**lokálsd változó scope –**

**formális paraméter-aktuális paraméter –**

**visszatérési érték –**

**final**

**private-public szűkebb bővebb**

**short, Short**

KÉREK MINDENKIT, HOGY A VIZSGÁN ÚJ REPO-BA DOLGOZZATOK! Előkészített pom.xml-t kaptok. Projekt, könyvtár, repository neve legyen: zarovizsga . GroupId: hu.nive.ujratervezes, artifactId: zarovizsga. Csomagnév: hu.nive.ujratervezes.zarovizsga.

**Új projekt létrehozása + GitHub push**

1.IDEA -> File - > New -> Project

2. pom.xml update + jobb fent Maven klikk

3. új package, új osztály, új teszt

4. VCS -> Enable Version Control Integration

5. .idea könyvtárra jobb klikk -> Git - > Add to .gitignore

target  
.idea  
\*.iml

6. Git -> Github - > Share Project on Github

**pom.xml**

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>org.example</groupId>  
 <artifactId>training-solutions</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <configuration>  
 <source>14</source>  
 <target>14</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>15</maven.compiler.source>  
 <maven.compiler.target>15</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.mariadb.jdbc</groupId>  
 <artifactId>mariadb-java-client</artifactId>  
 <version>2.7.2</version>  
 </dependency>  
 <dependency>  
 <groupId>org.flywaydb</groupId>  
 <artifactId>flyway-core</artifactId>  
 <version>7.5.3</version>  
 </dependency>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter-api</artifactId>  
 <version>5.7.1</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
*</project>*

**jar plugin (main metódus hol van):**

<build>

<plugins>

<plugin>  
 <artifactId>maven-jar-plugin</artifactId>  
 <configuration>  
 <archive>  
 <manifest>  
 <mainClass>CustomerMain</mainClass>  
 </manifest>  
 </archive>  
 </configuration>  
</plugin>

</plugins>

</build>

**Parancssoros**

**build**

mvn clean package

**jar indítás**

>java -classpath target\speedrent-1.0-SNAPSHOT.jar CustomerMain

**jar indítás úgy, hogy pom.xmlbe be van állítva a main:**

java -jar target\distjar-1.0-SNAPSHOT.jar;

**Java projekt ráállítása github repora (local repo – remote repo összekapcsolás):**

git init

git remote add origin repourl

git add . , git commit -m „comment”, git push, git fetch

**Main**

public static void main(String[] args) {

}

**Értékek beolvasása**

Scanner scanner = new Scanner(System.in);

System.out.println("What's your name?");

String name = scanner.nextLine();

System.out.println("What's your year of birth?");

int yearOfBirth = scanner.nextInt();

Vigyázz a nextInt() metódussal, mert az nem olvassa be a sorvége jelet, és a következő nextLine() nem fog működni.

int age = scanner.nextInt();

scanner.nextLine(); //Ez fogja az ottmaradt sorvége jelet beolvasni

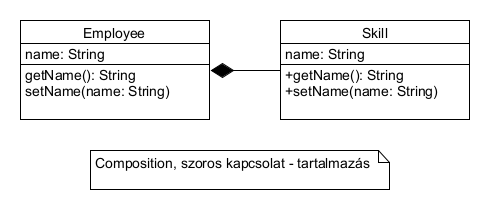
String name = scanner.nextLine();

Helyette jobb lehet a következő (NumberFormatException kivételt dob, ha nem szám):

String line = scanner.nextLine();

int age = Integer.parseInt(line); // Exception-t

**UML**

[](https://github.com/Training360/strukturavalto-java-public/blob/master/cheet-sheet/uml-demo.png)

**Számok**

boolean even = i % 2 == 0; // Páros?

double quotient = 10 / 4; // 2.0 - egészosztás

**Típuskényszerítés, autoboxing, String műveletek**

double d = 12; // int -> double implicit

int i = (int) 3.14; // double -> int kényszerítéssel, értéke 3

Integer i = 12; // int -> Integer autoboxing

int j = i; // Integer -> int autoboxing

int i = Integer.parseInt("12"); // Parse

double d = Double.parseDouble("12.1") // Parse

String si = Integer.toString(12); // toString

String di = Double.toString(d); // toString

**Stringműveletek**

boolean eq = name.equals(anotherName); // Egyenlőségvizsgálat

String fruit = "Apple";

int position = fruit.indexOf("pp"); // 1 (ha nem találja, -1)

int length = fruit.length() // 5 (hossza)

