



# Programozási technológia

Adatbáziskezelés (JDBC)

Dr. Szendrei Rudolf  
ELTE Informatikai Kar  
2020.



# Témák

- JDBC
- Adatbázisok
- JDBC használatának lépései
- Tranzakciók
- Adatbázis tervezés – EK diagram

# A JDBC...

- Java API szabvány relációs adatbázisok használatához
- Java SE része
- Felhasználása
  1. Kapcsolódás egy adatbázishoz
  2. Lekérdezések küldése az adatbázis felé
  3. Kapott eredmények feldolgozása

# Főbb komponensei

## ➤ A JDBC API

Az adatbázishoz való hozzáférést kezeli, SQL utasítások küldése az adatbázis felé, eredmények feldolgozása, az adatbázis módosítása. `java.sql`, `javax.sql`

## ➤ JDBC Driver Manager

Kapcsolódás a használandó JDBC driver-hez

# Relációs adatbázis áttekintés

- Az adatbázis egy adatok tárolására alkalmas rendszer, azzal a céllal, hogy az adatok könnyen lekérdezhetők és feldolgozhatók legyenek.
- Relációs adatbázis esetén az adatok táblákba, sorokba, oszlopokba rendezetten jelennek meg. A táblák adatai kapcsolódhatnak egymáshoz.
- Integritási szabályok:

- A táblák sorai egyediek
- Az adatok nem ismétlődnek, megfelelő táblában tároltak

```
TABLE: STUDENTS
```

```
| Name | Age | Pet | Pet Name |  
-----  
| Heather | 10 | Dog | Rex |
```

- NULL koncepció: NULL != ,üres', NULL != 0..... És NULL != NULL

# SQL

## ➤ DQL (Data Query Language)

```
SELECT First_Name, Last_Name  
FROM Employees  
WHERE Last_Name LIKE 'Tibi%' and Car_Number IS NULL
```

## ➤ DML (Data Manipulation Language)

```
INSERT INTO table (col_names...) VALUES(values...)  
UPDATE table SET col_name=... WHERE ...  
DELETE FROM table WHERE...
```

## ➤ DDL (Data Definition Language)

```
CREATE DATABASE db_name;  
CREATE TABLE table_name(col_name col_type,...);  
// DROP DATABASE..., DROP TABLE..., ALTER TABLE... stb..
```

# ResultSet, Cursor

- A lekérdezés eredményeként kapott sorhalmaz a ResultSet.
- A ResultSet elemeit soronként érhetjük el egyesével.
- Ezt a Cursor segítségével tehetjük meg, amely egy iterátorként viselkedik.
- A JDBC API kurzora képes a ResultSeten mindkét irányba mozogni

# Tranzakciók, Lockok

- Szükségessé válnak, amikor több felhasználó szeretne ugyanazon adatokkal dolgozni egy időben. Pl.: egy felhasználó sorokat módosít egy táblában, miközben egy másik felhasználó ugyanazt a táblát lekérdezi. Lehetséges, hogy a 2. felhasználó részben elavult adatokat kap eredményül.
- A tranzakciók SQL utasítások egy csoportja, amely egy logikai egységet képez.
- A tranzakciók eredménye commit; vagy rollback;
- Table Lock: megakadályozza a tábla eldobását, ha nem commitolt tranzakció tartozik a táblához.
- Row Lock: megakadályozza, hogy több tranzakció ugyanazt a sort módosítsa



# JDBC használatának lépései

1. Adatbázis-kezelő telepítése
2. Adatbázis létrehozása
3. Adatbázis-csatoló könyvtár hozzáadása a projekthez
4. Adatbázis specifikus JDBC driver betöltése
5. Kapcsolat létrehozása az adatbázissal
6. SQL utasítás objektum létrehozása
7. SQL utasítás végrehajtása
8. Eredmények feldolgozása
9. Az SQL utasítás és a kapcsolat lezárása

