gyakran előfordul, hogy az adatok más fájlban vannak tárolva, vagy az adatokat másik script generálja, és hozzá kell adni őket a MySQL adatbázishoz.

Erre az esetre van az .executemany() parancs. Két paramétert fogad:

1. Egy lekérdezést, amely placeholdereket tartalmaz a beszúrni kívánt rekordok számára.
2. Egy lista, amely az összes beszúrni kívánt rekordot tartalmazza.

Az alábbi példa beszúrja a rekordokat a reviewers táblába:

insert\_reviewers\_query = """

INSERT INTO reviewers

(first\_name, last\_name)

VALUES ( %s, %s )

"""

reviewers\_records = [

("Chaitanya", "Baweja"),

("Mary", "Cooper"),

("John", "Wayne"),

("Thomas", "Stoneman"),

("Penny", "Hofstadter"),

("Mitchell", "Marsh"),

("Wyatt", "Skaggs"),

("Sheldon", "Cooper"),

("Kimbra", "Masters"),

("Amy", "Farah Fowler")

]

with connection.cursor() as cursor:

cursor.executemany(insert\_reviewers\_query, reviewers\_records)

connection.commit()

A fenti kódban mind a lekérdezést, mind pedig a rekordok listáját átadod az .executemany() metódusnak. Ezek a rekordok lehetnek fájlból származó adatok vagy felhasználótól begyűjtött adatok, amelyeket a reviewers\_records listában tárolsz.

A kód a %s-t használja a két string placeholderként az insert\_reviewers\_query-ben. A placeholderek a formázó specifikátorokként működnek, és helyet foglalnak egy változónak a stringen belül. Az adott változó ezen a helyen kerül hozzáadásra a végrehajtás során.

Hasonlóképpen használhatod az .executemany() metódust rekordok beszúrására a ratings táblába is:

insert\_ratings\_query = """

INSERT INTO ratings

(rating, movie\_id, reviewer\_id)

VALUES ( %s, %s, %s)

"""

ratings\_records = [

(6.4, 17, 5), (5.6, 19, 1), (6.3, 22, 14), (5.1, 21, 17),

(5.0, 5, 5), (6.5, 21, 5), (8.5, 30, 13), (9.7, 6, 4),

(8.5, 24, 12), (9.9, 14, 9), (8.7, 26, 14), (9.9, 6, 10),

(5.1, 30, 6), (5.4, 18, 16), (6.2, 6, 20), (7.3, 21, 19),

(8.1, 17, 18), (5.0, 7, 2), (9.8, 23, 3), (8.0, 22, 9),

(8.5, 11, 13), (5.0, 5, 11), (5.7, 8, 2), (7.6, 25, 19),

(5.2, 18, 15), (9.7, 13, 3), (5.8, 18, 8), (5.8, 30, 15),

(8.4, 21, 18), (6.2, 23, 16), (7.0, 10, 18), (9.5, 30, 20),

(8.9, 3, 19), (6.4, 12, 2), (7.8, 12, 22), (9.9, 15, 13),

(7.5, 20, 17), (9.0, 25, 6), (8.5, 23, 2), (5.3, 30, 17),

(6.4, 5, 10), (8.1, 5, 21), (5.7, 22, 1), (6.3, 28, 4),

(9.8, 13, 1)

]

with connection.cursor() as cursor:

cursor.executemany(insert\_ratings\_query, ratings\_records)

connection.commit()

Az összes táblában most már vannak adatok. Van egy teljesen működőképes online filmértékelő adatbázisunk. A következőkben megnézzük, hogyan tudsz kommunikálni ezzel az adatbázissal.

# Keresés adatbázisokban

## Select

Az adatok lekéréséhez egy SELECT lekérdezést kell küldened a cursor.execute() metódussal. Ezután a cursor.fetchall() segítségével kinyerheted a visszakapott táblát, egy sorokból vagy rekordokból álló lista formájában.