String sub = fruit.substring(1, 4); // ppl (1. indexű bekerül, 4. indexű nem kerül bele)

String upper = fruit.toUpperCase(); // APPLE

boolean empty = fruit.isEmpty(); // false, üres string esetén true

boolean blank = fruit.isBlank(); // false, ha csak whitespace karaktereket tartalmaz, akkor true

boolean startsWith = fruit.startsWith("app"); // true

boolean endWith = fruit.endsWith("le"); // true

boolean containsDoubleP = fruit.contains("pp"); // true

String part = fruit.substring(0, 2).toLowerCase(); // ap - összefűzhetők

Daraboljunk!

String s = "John;1980";

String[] parts = s.split(";"); // ; a separator karakter

String name = parts[0];

int yearOfBirth = Integer.parseInt(parts[1]);

Járjuk be karakterenként!

String s = "abcd";

for (char c: s.toCharArray()) {

System.out.println(Character.isLetter(c));

}

Hosszú string összefűzése:

StringBuilder sb = new StringBuilder();

for (int i = 0; i < 10; i++) {

sb.append(i).append(" \* ").append(i).append(" = ").append(i \* i);

}

Formázás:

String formatted = String.format("Name: %s, year of birth: %d", employee.getName(), employee.getYearOfBirth());

// Name: John Doe, year of birth: 1980

// Vagy azonnal kiírva:

System.out.printf("Name: %s, year of birth: %d", employee.getName(), employee.getYearOfBirth());

**Dátumok**

LocalDate fromDate = LocalDate.of(2015, Month.JANUARY, 30);

LocalDateTime fromDateTime = LocalDateTime.of(2015, 1, 20, 10, 15);

**DateFormatter**

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy.MM.dd");

LocalDate dateOfOrders = LocalDate.parse(line, formatter);

**Ciklusok**

Klasszikus for ciklus. Csak akkor használjuk ezt, ha szükségünk van a ciklusváltozóra.

for (int i = 0; i < 10; i++) {

System.out.println(i);

}

Un. for-each ciklus, használjuk mindig ezt, ha lehet!

List<String> names = List.of("John Doe", "Jack Doe");

for (String name: names) {

System.out.println(name);

}

**Tömbök**

Lehetőleg kerüljük, helyette használjunk listát! Néhány helyen elkerülhetetlen, pl. varargs, split() vagy ha a feladat így kéri.

int[] numbers = new int[10];

int[] names = new String[]{"John", "Jack"};

int length = names.length;

for (String name: names) {

System.out.println(name);

}

**Listák**

List<String> names = List.of("John", "Jack"); // Módosíthatatlan lista

List<String> moreNumbers = new ArrayList<>(); // Módosítható lista, diamond operátor, primitív típus nem lehet

List<String> copy = new ArrayList<>(numbers); // Módosítható másolat

copy.add("Jane"); // ["John", "Jack", "Jane"] - hozzáadás

copy.remove("John"); // ["Jack", "Jane"] - eltávolítás

boolean containsJohn = names.contains("John"); // false - nincs benne

int size = names.size(); // 2 - hossza

int indexOfJane = names.indexOf("Jane") // 1 - 1. indexen

for(String name: names){

System.out.println(name);

}

**Véletlenszám**

Random rnd = new Random();

int randomNumberTo10 = rnd.nextInt(10); // 0 - 9-ig generálhat számokat, 10-et sosem generál

**Felsorolásos típus**

public enum Coin {

TWOHUNDRED, HUNDRED, TWENTY, TEN, FIVE

}

**JUnit**

**Függőségek**

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>5.7.1</version>

<scope>test</scope>

</dependency>

**Teszt osztály**

public class TestCalculator {

void testAdd() {

// Given

Calculator calculator = new Calculator();