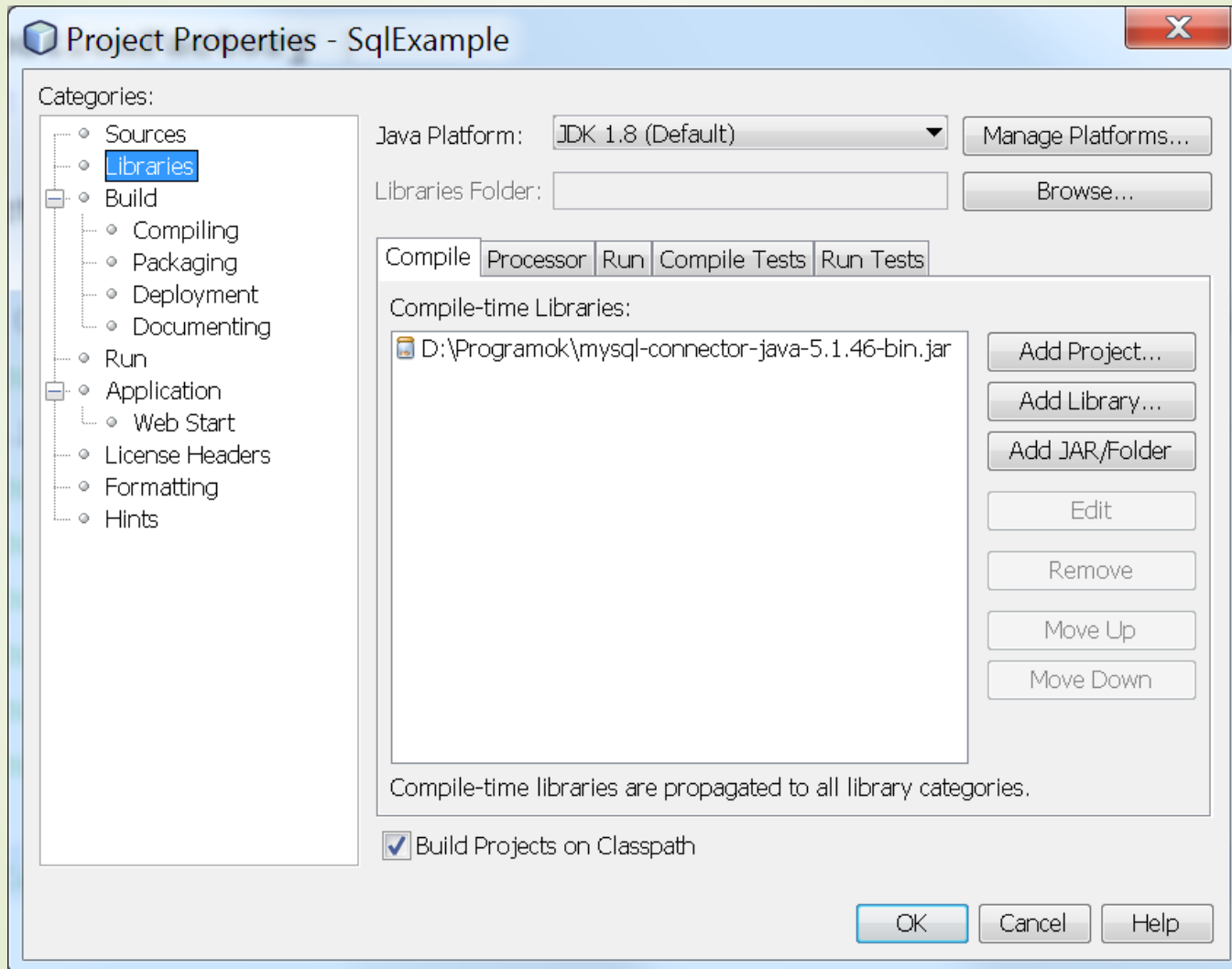
# Adatbázis-kezelő telepítése

- A labor gépeken a MySQL már telepítve, melyeken a hozzáféréshez a következő adatok szükségesek:
  - Felhasználó név: tanulo
  - Jelszó: asd123
  - Port: 3306 (alapértelmezett a telepítésnél)
- MySQL telepítése saját gépre
  - Letöltés: <https://dev.mysql.com/downloads/installer/>
  - Tartalmazza a következőket:
    - MySQL szerver
    - MySQL csatolók a programozási nyelvekhez (Connector-J) (a telepítési mappa lib könyvtárában található)
    - MySQL Workbench
- Adatbázis létrehozása MySQL Workbenchben vagy NetBeans-ben

# Adatbázis létrehozása NetBeans-ben

- Services fül → Drivers → jobb katt → New driver... → Add
  - MySQL könyvtárában: mysql-connector-java-[VERZIÓ]-bin.jar
- Drivers → MySQL (Connector/J driver) → jobb katt → Connect Using...
  - Adjuk meg a szükséges adatokat (a Database-t hagyjuk üresen).
  - Helyes beállítások esetén a Test Connection gomb lenyomása után megjelenik, hogy „Connection Succeeded”.
- A kapcsolat megjelenik, és kinyitva láthatjuk a már létező táblákat. Kattintsunk jobb gombbal rá, majd Execute Command Itt végrehajthatunk tetszőleges SQL utasításokat
  - CREATE DATABASE [ADATBÁZISNÉV];
  - Parancsok futtatása: Ctrl+Shift+E

