Adatbázis létrehozása MySQL-ben:

```
CREATE DATABASE minta DEFAULT CHARACTER SET utf8 COLLATE utf8_hungarian_ci;
```

Adatbázis létrehozása csak akkor, ha még nem létezik az adott névvel adatbázis:

```
CREATE DATABASE IF NOT EXISTS users DEFAULT CHARACTER SET utf8 COLLATE utf8_hungarian_ci;
```

Tábla létrehozása MySQL-ben:

```
CREATE DATABASE minta DEFAULT CHARACTER SET utf8 COLLATE utf8_hungarian_ci;
USE minta;
CREATE TABLE customers (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) UNIQUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

A **USE** paranccsal használatba vettük az adatbázist.

Ha az **INT** típusnak nem adunk szűkítést pl. INT(3), akkor automatikusan INT(11) lesz.

Az **AUTO_INCREMENT** automatikusan növeli rekordonként az értékét

A **PRIMARY KEY** jelenti, hogy ez a mező lesz az elsődleges kulcs (minden sorban különbözni fog, egyértelműen azonosítja a rekordokat!) Ez maga után vonja, hogy nem lehet NULL a mező, tehát **NOT NULL** és egyedi az értéke, tehát **UNIQUE**. Valamint a MySQL belsőleg létrehoz egy megszorítást az elsődleges kulcshoz és ennek a neve mindig **PRIMARY**, tehát nem érdemes saját megszorításnév az elsődleges kulcshoz! Pl. ilyet **NEM ÉRDEMES**: *CONSTRAINT pk_users PRIMARY KEY (id)*

A ***name*** mező **VARCHAR(100)-**a azt jelenti, hogy stringet tárol, maximum 100 karakterben, de a memóriából csak annyi bájtot foglal, amennyi kell az aktuális értékének és nem amennyi a 100 karakternek kell! A **NOT NULL** miatt ez egy KÖTELEZŐEN MEGADANDÓAN érték!

Az ***email*** mezőben az **UNIQUE** azt jelenti, hogy EGYEDINEK kell lennie az értékének a többi e-mail értékhez képest, viszont nincs NOT NULL, így nem kötelező kitölteni, lehet az értéke NULL!

A ***created_at*** mező típusa IDŐBÉLYEG! Tárolja a DÁTUMOT, az IDŐT és az IDŐZÓNÁT is, csak ebben különbözik a DATETIME-tól!!! Ha nem visz be értéket feltöltéskor a jóember, akkor alapértelmezésben (**DEFAULT** miatt) a pillanatnyi időt és időzónát kapja meg!

Az **ENGINE=InnoDB** kifejezést a modern MySQL-ben alapértelmezésben megkapja, így elhagyható, azt határozza meg, hogy melyik adattároló motort **(storage engine)** használja a tábla. Egyelőre elég annyit tudni róla, hogy az InnoDB motor a legjobb!

```sql
CREATE TABLE orders (
    id INT AUTO_INCREMENT PRIMARY KEY,
    order_date DATE NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    customer_id INT,
    CONSTRAINT fk_orders_customers
        FOREIGN KEY (customer_id)
        REFERENCES customers(id)
        ON DELETE CASCADE
) ENGINE=InnoDB;
```

Itt csak az újdonságokra kitérve:

A **DATE** érték csak az ÉV-HÓNAP-NAP -ot tárolja!

A **DECIMAL(10,2)** MySQL-ben rögzített tizedesjegyű számokat jelent (**pontos érték**, nem lebegőpontos)! NEM KEREKÍT! Itt 10 jegy, ebből 2 tizedes. PÉNZÜGYI számításokhoz KÖTELEZŐ a használata!!!

A **CONSTRAINT fk_orders_customers** egy megszorítást jelent, aminek az a neve, hogy **fk_orders_customers** és a nevével utal arra, hogy a **IDEGEN KULCS** az **ORDERS** táblában

Felépítése:

- ebben a táblában az idegen kulcs ez a mező lesz: **FOREIGN KEY (customer_id)**
- a másik (kapcsolt) táblában erre a mezőre hivatkozik: **REFERENCES customers(id)**
- törléskor törli a másik táblából is a kapcsolt rekordokat: **ON DELETE CASCADE**

A törlés huncut. A következő lehetőségek jöhetnek szóba:

ON DELETE RESTRICT (*alapértelmezett, nem kell külön megadni, tehát ha nem adunk meg semmit, akkor ez az érvényes*!!!) - Nem engedi a törlést, ha hivatkoznak a rekordra.

ON DELETE CASCADE - A hivatkozó rekordok is automatikusan törlődnek.

ON DELETE SET NULL - A hivatkozó mező értéke NULL lesz.

Végezetül nézzük meg az adatok feltöltését (vedd észre, hogy nem bajlódunk az ID-vel):

```
INSERT INTO customers (name, email) VALUES
('Kiss Péter', 'peter.kiss@gmail.com'),
('Nagy Anna', 'anna.nagy@yahoo.com');
INSERT INTO orders (order_date, amount, customer_id) VALUES
('2025-10-10', 19999.90, 1),
('2025-10-12', 4999.00, 1),
('2025-10-12', 24999.00, 2);
```

Nézzük az **UPDATE** parancsot, ami a már létező adatok módosítására szolgál:

```
UPDATE tabla_neve
SET oszlop1 = új_érték1,
    oszlop2 = új_érték2,
    ...
WHERE feltétel;
```

Itt arra kell nagyon figyelni, hogy ha kimarad a WHERE feltétel, akkor minden adat módosul a felsorolt oszlopokban!!!