// When

int result = calculator.add(2, 3);

// Then

assertEquals(5, result);

}

}

**Kivételkezelés**

try {

int result = a / b;

// Ha nincs hiba

System.out.println("A hányados alsó egészrésze: " + result);

} catch (ArithmeticException ae) {

// Ha van hiba

throw new IllegalArgumentException("Arithmetic error", ae);

}

finally {

// Minden esetben lefut

System.out.println("Finally");

}

**Összegzés tétele**

Listában szereplő számok összege

public int sum(List<Integer> numbers) {

int sum = 0;

for (Integer n: numbers) {

sum += n;

}

return sum;

}

**Számlálás tétele**

A stringben szereplő b betűk száma:

public int countLetterB(String s) {

int count = 0;

for (char c: s.toCharArray()) {

if (c == 'b') {

count++;

}

}

return count;

}

**Szélsőérték keresés tétele**

Listában szereplő legnagyobb szám:

public int max(List<Integer> numbers) {

int max = Integer.MIN\_VALUE;

for (Integer n: numbers) {

if (n > max) {

max = n;

}

}

return max;

}

**Eldöntés tétele**

Csak 100-nál nagyobb számokat tartalmaz?

public boolean containsGreaterThanHundred(List<Integer> numbers) {

for (Integer i : numbers) {

if (i > 100) {

return true;

}

}

return false;

}

**Szűrés**

Csak a száznál nagyobb számok listáját adja vissza:

public List<Integer> greaterThanHundred(List<Integer> numbers) {

List<Integer> filtered = new ArrayList<>();

for (Integer i : numbers) {

if (i > 100) {

filtered.add(i);

}

}

return filtered;

}

**Transzformáció**

Csak a neveket adja vissza:

public List<String> getNames(List<Employee> employees) {

List<String> names = new ArrayList<>();

for (Employee employee: employees) {

names.add(employee.getName());

}

return names;

}

**Szöveges fájl beolvasása soronként**

public class FileReader {

public List<String> readLines(BufferedReader reader) throws IOException {

String line;

while ((line = reader.readLine()) != null) {

System.out.println(line);

}

}

public static void main(String[] args) {

try (BufferedReader reader = Files.newBufferedReader(Path.of("data.csv"))) {

new FileReader().readLines(reader);

} catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

}

}

Ugyanez classpathról:

try (BufferedReader reader = new BufferedReader(new InputStreamReader(FileReader.class.getResourceAsStream("data.csv")))) {

new FileReader().readLines(reader);

} catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

**Szöveges állomány kiírása soronként**

public class FileWriter {

public void writeLines(List<String> employees, BufferedWriter writer) {

try {

for (String employee : employees) {

writer.write(employee.getName() + "," + employee.getYearOfBirth());

}

} catch (IOException ioe) {

throw new IllegalStateException("Can not read file", ioe);

}

}

public static void main(String[] args) {

try (BufferedWriter writer = Files.newBufferedWriter(Path.of("data.csv"))) {

new FileWriter().writeLines(writer);

} catch (IOException ioe) {

throw new IllegalStateException("Can not write file", ioe);

}

}

}

**Set**

Set<String> names = new HashSet<>();

names.add("John");

names.add("Jack");

names.add("John");

System.out.println(names); // [John, Jack]

names.remove("John");

boolean contains = names.contains("Jack"); // true

for (String name: names) {

System.out.println(name);

}

**Map**

Map<String, String> words = new HashMap<>();

words.put("apple", "alma");

words.put("peer", "körte");

System.out.println(words.get("apple")); // alma

words.put("apple", "Apfel"); // Felülírja az értéket

System.out.println(words.get("apple")); // Apfel

Set<String> keys = words.keySet(); // Kulcsok

Collection<String> values = words.values(); // Értékek

for (Map.Entry entry: words.entrySet()) {

System.out.println(entry.getKey() + " - " + entry.getValue());

}

**Számlálás Map-pel**

Számoljuk meg, melyik betűből mennyi van egy Stringben!