Próbálj megírni egy MySQL lekérdezést, amely kiválasztja az összes rekordot a movies táblából, és küldd el a .execute() metódusnak:

select\_movies\_query = "SELECT \* FROM movies LIMIT 5"

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

result = cursor.fetchall()

for row in result:

print(row)

A result változó tartalmazza a .fetchall() használatával visszakapott rekordokat. Ez egy tuple-kből álló lista, amely az egyes rekordokat reprezentálja a táblából:

(1, 'Forrest Gump', 1994, 'Drama', Decimal('330.2'))

(2, '3 Idiots', 2009, 'Drama', Decimal('2.4'))

(3, 'Eternal Sunshine of the Spotless Mind', 2004, 'Drama', Decimal('34.5'))

(4, 'Good Will Hunting', 1997, 'Drama', Decimal('138.1'))

(5, 'Skyfall', 2012, 'Action', Decimal('304.6'))

A fenti lekérdezésben a LIMIT clause segítségével korlátozod a SELECT utasítással visszakapott sorok számát. A fejlesztők gyakran használják a LIMIT-et, hogy a lapozást megvalósítsák nagy adatmennyiségek kezelésekor.

SELECT \* FROM movies LIMIT 2,5;

A MySQL-ben a LIMIT clause egy vagy két nemnegatív numerikus argumentumot fogad. Egy argumentum használatakor megadod a visszaadandó sorok maximális számát. Mivel a lekérdezésed tartalmazza a LIMIT 5-t, csak az első 5 rekordot fogja visszaadni. Mindkét argumentum használatakor megadod az első visszaadandó sor eltolását (offset) is:

Az első argumentum 2-es eltolást jelöl, a második argumentum pedig a visszaadandó sorok számát korlátozza 5-re. Az fenti lekérdezés 3-tól 7-ig fog visszaadni sorokat.

Lehetőséged van csak kiválasztott oszlopok lekérdezésére is:

select\_movies\_query = "SELECT title, release\_year FROM movies LIMIT 5"

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for row in cursor.fetchall():

print(row)

('Forrest Gump', 1994)

('3 Idiots', 2009)

('Eternal Sunshine of the Spotless Mind', 2004)

('Good Will Hunting', 1997)

('Skyfall', 2012)

Most a kód csak az adott két oszlop értékét jeleníti meg: cím és megjelenési év.

## Where

A WHERE clause-al specifikus feltételek alapján szűrheted a táblarekordokat. Például, ha szeretnéd visszakapni az összes olyan filmet, amelynek boxoffice bevétele több mint 300 millió dollár, a következő lekérdezést futtathatod:

SELECT title, collection\_in\_mil

FROM movies

WHERE collection\_in\_mil > 300;

Az ORDER BY clause-t is használhatod az előző lekérdezésben a visszakapott eredmények rendezésére a legmagasabbtól a legalacsonyabbig:

select\_movies\_query = """

SELECT title, collection\_in\_mil

FROM movies

WHERE collection\_in\_mil > 300

ORDER BY collection\_in\_mil DESC

"""

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for movie in cursor.fetchall():

print(movie)

A MySQL számos string formázási műveletet kínál, például a CONCAT-ot a karakterláncok összefűzésére. Gyakran előfordul, hogy a weboldalak a filmcímet a megjelenési évvel együtt jelenítik meg a félreértések elkerülése érdekében. A legnagyobb bevételt hozó öt film címét, együtt a megjelenési évükkel, a következő lekérdezéssel kaphatod meg:

select\_movies\_query = """

SELECT CONCAT(title, " (", release\_year, ")"),

collection\_in\_mil

FROM movies

ORDER BY collection\_in\_mil DESC

LIMIT 5

"""

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for movie in cursor.fetchall():

print(movie)

Ha nem szeretnéd használni a LIMIT clause-t, és nem kell az összes rekordot visszakapnod, akkor a cursor objektumnak van .fetchone() és .fetchmany() metódusa is:

* .fetchone() vagy a következő sor eredményét adja vissza egy tuple-ként, vagy None értéket, ha nincs több sor.
* .fetchmany() a következő rekordkészletet adja vissza egy tuple-k listájaként. Egy méret argumentumot használ, amely alapértelmezetten 1, és amellyel megadhatod, hogy hány rekordot kell visszakapnod. Ha nincs több sor, akkor a metódus egy üres listát ad vissza.

