

Verarbeitung von Null-Werten



- In Datenbanken kann man bei der Spaltendefinition angeben, dass NULL-Werte erlaubt sind. Das bedeutet, dass in einem bestimmten Datensatz "kein Wert" für dieses Attribut gesetzt ist.
- Im Beispiel rechts heißt das, dass für die Stadt Iraklion der Längen- und Breitengrad unbekannt ist - das entsprechende Attribut enthält keinen Wert.
- Man muss NULL hierbei von 0 oder einem leeren String unterscheiden. 0 wären gültige Koordinaten, ein leerer String wäre immer noch ein String, NULL heißt jedoch der Wert ist nicht eingetragen, es liegt keine Information vor.

Verarbeitung mit Java

Bei der Iteration über ein ResultSet werden die Attributwerte mit entsprechenden get-Methoden in passenden Java Variablen gespeichert:

```
Statement statement = DBConnection.createStatement();
ResultSet result = statement.executeQuery("SELECT * FROM schueler");
while(result.next()) {
    Integer klasse = result.getInt("SKlasse");           // get-Methode für
Integer
    String vorname = result.getString("SVorname");      // get-Methode für String
}
```

Wenn eine Spalte der Datenbank einen NULL-Wert enthält, liefert die get-Methode von Java dort den Default-Wert des Datentyps zurück. Für Integer-Werte ist das 0, für Strings null (das "Java" null).

Man kann bei Integer Variablen also nicht mehr unterscheiden, ob das Datenbankfeld tatsächlich den Wert 0 enthalten hat oder in der Datenbank keine Daten vorhanden waren, also ein NULL-Wert ausgelesen wurde, der dann zu einer 0 in der Java Variablen wurde.

Um NULL von 0 unterscheiden zu können, gibt es in der Klasse ResultSet die Methode `wasNull()`: boolean. Sie gibt an, ob der zuletzt gelesene Wert ein NULL-Wert war oder nicht:

name	population	longitude	latitude
Name der Stadt	Einwohnerzahl	Längengrad	Breitengrad
Thessaloniki	406413	23	41
Iraklion	102398	NULL	NULL
—	..		

```
ResultSet rs = stm.executeQuery(...);  
rs.next(); // Bearbeite "Thessaloniki"
```

```
int temp = rs.getInt("longitude"); // → 23  
boolean b1 = rs.wasNull(); // → false
```

```
temp = rs.getInt(2); // → 406413  
boolean b2 = rs.wasNull(); // → false
```

```
rs.next(); // Nächster Datensatz
```

```
temp = rs.getInt("longitude"); // → 0  
boolean b3 = r.wasNull(); // → true
```

Man könnte jetzt also so etwas tun, um das Problem zu umgehen, und den NULL_wert aus der Datenbank in die Java-Variable zu überführen:

```
[...]  
int wert = rs.getInt(3);           // → 0, aber in der DB NULL!  
boolean wasnull = rs.wasNull();   // → true  
if(wasnull) {  
    wert = null;  
}  
[...]
```

Allerdings klappt das bei den primitiven Datentypen wie `int`, `char`, `double` u.ä. nicht, da diese keine Objekte sind und darum nicht `null` sein können.

Aus diesem Grund muss man bei den primitiven Datentypen die entsprechenden [Wrapper-Klassen](#) verwenden - also `Integer`, `Character`, `Double` und Co.

Eine **korrekte Variante** sieht also so aus:

```
[...]  
Integer wert = rs.getInt(3);       // → 0, aber in der DB NULL!
```

```
if(rs.isNull()) {  
    wert = null;  
}  
[...]
```

Warum ist das wichtig?

Wenn man die Werte die in einer Datenbank gespeichert sind ausgeben oder verändern möchte, ist es wichtig, zu unterscheiden, ob ein Information nicht vorhanden war oder obn der Wert 0 in der Datenbank gespeichert war. wenn man beispielsweise die Textfelder einer GUI mit werten aus eine Datenbank befüllen möchte, könnte das so aussehen:

```
ResultSet rs = stm.executeQuery("...");  
rs.next();  
  
Integer x = rs.getInt(1);  
if (rs.isNull()) x = null;  
  
[...]  
  
if (x == null) {  
    textBox1.setText("Kein Wert in der DB vorhanden");  
} else {  
    textBox1.setText(x.toString());  
}
```

From:
<https://www.info-bw.de/> -

Permanent link:
https://www.info-bw.de/faecher:informatik:oberstufe:datenbanken:projekt:java_db:java_db_null:start?rev=1743614154

Last update: **02.04.2025 17:15**