String s = "alma mater";

Map<Character, Integer> countOfLetters = new HashMap<>();

for (char c: s.toCharArray()) {

if (countOfLetters.containsKey(c)) {

countOfLetters.put(c, countOfLetters.get(c) + 1);

}

else {

countOfLetters.put(c, 1);

}

}

System.out.println(countOfLetters); // { =1, a=3, r=1, t=1, e=1, l=1, m=2}

int numberOfM = countOfLetters.get('m'); // 2

**Iterátor**

List<Integer> values = new ArrayList<>(Arrays.asList(1, 2, 3, 4, 5));

for (Iterator<Integer> i = values.iterator(); i.hasNext()) {

int value = i.next();

if (value % 2 == 0) {

i.remove();

}

}

**Lista rendezése**

List<Integer> numbers = new ArrayList<>(List.of(6, 5, 8, 3));

Collections.sort(numbers);

// Figyelembe veszi az ékezeteket

List<String> names = new ArrayList<>(List.of("Benjámin", "Áron", "József", "Arnold"));

Collections.sort(names, Collator.getInstance(new Locale("hu", "HU")));

List<Employee> employees = List.of(

new Employee("John Doe", 1980),

new Employee("Jack Doe", 1970));

// Év vagy név szeretnénk rendezni

Collections.sort(employees, new Comparator<Employee>() {

@Override

public int compare(Employee o1, Employee o2) {

// return o1.getYearOfBirth() - o2.getYearOfBirth();

return o1.getName().compareTo(o2.getName());

}

});

**Copy konstruktor deep**

**public** Auction(Auction anotherAuction) {

user = new User(anotherAuction.user);

start = anotherAuction.start;

price = anotherAuction.price;

}

public User(User anotherUser) {  
 this.name = anotherUser.name;  
}

**JDBC**

**Függőségek**

<dependency>

<groupId>org.mariadb.jdbc</groupId>

<artifactId>mariadb-java-client</artifactId>

<version>2.7.2</version>

</dependency>

<dependency>

<groupId>org.flywaydb</groupId>

<artifactId>flyway-core</artifactId>

<version>7.5.3</version>

</dependency>

**DataSource létrehozása**

MariaDbDataSource dataSource;

try {

dataSource = new MariaDbDataSource();

dataSource.setUrl("jdbc:mariadb://localhost:3306/employees?useUnicode=true");

dataSource.setUser("employees");

dataSource.setPassword("employees");

}

catch (SQLException se) {

throw new IllegalStateException("Can not create data source", se);

}

**Flyway**

Flyway flyway = Flyway.configure().dataSource(dataSource).load();

flyway.clean();

flyway.migrate();

**Paraméterezett insert, update, delete**

try (

Connection conn = dataSource.getConnection();

PreparedStatement stmt =

conn.prepareStatement("insert into employees(emp\_name) values (?)")) {

stmt.setString(1, name);

stmt.executeUpdate();

}

catch (SQLException se) {

throw new IllegalStateException("Cannot insert", se);

}

**Lekérdezés**

public List<String> listEmployeeNames() {

try (

Connection conn = dataSource.getConnection();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("select emp\_name from employees order by emp\_name")

) {

List<String> names = new ArrayList<>();

while (rs.next()) {

String name = rs.getString("emp\_name");

names.add(name);

}

return names;

}

catch (SQLException se) {

throw new IllegalStateException("Cannot select employees", se);

}

}

**Paraméterezett lekérdezés**

try (

Connection conn = dataSource.getConnection();

PreparedStatement stmt =

conn.prepareStatement("select emp\_name from employees where id = ?");

) {

stmt.setLong(1, id);

// ...