Próbáld ki ismét a legnagyobb bevételt hozó öt film címét, együtt a megjelenési évekkel, de ezúttal használd a .fetchmany() metódust:

select\_movies\_query = """

SELECT CONCAT(title, " (", release\_year, ")"),

collection\_in\_mil

FROM movies

ORDER BY collection\_in\_mil DESC

"""

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for movie in cursor.fetchmany(size=5):

print(movie)

cursor.fetchall()

A .fetchmany()-vel kapott kimenet hasonló az előző LIMIT clause-zal kapott eredményhez. Észrevehettél egy további cursor.fetchall() hívást a végén. Ez azért van ott, hogy töröld az összes olvasatlan eredményt, amelyet a .fetchmany() nem olvasott be.

Fontos, hogy töröld az összes olvasatlan eredményt, mielőtt más utasításokat hajtasz végre ugyanazon a kapcsolaton. Ellenkező esetben egy InternalError: Unread result found kivétel fog dobódni.

## Feladatok

Mire vannak az alábbi query-k?

Első:

select\_movies\_query = """

SELECT title, AVG(rating) as average\_rating

FROM ratings

INNER JOIN movies

ON movies.id = ratings.movie\_id

GROUP BY movie\_id

ORDER BY average\_rating DESC

LIMIT 5

"""

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for movie in cursor.fetchall():

print(movie)

Második:

select\_movies\_query = """

SELECT CONCAT(first\_name, " ", last\_name), COUNT(\*) as num

FROM reviewers

INNER JOIN ratings

ON reviewers.id = ratings.reviewer\_id

GROUP BY reviewer\_id

ORDER BY num DESC

LIMIT 1

"""

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for movie in cursor.fetchall():

print(movie)

# Frissítés és törlés

Ebben a részben az adatbázisban található rekordok módosítását és törlését tárgyaljuk. Mindkét műveletet elvégezheted egyetlen rekordon vagy több rekordon a táblában. A módosítandó sorokat a WHERE clause segítségével választod ki.

## Update

Az adatbázisban található egyik reviewer, Amy Farah Fowler, most már Sheldon Cooperrel házas, így a vezetékneve Cooperre változott. Az adatbázist ez alapján kell frissíteni. Az adatok frissítéséhez a MySQL az UPDATE utasítást használja:

update\_query = """

UPDATE

reviewers

SET

last\_name = "Cooper"

WHERE

first\_name = "Amy"

"""

with connection.cursor() as cursor:

cursor.execute(update\_query)

connection.commit()

A kód átadja a frissítési lekérdezést a cursor.execute() metódusnak, majd a .commit() metódus segítségével érvényesíti a szükséges változtatásokat a reviewers táblán.

Megjegyzés: Az UPDATE lekérdezésben a WHERE clause segít meghatározni, hogy mely rekordokat kell frissíteni. Ha nem használod a WHERE-t, akkor az összes rekord frissítésre kerül!

Tegyük fel, hogy egy olyan lehetőséget kell biztosítanod, amely lehetővé teszi a reviewerek számára a pontszámok módosítását. Egy reviewernek három értéket kell megadni: a film azonosítóját (movie\_id), a reviewer azonosítóját (reviewer\_id) és az új pontszámot. A kód megjeleníti a módosítás utáni rekordot.

Feltételezve, hogy movie\_id = 18, reviewer\_id = 15 és az új pontszám = 5.0, a következő MySQL lekérdezéseket használhatod a szükséges módosítások elvégzéséhez:

UPDATE ratings

SET rating = 5.0

WHERE movie\_id = 18 AND reviewer\_id = 15;

SELECT \* FROM ratings

WHERE movie\_id = 18 AND reviewer\_id = 15;

Az fenti lekérdezések először frissítik a pontszámot, majd megjelenítik azt. Létrehozhatsz egy teljes Python scriptet, amely kapcsolatot hoz létre az adatbázissal és lehetővé teszi a reviewer számára a pontszám módosítását:

from getpass import getpass

from mysql.connector import connect, Error

movie\_id = input("Enter movie id: ")

reviewer\_id = input("Enter reviewer id: ")

new\_rating = input("Enter new rating: ")

update\_query = """

UPDATE

ratings

SET

rating = "%s"

WHERE

movie\_id = "%s" AND reviewer\_id = "%s";

SELECT \*

FROM ratings

WHERE

movie\_id = "%s" AND reviewer\_id = "%s"

""" % (

new\_rating,

movie\_id,

reviewer\_id,

movie\_id,

reviewer\_id,

)

try:

with connect(

host="localhost",

user=input("Enter username: "),

password=getpass("Enter password: "),

database="online\_movie\_rating",

) as connection:

with connection.cursor() as cursor:

for result in cursor.execute(update\_query, multi=True):

if result.with\_rows:

print(result.fetchall())

connection.commit()

except Error as e:

print(e)

Mentsd el ezt a kódot egy "modify\_ratings.py" nevű fájlba. A fenti kód %s placeholdereket használ a kapott bemenet beillesztésére az update\_query stringbe. Ez az első több lekérdezést tartalmazó query string. Több lekérdezést szeretnél átadni egyetlen cursor.execute()-nak, ebben az esetben a metódus multi argumentumát True értékre kell állítanod.