```
UPDATE customers
SET name = 'Fehér Dezső',
    email = 'dezso.feher@gmail.com'
WHERE name = 'Kiss Péter';
```

Számított mezőknél lehet matematika műveleteket is végrehajtani. Pl. növeljük meg az értéket 20 %-al a 100 000 kisebb értékek esetén:

```
UPDATE orders
SET amount = amount * 1.2
WHERE amount < 100000;
```

A **DELETE** parancs rekordok törlésére szolgál egy táblából.

```
DELETE FROM tabla_neve
WHERE feltétel;
```

Ha kihagyod a WHERE feltételt, akkor minden rekordot töröl!!!

Töröljük az 1-es ID-val rendelkező rekordot:

```
DELETE FROM customers
WHERE id = 1;
```

Töröljünk minden rekordot:

```
DELETE FROM customers;
```

Ebben az esetben a **TÁBLA SZERKEZETE** megmarad és az ***AUTO_INCREMENT NEM KEZDI ELŐRÖL*** a számolást, ha új rekordokat szúrunk a táblába!!!

Ha szeretnénk 1-től indítani az ID értékét, akkor következő parancsot kell használni:

```
TRUNCATE TABLE customers;
```

Ez gyorsabb (nem logol) a DELETE-nél és a számlálót is visszaállítja 1-re!

Törlés előtt érdemes TESZTELNI! Pl.:

```
SELECT * FROM customers WHERE name LIKE 'na';
```

Ha csak a törlendő rekord jelenik meg, akkor kiadhatjuk a törlés parancsot is:

```
DELETE FROM customers WHERE name LIKE 'na';
```

A MySQL alapértelmezés szerint **autocommit = ON** módban üzemel, ami azt jelenti:

minden parancs azonnal véglegesül, így a **DELETE NEM VISSZAVONHATÓ**!

A **TRUNCATE** huncut parancs, szintaxisa:

```
TRUNCATE TABLE tabla_neve;
```

Ez a parancs minden sort töröl a táblából, de nem egyenként (mint a DELETE), hanem egyszerűen újra létrehozza a táblát a háttérben azonos szerkezettel és visszaállítja az AUTO_INCREMENT számlálót 1-re, így sokkal gyorsabb, mint a **DELETE**. Viszont **NEM** lehet benne **WHERE**, **NEM VONHATÓ VISSZA** és nem hajtható végre, ha függő rekordok vannak!

A már létező tábla szerkezetének módosítása MySQL-ben az **ALTER TABLE** paranccsal történik.

Ezzel lehet a következőket végrehajtani:

- új oszlopokat hozzáadni,
- meglévőket módosítani,
- átnevezni,
- törölni,
- kulcsokat, indexeket vagy megszorításokat hozzáadni.

```
ALTER TABLE tabla_neve művelet;
```

A következő műveletek a leggyakoribbak: **ADD, MODIFY, CHANGE, DROP**

pl. két oszlopot adunk a meglévő táblánkhoz:

```
ALTER TABLE customers
ADD COLUMN phone VARCHAR(20),
ADD COLUMN postal_code VARCHAR(10);
```

Ilyenkor a már a táblában lévő rekordoknál az új mezőértékek NULL-ak lesznek!!!

Oszlop módosítása:

```
ALTER TABLE customers
MODIFY COLUMN phone VARCHAR(30);
```

Oszlop átnevezése (**CHANGE**) (közben módosíthatjuk a típust is!)

```
ALTER TABLE customers
CHANGE COLUMN phone mobile_number VARCHAR(40);
```

Oszlop törlése:

```
ALTER TABLE customers
DROP COLUMN postal_code;
```

Elsődlegs kulcs hozzáadása:

```
ALTER TABLE customers
ADD PRIMARY KEY (id);
```

Ha már volt elsődleges kulcs, akkor előbb törölni kell:

```
ALTER TABLE customers
DROP PRIMARY KEY;
```

Idegen kulcs hozzáadása:

```
ALTER TABLE orders
ADD CONSTRAINT fk_orders_customer
FOREIGN KEY (customer_id) REFERENCES customers(id)
ON DELETE CASCADE;
```

Index hozzáadása:

```
ALTER TABLE customers
ADD INDEX idx_name (name);
```

Egyedi index hozzáadása:

```
ALTER TABLE customers
ADD UNIQUE (email);
```

Tábla átnevezése:

```
ALTER TABLE customers
RENAME TO clients;
```