# Online Könyvesbolt Adatbázis Tervezése

## Bevezetés

Az online könyvesbolt adatbázis célja, hogy támogassa a könyvek, vásárlók, rendelések és számlák adatainak hatékony kezelését. Az adatbázis tervezése során különös figyelmet fordítunk a funkcionalitásra, a teljesítményre és a bővíthetőségre.

## Táblák Tervezése

### 1. Könyvek (books)

Ez a tábla tárolja az összes könyv adatait, beleértve a címet, szerzőt, árat, készletet és elérhetőséget.

Oszlopok:

* - id: SERIAL PRIMARY KEY
* - title: VARCHAR(255) NOT NULL
* - author: VARCHAR(255)
* - price: NUMERIC(10, 2) NOT NULL
* - stock: INT NOT NULL
* - is\_available: BOOLEAN DEFAULT TRUE

### 2. Vásárlók (customers)

Ez a tábla tárolja az online könyvesbolt regisztrált vásárlóinak adatait.

Oszlopok:

* - id: SERIAL PRIMARY KEY
* - name: VARCHAR(255) NOT NULL
* - email: VARCHAR(255) UNIQUE NOT NULL
* - address: TEXT
* - city: VARCHAR(100)

### 3. Rendelések (orders)

Ez a tábla tárolja a vásárlók által leadott rendeléseket, beleértve a rendelés állapotát és dátumát.

Oszlopok:

* - id: SERIAL PRIMARY KEY
* - customer\_id: INT REFERENCES customers(id)
* - order\_date: TIMESTAMP DEFAULT CURRENT\_TIMESTAMP
* - status: VARCHAR(50) DEFAULT 'Pending'

### 4. Rendelés tételek (order\_items)

Ez a tábla tartalmazza a rendeléseken belüli könyvek részleteit, például a mennyiséget.

Oszlopok:

* - order\_id: INT REFERENCES orders(id)
* - book\_id: INT REFERENCES books(id)
* - quantity: INT NOT NULL

### 5. Számlák (invoices)

Ez a tábla tárolja a rendeléshez kapcsolódó számlák adatait, beleértve az összeget és a számla dátumát.

Oszlopok:

* - id: SERIAL PRIMARY KEY
* - order\_id: INT REFERENCES orders(id)
* - total\_amount: NUMERIC(10, 2) NOT NULL
* - invoice\_date: TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

## Kapcsolatok

Az adatbázis táblái közötti kapcsolatok a következőképpen alakulnak:  
- A könyvek és rendelés tételek között idegen kulcs kapcsolat van.  
- A vásárlók és a rendelések között idegen kulcs kapcsolat van.  
- A rendelések és a számlák között idegen kulcs kapcsolat van.

## Indexek és Optimalizáció

Az indexek és optimalizációs megoldások célja a gyors lekérdezések és tranzakciók biztosítása:  
- Elsődleges kulcs indexek minden táblán.  
- Egyedi index az email mezőn a vásárlók táblában.  
- Kapcsolati indexek a rendelés tételeken és a számlákon.

Adatok beszúrása

A teszt adatokat ezzel a parancsal töltöttem be:

INSERT INTO books (title, author, price, stock) VALUES ('PostgreSQL alapok', 'Kiss János', 4500.00, 50), ('SQL Mesterfokon', 'Nagy Anna', 5200.00, 30);

A tömeges adatot (10000 rekordott):

COPY books(id, title, author, price, stock) FROM '/path/to/books.csv' DELIMITER ',' CSV HEADER; parancsal töltöttem meg.

### 4. ****Lekérdezések****

* **Könyvek elérhetősége**:

SELECT title, stock FROM books WHERE stock > 0;

* **Vásárlók vásárlási előzményei**:

SELECT c.name, o.id AS order\_id, o.order\_date

FROM customers c

JOIN orders o ON c.id = o.customer\_id

WHERE c.email = ' customer2@example.com';

* **Rendelések státusza**:

SELECT id, customer\_id, status FROM orders;

ezeket a query-ket használtam a tesztnél, ezeket mellékelem fájlként a feltöltésnél.

### 5.Trigger megvalósítása

**is availilable hozzáadása a books táblához:**

**ALTER TABLE books ADD COLUMN is\_available BOOLEAN DEFAULT TRUE;**

**frissités létrehozása:**

**CREATE OR REPLACE FUNCTION update\_book\_availability()**

**RETURNS TRIGGER AS $$**

**BEGIN**

**Ellenőrzés:**

**SET is\_available = CASE**

**WHEN stock - NEW.quantity <= 0 THEN FALSE**

**ELSE TRUE**

**END,**

**stock = stock - NEW.quantity**

**WHERE id = NEW.book\_id;**

**RETURN NEW;**

**END;**

**$$ LANGUAGE plpgsql;**

**Trigger létrehozása:**

**CREATE TRIGGER trg\_update\_book\_availability**

**AFTER INSERT OR UPDATE ON order\_items**

**FOR EACH ROW**

**WHEN (NEW.quantity IS NOT NULL) -- Csak akkor fut, ha van mennyiség**

**EXECUTE FUNCTION update\_book\_availability();**

### 6.Tárolt eljárás létrehozása

CREATE OR REPLACE FUNCTION generate\_invoice(order\_id INT)