Ha a multi értéke True, akkor a cursor.execute() egy iterátort ad vissza. Az iterátor elemei a lekérdezésben átadott utasítást végrehajtó cursor objektumokhoz tartoznak. A fenti kód egy for ciklust futtat ezen az iterátoron, majd mindegyik cursor objektumra .fetchall() hívást hajt végre.

Megjegyzés: Fontos, hogy minden cursor objektumon végrehajtsd a .fetchall() függvényt. Ha új utasítást szeretnél végrehajtani ugyanazzal a kapcsolattal, biztosítanod kell, hogy nincsenek olvasatlan eredmények a korábbi végrehajtásokból. Ha vannak olvasatlan eredmények, akkor kivételt fogsz kapni.

Ha egy művelet során nincs eredményhalmaz lekérdezve, akkor a .fetchall() kivételt fog generálni. Ezt a hibát elkerülendő, a fenti kódban a cursor.with\_rows tulajdonságot használjuk, amely jelzi, hogy a legutóbb végrehajtott művelet sorokat hozott létre.

### SQL injection

Bár ez a kód megoldást nyújthat a célodra, a WHERE clause a jelenlegi állapotában a webes hackerek fő célpontja. Ez sérülékeny a SQL injekciónak nevezett támadásra, amely lehetővé teszi a rosszindulatú szereplők számára az adatbázisod megrongálását vagy egyéb visszaéléseket.

Figyelem: Ne próbáld ki az alábbi bemeneteket az adatbázisodon! Ezek megrongálhatják a tábládat, és újra kell létrehoznod.

Mi lesz az elábbi futás eredménye?

$ python modify\_ratings.py

Enter movie id: 18

Enter reviewer id: 15

Enter new rating: 5.0

Enter username: <user\_name>

Enter password:

Mi lesz az alábbi futás eredménye?

$ python modify\_ratings.py

Enter movie id: 18

Enter reviewer id: 15"; UPDATE reviewers SET last\_name = "A

Enter new rating: 5.0

Enter username: <user\_name>

Enter password:

## Delete

A rekordok törlése nagyon hasonlóan működik, mint a rekordok frissítése. A DELETE utasítást használod a kiválasztott rekordok eltávolításához.

Megjegyzés: A törlés visszafordíthatatlan folyamat. Ha nem használod a WHERE clause-t, akkor az összes rekord törlődni fog a megadott táblából. A törölt rekordok visszaállításához újra le kell futtatnod az INSERT INTO query-ket.



Ajánlott, hogy először fuss egy SELECT lekérdezést ugyanazzal a szűrővel, hogy biztos legyél benne, hogy a megfelelő rekordokat törlöd. Például, ha törölni szeretnéd az összes értékelést, amit a reviewer\_id = 2 adott, először futtasd le a megfelelő SELECT lekérdezést:

select\_movies\_query = """

SELECT reviewer\_id, movie\_id FROM ratings

WHERE reviewer\_id = 2

"""

with connection.cursor() as cursor:

cursor.execute(select\_movies\_query)

for movie in cursor.fetchall():

print(movie)

A fenti kódrészlet kiírja a reviewer\_id és movie\_id értékeket a ratings táblából azoknál a rekordoknál, ahol reviewer\_id = 2. Miután megerősítetted, hogy ezek azok a rekordok, amelyeket törölni szeretnél, futtathatsz egy DELETE lekérdezést ugyanezzel a szűrővel:

delete\_query = "DELETE FROM ratings WHERE reviewer\_id = 2"

with connection.cursor() as cursor:

cursor.execute(delete\_query)

connection.commit()

Ezzel a lekérdezéssel eltávolítod az összes értékelést, amit a reviewer\_id = 2 értéket tartalmazó értékelő adott a ratings táblából.