# Adatbázis-csatoló hozzáadása a projekthez



# Adatbázis driver betöltése

```
try {  
    // This loads an instance of the MySQL Driver.  
    // The driver has to be in the classpath.  
    Class.forName("com.mysql.jdbc.Driver");  
} catch (ClassNotFoundException cnfe) {  
    System.out.println("" + cnfe);  
}
```

# Kapcsolódás adatbázishoz - DriverManager

- A megadott adatbázis URL felhasználásával kapcsolódik az adatbázishoz
- Format
  - jdbc:[subprotocol]:[server-location]/[database-name]
  - Pl.: jdbc:mysql://localhost:3306/sample
- Az adatbázis kapcsolat elkérése:  
`DriverManager.getConnection`
- Connection objektum szálak közötti megosztása nem ajánlott, létrehozásuk és a rajtuk végzett műveletek költségesek.

# Driver betöltése és kapcsolódás Connection Pool technikával az adatbázishoz

```
public class ConnectionFactory {
private static MysqlConnectionPoolDataSource conn;
    private ConnectionFactory() {}
    public static Connection getConnection() throws
        ClassNotFoundException, SQLException {
        if (conn == null){
            Class.forName("com.mysql.jdbc.Driver"); // driver betöltése
            conn = new MysqlConnectionPoolDataSource();
            conn.setServerName("localhost");
            conn.setPort(3306);
            conn.setDatabaseName("sokoban");
            conn.setUser("tanulo");
            conn.setPassword("asd123");
        }
        return conn.getPooledConnection().getConnection();
    }
}
```



# Connection részletesebben

- ▶ `java.sql.Connection` csomagban található
- ▶ `Statement createStatement() throws SQLException`
  - ▶ Statementekkel adhatóak meg SQL utasítások
- ▶ `void close() throws SQLException`
  - ▶ Connection manuális lezárása
  - ▶ Lezárható a try-with-resources koncepcióval is
- ▶ `void setAutoCommit(boolean b) throws SQLException`
  - ▶ külön tranzakció legyen-e minden utasítás
- ▶ `void commit() throws SQLException`
- ▶ `void rollback() throws SQLException`



# SQLExceptions

- Az adatbázissal való munka közben fellépett hibák esetén kapjuk.
- Kinyerhető Információk:
  - A hiba leírása
  - SQL hibakód
  - Driver implementáció specifikus hibakód amely megegyezhet az adatbázis hibakódjával
- Warningok
  - `SQLWarning` az `SQLException` leszármazottja, a kevésbé kritikus hibákról informál
  - `Connection`, `Statement` és a `ResultSet` objektum esetén `getWarnings`

# Lekérdezések végrehajtása

- Az adatbázisban módosítást végrehajtani, vagy abból lekérdezni a következőképpen tudunk
- Connection segítségével létrehozunk egy utasítást

## Statement

- `Statement stmt = connection.createStatement();`
- Általános célú, statikus SQL lekérdezések végrehajtására.
- Nem fogad paramétereket

## PreparedStatement

- `PreparedStatement stmt = connection.prepareStatement();`
- A Statement-ből származik
- Dinamikus, többször, különböző paraméterekkel futtatandó lekérdezések végrehajtására
- Megadhatók input paraméterek

- Az utasításon végrehajtjuk az SQL parancsot

# Lekérdezések végrehajtása

## ➤ CallableStatement

- Tárolt eljárások hívására, a PreparedStatement-ből származik.

## ➤ Végrehajtás:

- `boolean execute (String SQL)`: Ha a visszatérési érték `true`, elérhető a `ResultSet`. DDL utasítások végrehajtására
- `int executeUpdate (String SQL)`: Az érintett sorok számával tér vissza. INSERT, UPDATE, és DELETE utasításokhoz.
- `ResultSet executeQuery (String SQL)`: Lekérdezések végrehajtásához.

