Schweizerische
Fachschule
TEKO

Semester Arbeit

Klassifizierung	VERTRAULICH
Status	in Arbeit
Projektname	News_Scraper_FLZ
Dok Nummer	SEA-001- NeSc
Dokumenten	Semesterarbeit Neus_Scraper
Projektleiter	Florian Zingg
Version	1.0
Datum	25. Juni 2025
Auftraggeber	Teko Bern
Autor/Autoren	Florian Zingg

Änderungsverzeichnis

Version	Datum	Änderung	Autor
0.1	30.06.2025	Dokument Erstellen	Florian Zingg
0.2	10.07.2025	Dokument review	Florian Zingg
0.3	12.08.2025	Dokument Überarbeitung	Florian Zingg
1.0	14.08.2025	Dokument Freigabe	Florian Zingg

Tabelle 1:Änderungskontrolle

Zweck der Untersuchung

Die Semesterarbeit hatte zum Ziel, am Beispiel des Projekts *News Scraper FLZ* den Prozess von der Analyse einer bestehenden Datenbank bis zur Umsetzung eines optimierten Modells durchzuführen. Dabei sollten Datenbankkonzepte wie Normalisierung, Tabellenbeziehungen und Abfrageoptimierung praxisnah angewendet werden. Die Arbeit verband Programmierung und Datenbankdesign, um ein praxisorientiertes Lernergebnis zu erzielen.

Erkenntnis:

Im Verlauf der Arbeit wurde deutlich, wie wichtig ein durchdachtes Datenbankdesign für die Stabilität und Erweiterbarkeit eines Projekts ist. Die Übung bot die Möglichkeit, praxisnah mit SQL, Tabellenbeziehungen und Normalisierung zu arbeiten und so ein tieferes Verständnis für den Nutzen einer sauberen Datenstruktur zu entwickeln. Wir konnten nachvollziehen, wie Änderungen an externen Datenquellen direkte Auswirkungen auf die Datenbankanbindung und den Anwendungscode haben. Zudem zeigte sich, dass ein flexibles Modell die Integration neuer Funktionen – wie etwa den Export in verschiedene Formate oder die Erweiterung um weitere Quellen – erheblich erleichtert. Insgesamt vermittelte die Arbeit wertvolle Erfahrungen im Zusammenspiel von Softwareentwicklung und Datenbanktechnik, die in künftigen Projekten von Nutzen sein werden.

Inhalt

1 Modulararbeit – Datenbankdesignprozess News Scraper

1. Einleitung

Ziel dieser Arbeit ist es, den **Datenbankdesignprozess** praxisnah nachzuvollziehen und umzusetzen. Als Beispiel dient das Projekt **„News Scraper FLZ“**, eine Python-Anwendung mit PyQt6-Oberfläche, die Nachrichtenartikel von Online-Quellen automatisch abruft, anzeigt und in einer SQLite-Datenbank speichert. Der Schwerpunkt liegt auf der Analyse der bestehenden Datenbankstruktur (**IST-Zustand**), deren Optimierung durch **Normalisierung** sowie der Umsetzung eines **SOLL-Modells**
mit verbesserter Erweiterbarkeit und Redundanzvermeidung.

Das Endergebnis umfasst:

- eine funktionierende SQLite-Datenbank
- ein vollständiges SQL-Schema
- Beispiel-Datensätze
- Testabfragen mit SQL-JOINs
- Crow’s-Foot-Diagramme für IST- und SOLL-Struktur
- eine begleitende Dokumentation und Präsentation

Vorgehensweise in der Arbeit:

1. Analyse der bestehenden Tabellenstruktur
2. Identifikation von Schwachstellen
3. Entwicklung eines normalisierten SOLL-Datenmodells
4. Implementierung des SQL-Schemas
5. Einfügen von Beispieldaten
6. Überprüfung mit Testabfragen
7. Auswertung und Fazit

2 IST-Datenmodell

Im aktuellen Stand besteht die Datenbank aus **nur einer Tabelle** articles.
Diese speichert sämtliche Informationen zu den gescrapten Artikeln in einer einzigen Struktur.Crow’s-Foot-Diagramm (IST):
→ Hier Bild erd_news_scraper_IST.png einfügen

2.1 Tabellenbeschreibung (articles)

Spalte	Typ	Beschreibung
id	INTEGER	Eindeutige Artikel-ID (Primärschlüssel)
title	TEXT	Titel des Artikels
url	TEXT	Webadresse zum Artikel (muss eindeutig sein)
date	TEXT	Veröffentlichungsdatum
snippet	TEXT	Kurzer Auszug/Inhalt
image_url	TEXT	Link zum Vorschaubild
source	TEXT	Name der Nachrichtenquelle
scraped_at	TEXT	Zeitstempel des Scraping-Vorgangs