} catch (SQLException sqle) {

throw new IllegalArgumentException("Error by insert", sqle);

}

try (

ResultSet rs = stmt.executeQuery();

) {

if (rs.next()) {

String name = rs.getString("emp\_name");

return name;

}

throw new IllegalArgumentException("No result");

} catch (SQLException sqle) {

throw new IllegalArgumentException("Error by insert", sqle);

}

**Generált azonosító lekérdezése**

try (Connection conn = dataSource.getConnection();

PreparedStatement stmt = conn.prepareStatement("insert into employees(emp\_name) values (?)",

Statement.RETURN\_GENERATED\_KEYS)

) {

stmt.setString(1, name);

stmt.executeUpdate();

return executeAndGetGeneratedKey(stmt);

} catch (SQLException sqle) {

throw new IllegalArgumentException("Error by insert", sqle);

}

private long executeAndGetGeneratedKey(PreparedStatement stmt) {

try (

ResultSet rs = stmt.getGeneratedKeys();

) {

if (rs.next()) {

return rs.getLong(1);

} else {

throw new SQLException("No key has generated");

}

} catch (SQLException sqle) {

throw new IllegalArgumentException("Error by insert", sqle);

}

}

**SQL lekérdezések**

**Adatbázis- és tábla létrehozás**

CREATE DATABASE `hambi`;

USE CUSTOMERS;

## CREATE TABLE `products` (

## `id` int,

## `name varchar` (255),

## `description` varchar(255),

## `price` int

## );

**SELECT**

## SELECT \* FROM table\_name;

## SELECT name, city FROM customers;

## SELECT DISTINCT country FROM customers;

## SELECT DISTINCT country AS ’Ország’ FROM customers;

**WHERE - Selectek feltételek alapján**

## SELECT \* FROM customers WHERE country = ’Hungary’;

**WHERE operátorai**

## <>(nem egyenlő), < és >, between 2 and 4, in(2,4,5)

**WHERE feltételek összekacsolása**

## SELECT \* FROM customers WHERE country = ’Hungary’ AND city = ’Budapest’;

## SELECT \* FROM customers WHERE country = ’Hungary’ OR city = ’Budapest’;

## SELECT \* FROM customers WHERE NOT country = ’Germany’;

## SELECT \* FROM customers WHERE country = ’Hungary’ AND city = Berlin OR city = Budapest’;

## SELECT \* FROM customers WHERE country = ’Hungary’ AND (city = Berlin; OR city = Budapest’);

**ORDER BY - Találatok sorba rendezése**

## SELECT \* FROM customers ORDER BY country;

## SELECT \* FROM customers ORDER BY country, name;

## SELECT \* FROM customers ORDER BY country DESC, name;

## SELECT \* FROM customers ORDER BY country ASC, name DESC;

**LIMIT – csak az első x rekord!**

…

## ORDER BY.. DESC LIMIT 10;

**MIN, MAX COUNT, AVG, SUM**

SELECT MIN(price) AS SmallestPrice FROM products;

SELECT MAX(price) AS LargestPrice FROM products;

SELECT COUNT(id)FROM products WHERE category\_Id = 2;

SELECT COUNT(\*) FROM `orszagok` WHERE `terulet` > 50000 AND `terulet` < 150000;

SELECT AVG(price) FROM products;

SELECT SUM(price) FROM products WHERE category\_id=1;

**GROUP BY - Találatok csoportosítása (A GROUP BY mindig az ORDER BY előtt van!)**

SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;

**LIKE - Keresés hasonlóság alapján - ’\_egy karakter ÉS ’%’ bármi**

SELECT \* FROM customers WHERE name LIKE ’a%’;

SELECT \* FROM Customers WHERE CustomerName LIKE '\_r%';

**Beágyazott lekérdezések – selectben select**

SELECT `orszag` FROM `orszagok`

WHERE `terulet` < (SELECT `terulet` FROM `orszagok` WHERE `orszag` = 'MAGYARORSZÁG');

SELECT `orszag`, `terulet` FROM `orszagok` WHERE `terulet` = (SELECT MAX(`terulet`) FROM `orszagok`);

SELECT `orszag`, `terulet` FROM `orszagok` WHERE `terulet` = (SELECT MIN(`terulet`) FROM `orszagok`);

**INSERT – Új rekordok beszúrása az adattáblákba**

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

INSERT INTO Customers VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway');

INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),

('King', 'Koen Jensen', 'Oslo 21', 'Oslo', '7000', 'Norway'),

('Cool', 'Viking Kirksen', 'Berlin 44', 'Bergen', '2001', 'Norway');

**ADD – meglévő tábla oszloppal való bővítése**

ALTER TABLE `befiz` ADD `ugyfel\_azon` INT(10) UNSIGNED NOT NULL;