# Eredmény feldolgozása – ResultSet

- Az lekérdezés eredményének sorait `ResultSet` típusú objektumban kapjuk meg.
- Kezdetben az iterátor az első sor előtt áll, amely a `next()` metódus hívással mozdul az első sorra.

```
while (rs.next()) {  
    String value1 = rs.getString(1);  
    int value2    = rs.getInt(2);  
    int value3    = rs.getInt("ADDR_LN1");  
}
```

- A `ResultSet` – nek megfelelő `getXXX()` metódusa van minden `java.sql.Types` típushoz.
- A sor mezőinek indexelése 1-el kezdődik!

# ResultSet

- Az adatok olvasásán kívül azok manipulálásra is tartalmaz műveleteket.
- ResultSet típusok
  - TYPE\_FORWARD\_ONLY: Nem scrollozható, a kurzor csak előre mozoghat az első elem előttől az utolsóig.
  - TYPE\_SCROLL\_INSENSITIVE: Scrollozható, a kurzor előre és hátra is mozgatható, abszolút pozícióra ugorhatunk. Bejárás közben az adatokon végzett módosítások nem láthatóak.
  - TYPE\_SCROLL\_SENSITIVE: Az adaton végzett változások láthatóak (amíg a result set nyitva van)
- Az alapértelmezett ResultSet nem update-elhető és csak előre tud lépni.

# További hasznos JDBC ismeretek

Adatbázis frissítése a lekérdezés eredményén keresztül

```
try (Statement stmt =
    con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                        ResultSet.CONCUR_UPDATABLE)) {
    String query = "SELECT * FROM coffees";
    ResultSet uprs = stmt.executeQuery(query);
    while (uprs.next()) {
        float f = uprs.getFloat("PRICE");
        uprs.updateFloat("PRICE", f * percentage);
        uprs.updateRow();
    }
}
```

# További hasznos JDBC ismeretek

## Adatbázis utasítások kötegetelt feldolgozása

- ▶ A Statement objektumokhoz tartozik egy (kezdetben üres) végrehajtási lista, amelyhez az addBatch metódussal adhatunk hozzá DML utasításokat (kiüríteni a clearBatch-el lehet).
- ▶ Az executeBatch metódus egyetlen végrehajtási egységként küldi el az utasításokat az adatbázisnak.
- ▶ A helyes hibakezeléshez, az auto commit-ot ki kell kapcsolni.

```
String stmt = "INSERT INTO COFFEES VALUES (?, ?) ";
PreparedStatement pstmt = con.prepareStatement(stmt);
pstmt.setString(1, "asd");
pstmt.setInt(2, 49);
pstmt.addBatch();

...

pstmt.executeBatch();
```



# További hasznos JDBC ismeretek

Insert után az automatikusan generált értékek elérésének engedélyezése

```
PreparedStatement stmt = conn.prepareStatement(  
    getInsertSql(),  
    Statement.RETURN_GENERATED_KEYS);  
try (ResultSet generatedKeys = stmt.getGeneratedKeys()) {  
    if (generatedKeys.next()) {  
        t.setId(generatedKeys.getInt(1));  
    } else {  
        throw new SQLException(  
            "Creating entity failed, no ID obtained.");  
    }  
}
```



# Tranzakció példa

- Feladat: Kölcsönzések kezelése
  - Egy kölcsönzéshez több kölcsönzött dolog tartozhat
  - A dolgok megadott kezdeti példányszámban állnak rendelkezésre
- Probléma: két kliens ugyanazt a dolgot szeretné kölcsön venni, de csak egy példány van bent. (megszorítás:  $\text{bent\_levo\_db} \geq 0$ )

- Táblák: Kölcsönzés(id, dátum, user)

Kölcsönzés\_elem(kölcsönzés\_id, elem\_id)

Elem(elem\_id, bent\_levo\_db)

Kliens 1

1. Kikölcsönzi ,A' dolgot → sikerül
2. Kikölcsönzi ,B' dolgot → nincs elég
3. Kikölcsönzi ,C' dolgot → ??

Kliens 2.

1. Kikölcsönzi ,B' dolgot → sikerül
2. Ementi a kölcsönzés adatait

Kölcsönzés lépései:

1. Új sor a kölcsönzés táblába
2. Minden elemre:
  1. Kölcsönzés\_elem felvétele
  2. Elem bent\_levo\_db csökkentése



# Megoldás

```
Auto commit off
```

```
Try {  
    Kikölcsönzi ,A' dolgot  
    Kikölcsönzi ,B' dolgot  
    Kikölcsönzi ,C' dolgot  
    commit;  
} catch (SQLException e) {  
    rollback;  
}
```