RETURNS VOID AS $$

DECLARE

total\_amount NUMERIC(10, 2); -- A rendelés teljes összege

BEGIN

-- Számítsd ki a rendelés teljes összegét

SELECT SUM(oi.quantity \* b.price) INTO total\_amount

FROM order\_items oi

JOIN books b ON oi.book\_id = b.id

WHERE oi.order\_id = order\_id;

-- Hozz létre egy új számla bejegyzést

INSERT INTO invoices (order\_id, total\_amount, invoice\_date)

VALUES (order\_id, total\_amount, CURRENT\_TIMESTAMP);

-- Jelöld meg a rendelést teljesítettként

UPDATE orders

SET status = 'Completed'

WHERE id = order\_id;

RAISE NOTICE 'Invoice generated for order % with total amount %', order\_id, total\_amount;

END;

$$ LANGUAGE plpgsql;

**Legjobban fogyókönyv:**

CREATE OR REPLACE FUNCTION best\_selling\_books(limit\_count INT)

RETURNS TABLE(book\_id INT, title TEXT, total\_sales INT) AS $$

BEGIN

RETURN QUERY

SELECT

b.id AS book\_id,

b.title,

SUM(oi.quantity) AS total\_sales

FROM order\_items oi

JOIN books b ON oi.book\_id = b.id

GROUP BY b.id, b.title

ORDER BY total\_sales DESC

LIMIT limit\_count;

END;

$$ LANGUAGE plpgsql;

Legaktívabb vásárlók riportja:

**CREATE OR REPLACE FUNCTION most\_active\_customers(limit\_count INT)**

**RETURNS TABLE(customer\_id INT, name TEXT, email TEXT, total\_orders INT, total\_spent NUMERIC) AS $$**

**BEGIN**

**RETURN QUERY**

**SELECT**

**c.id AS customer\_id,**

**c.name,**

**c.email,**

**COUNT(o.id) AS total\_orders,**

**COALESCE(SUM(oi.quantity \* b.price), 0) AS total\_spent**

**FROM customers c**

**LEFT JOIN orders o ON c.id = o.customer\_id**

**LEFT JOIN order\_items oi ON o.id = oi.order\_id**

**LEFT JOIN books b ON oi.book\_id = b.id**

**GROUP BY c.id, c.name, c.email**

**ORDER BY total\_orders DESC**

**LIMIT limit\_count;**

**END;**

**$$ LANGUAGE plpgsql;**

### 7.10000 rekordos tesztelés

**SELECT**

**c.id AS customer\_id,**

**c.name AS customer\_name,**

**c.email AS customer\_email,**

**o.id AS order\_id,**

**o.order\_date,**

**b.id AS book\_id,**

**b.title AS book\_title,**

**oi.quantity,**

**(oi.quantity \* b.price) AS total\_price**

**FROM**

**customers c**

**JOIN orders o ON c.id = o.customer\_id**

**JOIN (**

**SELECT**

**customer\_id,**

**MAX(order\_date) AS latest\_order\_date**

**FROM orders**

**GROUP BY customer\_id**

**) latest\_orders ON c.id = latest\_orders.customer\_id AND o.order\_date = latest\_orders.latest\_order\_date**

**JOIN order\_items oi ON o.id = oi.order\_id**

**JOIN books b ON oi.book\_id = b.id**

**ORDER BY c.id, o.order\_date DESC;**

**ez az összetett sql lekérdezés 133 msec alatt futott le és 18903 sor volt használva.**

Az indexek használatának elemzése a következő eredményeket hozta:

- \*\*orders.customer\_id\*\*: Nincs index, ami lassítja a vásárlók rendeléseinek lekérdezését.  
- \*\*order\_items.order\_id\*\*: Index jelen van, és hatékonyan gyorsítja a csatlakozásokat.  
- \*\*customers.city\*\*: Nincs index, ami lassú szűrést eredményez város alapján.

Az adatbázis táblák méretének ellenőrzése alapján az alábbi adatok állnak rendelkezésre:

- \*\*customers\*\*: 10 MB  
- \*\*orders\*\*: 20 MB  
- \*\*order\_items\*\*: 50 MB  
- \*\*books\*\*: 5 MB  
- \*\*invoices\*\*: 2 MB

**Teljesitmény alapján nem rossz ennyi adattal, de lehet még indexeléssel vagy particiónálással gyorsitani ezt, például ha gyakran szűrsz városokra, ha kiváncsi vagy hol érdemes nyitni egy boltot a vásárlások alapján akkor érdemes indexelni a indexelni a customers tábla city értékét vagy gyorsabba cim szűrésre a könyvek cimét indexelni stb. A lekérdezések finom hangolása is jó megoldás lehet gyorsabb lekérdezések eléréséhez.**