**UPDATE - Az egyes rekordok frissítése; legyen mindig WHERE!**

UPDATE Customers SET ContactName = 'Alfred Schmidt', City = 'Frankfurt' WHERE CustomerID = 1;

**DELETE - Rekordok törlése; legyen mindig WHERE!**

DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste'

**JOIN - Táblák összekötése - egyszerre több táblából is szeretnénk lekérni adatot, egy lekérdezéssel** (inner/left/right, full outer join)

SELECT `orders`.id, `customers`.name, `orders`.order\_date FROM `orders` INNER JOIN `customers` ON `orders`.customer\_id = `customers`.id;

**Join megvalósítása összerendelő tábla segítségével több a többhöz kapcsolat esetén**

SELECT \* FROM products p JOIN product\_to\_category ptc ON ptc.product\_id = p.Productid JOIN categories c ON c.id = ptc.category\_id ORDER by c.id DESC;

SELECT citizens.citizen\_name, citizens.age, cities.city FROM citizens JOIN cities ON citizens.zip=cities.zip WHERE citizens.age >100;

**Egy tábla csatolása saját magához – egyezések keresése vagy párokba rendezés**

SELECT A.name AS CustomerName1, B.name AS CustomerName2, A.city FROM customers A, customers B WHERE A.id <> B.id AND A.city = B.city;

**Tábla létrehozása megszorításokkal**

CREATE TABLE `log` (

id INT NOT NULL AUTO\_INCREMENT,

message VARCHAR(500) NOT NULL,

product\_id INT NOT NULL,

PRIMARY KEY (id),

FOREIGN KEY (product\_id) REFERENCES products (id)

);

CREATE TABLE Persons (

ID int NOT NULL UNIQUE,

ID email NOT NULL UNIQUE,

LastName varchar(255) NOT NULL,

FirstName varchar(255),

Age int

);

## *ON DELETE CASCADE*

## töri a hozzátartozókat is (rekordhoz tartozó összes másik rekordot az idegenek táblákból a foreign kulcs mentén)

## *ON UPTADE CASCADE*

## végigviszi a változtatást az összes kacsolódó táblában is

## *SET DEFAULT*

## *megmarad az adat, ráállítjuk egy default azonosítóra*

## *ON DELETE SET NUL*

## ami foreign keyből jön, arra törléskor (pl tábla drop) NULL-t állít be

**Megszorítások hozzáadása**

ALTER TABLE customers ADD CONSTRAINT unique\_name\_pairs UNIQUE (`name`, `contact`);

ALTER TABLE `customers` MODIFY COLUMN `id` int NOT NULL AUTO\_INCREMENT;

**Beszúrás utáni autoincrement helyreállítás:**

ALTER TABLE `customers` AUTO\_INCREMENT = 6;

**Index hozzárendelése (gyorsabb kereséspl. név és kontakt alapján, de plusz memória)**

CREATE INDEX name\_contact ON customers (NAME, contact);

**UNION - kettő vagy több SELECT utasítás eredménykészletét kombinálja** (a MariadBD nem ismeri a full outer joint, a helyett is jó az union)

SELECT City, Country FROM Customers WHERE Country='Germany' UNION

SELECT City, Country FROM Suppliers WHERE Country='Germany' ORDER BY City;

SELECT 'Customer' As Type, ContactName, City, Country

FROM Customers UNION SELECT 'Supplier', ContactName, City, Country FROM Suppliers;

SELECT \* FROM `pizza` WHERE `pizza\_id` < 2000 UNION

SELECT \* FROM `pizza` WHERE `pizza\_name` LIKE '%m%';

**HAVING - Csoportfeltételek létrehozása**(ha már csoportosítottam az oszlopokat group by-al, utána már alapba nem tudok szűrőt megadni; ezt lehet megoldani HAVING-el)

SELECT Sz\_Helye AS 'Születési Hely', SUM(fiz) AS 'Össz fizetés', AVG(fiz) AS 'Átlag fizetés'

FROM employees GROUP BY Sz\_Helye HAVING AVG(fiz) >= 120000;