# Tranzakciós izolációs szintek

1. TRANSACTION\_READ\_UNCOMMITTED

2. TRANSACTION\_READ\_COMMITTED

Prevents: Dirty Reads

3. TRANSACTION\_REPEATABLE\_READ

Prevents: Dirty Reads, Non-Repeatable Reads

4. TRANSACTION\_SERIALIZABLE

Prevents: Dirty Reads, Non-Repeatable Reads, Phantom Reads

# Tranzakció izolációs problémák

## ➤ Dirty Reads:

- Nem véglegesített adatok olvasása  
Pl.: nem commitált update
- Lehetséges, hogy a változtatás visszavonásra kerül az olvasás során

## ➤ Non-Repeatable

- Akkor történik, amikor egy (A) tranzakció beolvas egy sort, amelyet (B) tranzakció időközben módosít. (A) másodszor is kiolvassa a sort, de különböző értéket lát.

## ➤ Phantom Reads

- Az (A) tranzakció beolvas egy sorhalmazt, a (B) tranzakció beilleszt egy új sort, (A) másodszor is kiolvassa a sorokat, de különböző számú sort kap ugyanarra a lekérdezésre.

# Adatbázis tervezés

- Egy jó adatbázis:
  - Nem tartalmaz redundanciát
  - Biztosítja az adatok épségét és pontosságát
- Tervezéskor a feladat az alkalmazás által használandó adatok táblákba rendezése, és kapcsolataik definiálása.
- Objektumelvű adatbázisoknál a relációs adatbázisra gyakran gondolunk úgy, mint amiben az osztályainkat táblaként, az objektumokat pedig az osztályhoz tartozó tábla soraiként tároljuk. Az objektumok közötti kapcsolatot kulcsokkal valósítjuk meg.

# Egyedkapcsolat diagram

- Az adatbázis logikai modellje egyedkapcsolat diagrammal írható le.
- A diagram elemei:
  - Entitások
  - Attribútumok
  - Kulcsok
  - Kapcsolatok
    - Egy-egy, egy-sok, sok-sok

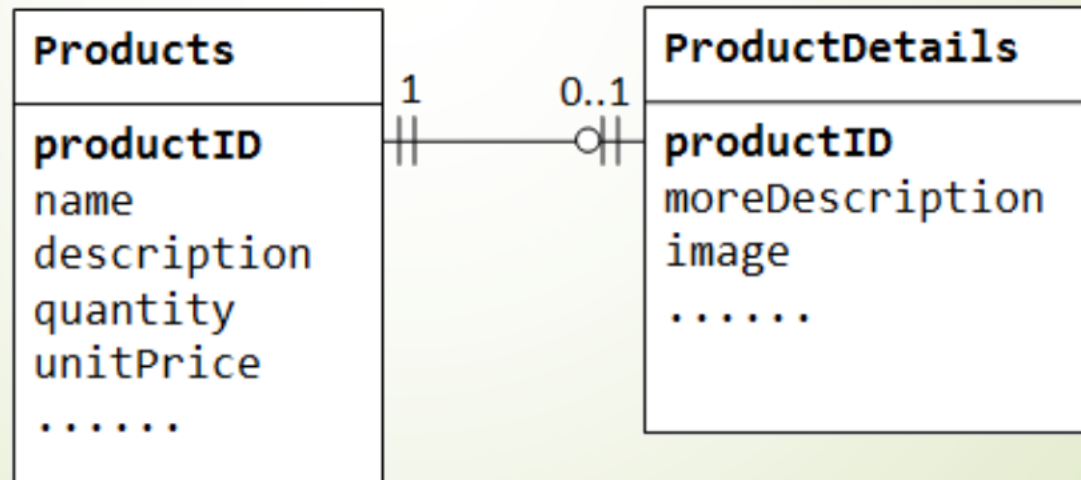
# Példa

- Hogyan reprezentálható:  
Egy tanár taníthat nulla vagy több osztályt, de egy osztályt pontosan egy tanár tanít.
- **1. lehetőség:** Induljunk ki a tanár táblából, ahol tároljuk az adatain túl a tanított osztályokat: osztály1, osztály2, ...  
**Probléma:** szükséges oszlopok száma?
- **2. lehetőség:** Induljunk ki az osztály táblából, adjuk hozzá a tanár adatait tartalmazó oszlopokat  
**Probléma:** a tanár adatai többször szerepelnek