Tabelle 2 Tabellenbeschreibung

2.2 Schwachstellen des IST-Modells

- **Redundanz:** Quellenname wird bei jedem Artikel erneut gespeichert.
- **Keine Trennung von Entitäten:** Autoren, Schlagwörter oder Quellen werden nicht in eigenen Tabellen verwaltet.
- **Eingeschränkte Erweiterbarkeit:** Neue Quellen oder Metadaten erfordern Änderungen direkt in der Artikeltabelle.

- **Fehlende m:n-Beziehungen:** Schlagwörter (Tags) können nicht sauber mehreren Artikeln zugeordnet werden.

3 Datenmodell

Um Redundanzen zu vermeiden und die Erweiterbarkeit der Datenbank sicherzustellen, wurde das bestehende Datenmodell normalisiert. Dabei erfolgte eine Trennung der Daten in mehrere logisch getrennte Tabellen. Ziel ist es, eine klare Struktur zu schaffen, die flexibel auf zukünftige Anforderungen reagieren kann und gleichzeitig eine bessere Datenintegrität gewährleistet.

3.1 Änderungen gegenüber IST

- Quellen (sources) in eigene Tabelle ausgelagert
- Autoren (authors) in eigene Tabelle ausgelagert (optional)
- Tags (tags) für Schlagwörter eingeführt
- Zwischentabelle (article_tag) für m:n-Beziehung zwischen Artikeln und Schlagwörtern

Tabelle	Beschreibung
sources	Speichert Informationen zu Nachrichtenquellen, inkl. Name und Basis-URL.
authors	Optional: Enthält Autorennamen und Profil-Links.
tags	Speichert Schlagwörter zur Kategorisierung von Artikeln.
articles	Speichert Artikelinformationen mit Verweisen auf Quelle, Autor und Schlagwörter.
article_tag	Zwischentabelle, verknüpft Artikel mit Schlagwörtern (m:n-Beziehung).