# Egy-egy kapcsolat

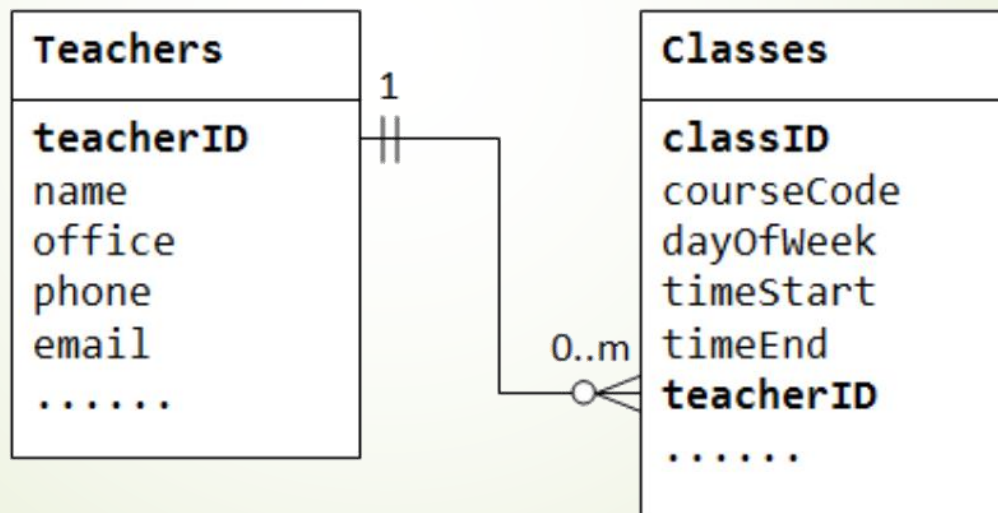
- Egy termék adatbázisban a termékeknek lehetnek kiegészítő, opcionális adatai.
- A termékekkel egy táblában tárolva számos üres mezőt eredményeznének ezen adatok helyei.





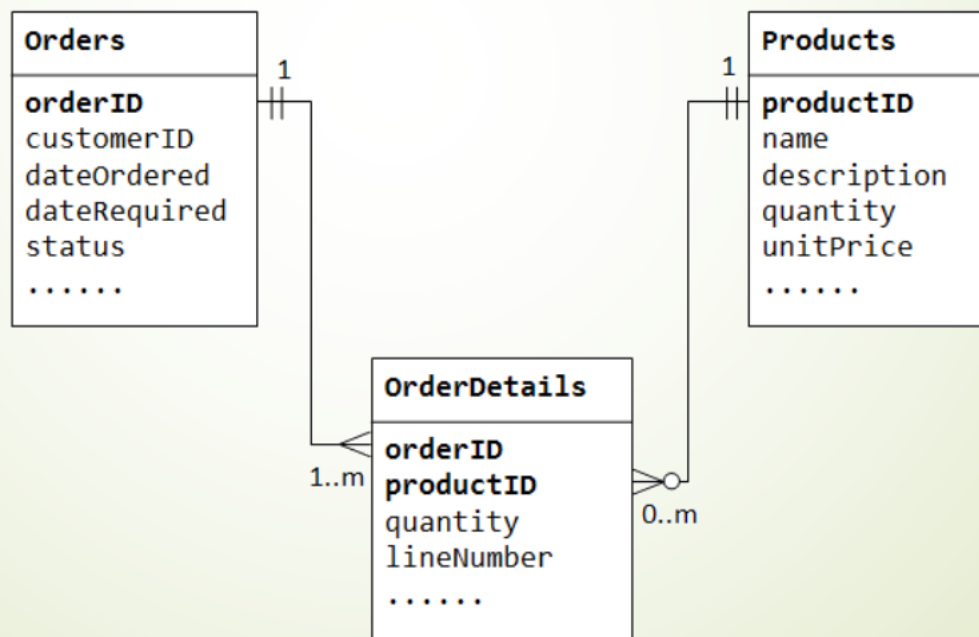
# Egy-sok kapcsolat

- Egy tanár taníthat nulla vagy több osztályt, de osztályt pontosan egy tanár tanít.



# Sok-sok kapcsolat

- Egy megrendeléshez egy vagy több termék tartozik,
- egy termék több megrendelésben is szerepelhet.

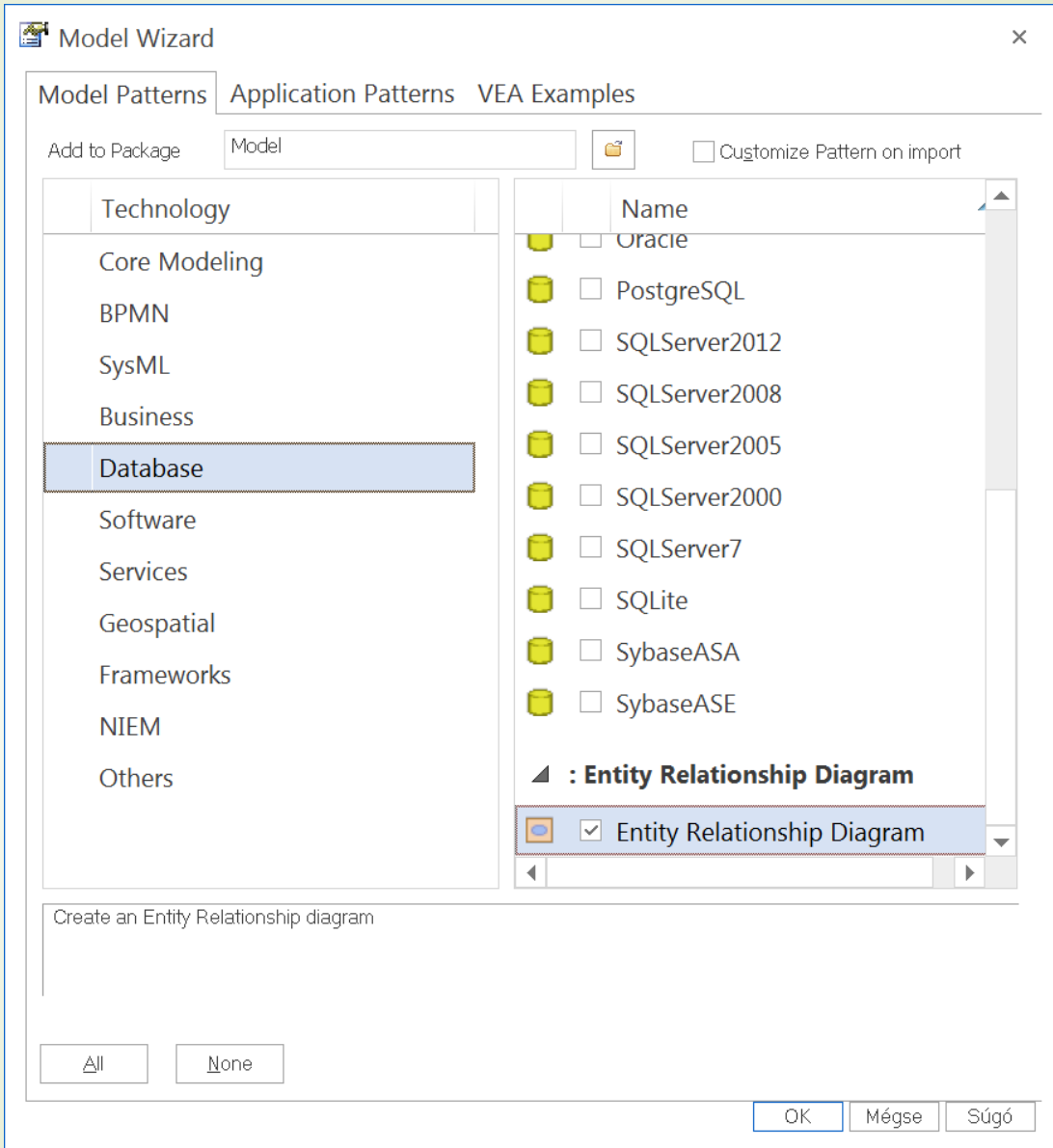


# EK diagram lépései

- **Entitások azonosítása**  
(Az alkalmazásban használt „dolgok”: megrendelés, termék, stb.)
- **Kapcsolatok definiálása**  
(Entitások közötti logikai kapcsolat)
- **Számosság**  
(Kapcsolatokban hány entitás vehet részt? Egy-egy, sok-egy.. Kötelező?)
- **Elsődleges kulcsok megadása**  
(Az adott jellemzők, amelyek egyértelműen azonosítanak egy entitást)
- **Sok-Sok kapcsolatok átírása sok-egy kapcsolattá**  
(Kapcsolótáblák)
- **Attribútumok definiálása és entitáshoz rendelése**  
(Entitások egyéb tulajdonságai, melyik entitáshoz tartozzon?)

# Egyedkapcsolat diagram EA-ben

- Nyissuk meg a diagram varázslót
  - Új projekt esetén a varázsló automatikusan megjelenik
  - Meglévő projektnél a Project Browserben jobb klikk a Model-en → Add → Add a Model Using Wizard, vagy
  - Gyorsbillentyű: Ctrl+Shift+M
- A Technology oldalon válasszuk ki a Database-t, a jobb oldalon pedig az Entity Relation Diagram-ot



# Egyedkapcsolat diagram EA-ben

- A Toolbox segítségével adjuk hozzá a diagramhoz az entitásokat (Entity), valamint az attribútumokat (Attribute).
- Vegyük fel a kapcsolatokat a megfelelő entitások közé (Relationship)
- Definiáljuk a kapcsolat tulajdonságait (dupla katt)
  - General fül: adjuk meg az irányt (Direction)
  - Role fül: multiplicitás megadása (Multiplicity / Multiplicity)

# Toolbox

▼ 🔍 ✕

More tools...

## ERD

- Entity
- Attribute
- N-ary Association

## ERD - Relationships

- Connector
- Relationship
- Disjoint
- Overlapping

## Common

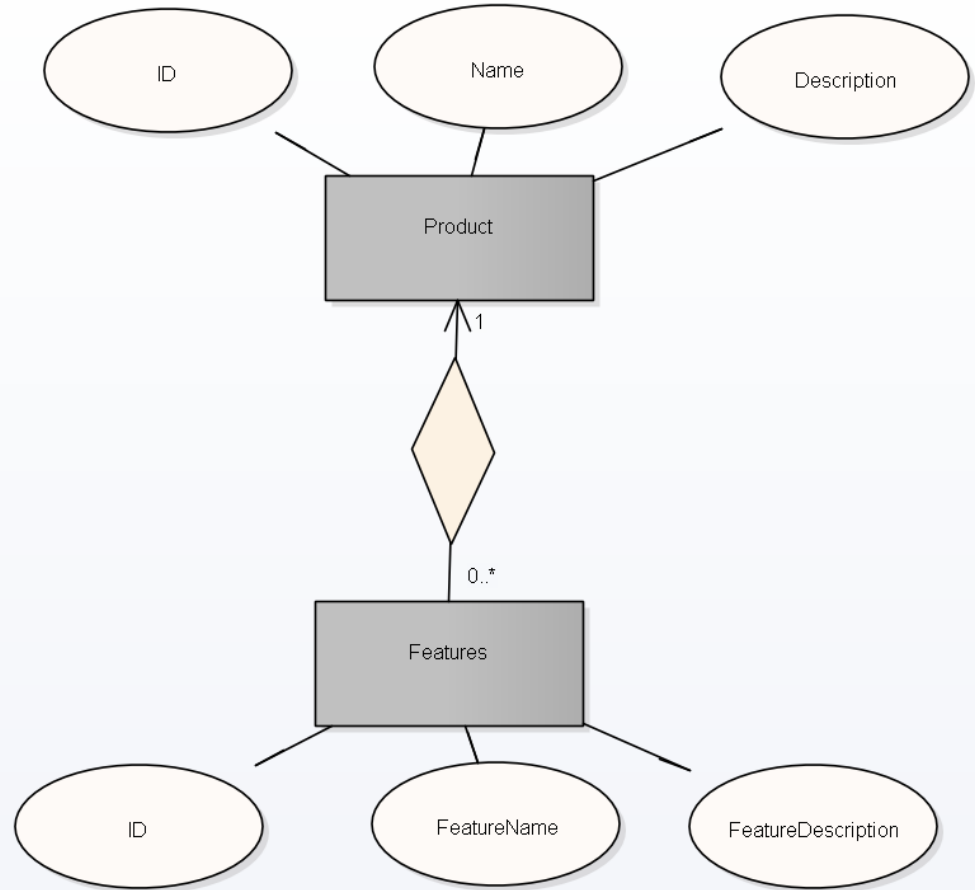


## Artifacts

- Artifact
- Document
- Encrypted Document
- Checklist
- Audited Checklist

Entity Relationship Diagram: "Entity Relationship Diagram"

\*Entity Relationship Diagram ✕



# A diagramhoz tartozó SQL

```
CREATE TABLE Product (  
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    Name VARCHAR(100) NOT NULL,  
    Description TEXT NOT NULL  
);  
  
CREATE TABLE Features (  
    ID INT NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    Name VARCHAR(100) NOT NULL,  
    Description TEXT NOT NULL,  
    FK_ProductID INT NOT NULL REFERENCES Product(ID)  
);
```