Tabelle 3 Tabellenbeschreibung

3.2 Crow's Foot Diagramm

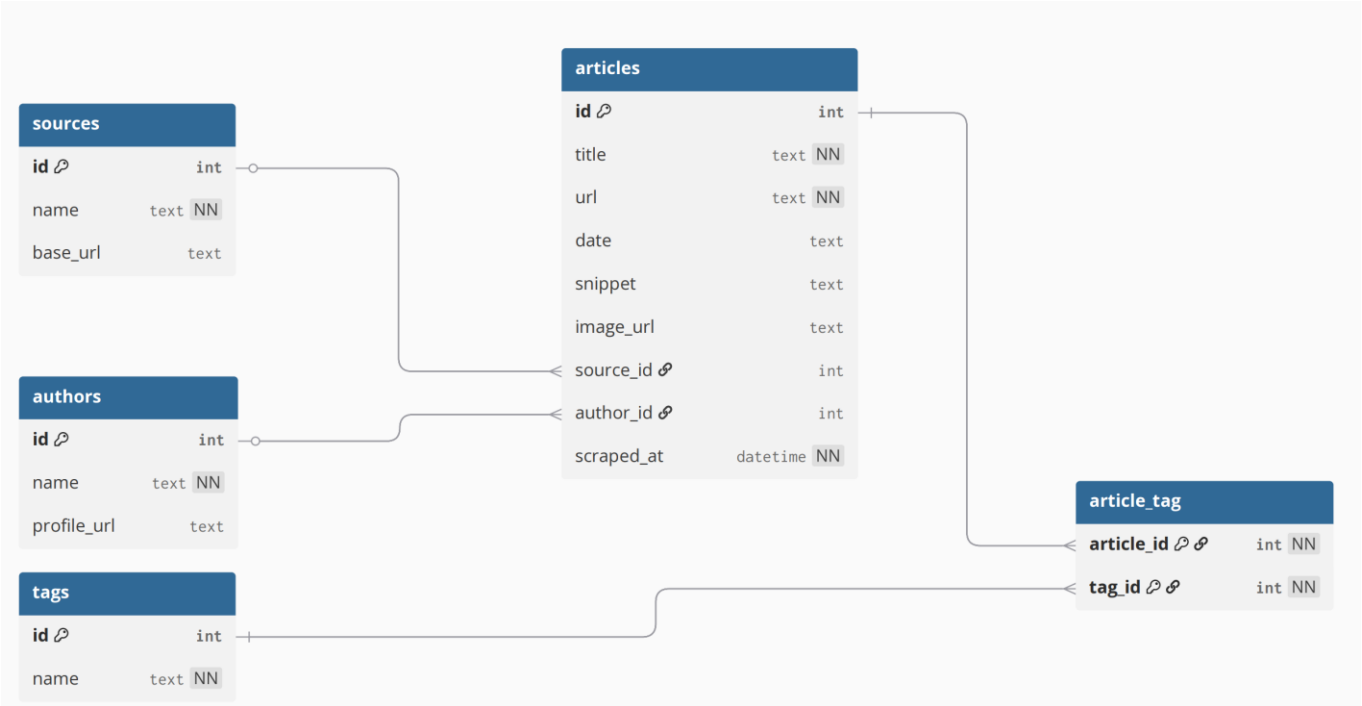

Abbildung 1 DB Diagramm

3.3 Legende zu Diagrammsymbolen

Symbol / Abkürzung	Bedeutung
PK	Primary Key – Primärschlüssel der Tabelle
FK	Foreign Key – Fremdschlüssel, verweist auf einen Primärschlüssel einer anderen Tabelle
NN	Not Null – Feld darf nicht leer sein
UQ	Unique – Wert muss innerhalb der Tabelle eindeutig sein
m:n	Viele-zu-Viele-Beziehung
1:n	Eins-zu-Viele-Beziehung
1:1	Eins-zu-Eins-Beziehung

4 Fazit und Lessons Learned

Im Rahmen dieser Arbeit wurde das Datenbankdesign des Projekts 'News Scraper FLZ' von einer einfachen Einzel-Tabellen-Struktur (IST) zu einem normalisierten und erweiterbaren Modell (SOLL) weiterentwickelt. Durch die Trennung von Artikeln, Quellen, Autoren und Schlagwörtern konnten Redundanzen minimiert und die Grundlage für zukünftige Erweiterungen geschaffen werden.

4.1 Lessons Learned

- Änderungen an der Website-Struktur: Bei der ersten Version des Scrapers wurden die Artikel-Links direkt im HTML-Element <article> gefunden. Später stellte sich heraus, dass 20Minuten die Struktur geändert hatte – die Artikel waren nun im JavaScript-basierten Code (getArticle) eingebettet und nicht mehr im ursprünglichen <article>-Container. Dadurch schlugen die bisherigen Selektoren fehl, und der Scraper musste entsprechend angepasst werden.

- Anpassung der Datenexporte: In einer früheren Version war bereits eine Exportfunktion für .xlsx-Dateien implementiert. Diese konnte relativ einfach um eine SQL-Speicherfunktion erweitert werden. Durch die neue Exportmöglichkeit in die SQLite-Datenbank wurde eine direkte und automatisierte Speicherung der gescrapten Artikel ermöglicht.

- Zusammenwirken von Code und Datenbankdesign: Es zeigte sich, dass ein gutes Datenbankdesign nicht nur theoretisch sauber sein muss, sondern auch eng mit den technischen Gegebenheiten der Datenquelle abgestimmt werden sollte. Änderungen am Quellcode der Website können unmittelbare Anpassungen in der Datenerfassung erfordern.

4.2 Fazit

Die Umstellung auf das normalisierte Datenmodell hat die Flexibilität und Stabilität des Systems erhöht. Probleme durch Website-Änderungen konnten durch gezielte Anpassungen behoben werden. Die neue Struktur ermöglicht eine schnelle Integration zusätzlicher Funktionen, z. B. Filterungen, komplexe Suchabfragen oder den Import weiterer Nachrichtenquellen.

5 Anhang Code

Code für Untitled - dbdiagram.io

```
// Use DBML to define your database structure
// Docs: https://dbml.dbdiagram.io/docs

Project News_Scraper_SOLL {
  database_type: "SQLite"
  Note: 'Normalisierte DB: sources, authors, tags, articles, article_tag'
}

Table sources {
  id       int    [pk, increment]
  name     text   [not null, unique]
  base_url  text
}

Table authors {
  id        int    [pk, increment]
  name      text   [not null]
  profile_url text
}

Table tags {
  id    int   [pk, increment]
  name  text  [not null, unique]
```

```
}

Table articles {
  id        int      [pk, increment]
  title     text     [not null]
  url       text     [not null, unique] // kanonische URL
  date      text
  snippet   text
  image_url  text
  source_id  int
  author_id  int
  scraped_at datetime  [not null]

  indexes {
    (source_id)  [name: 'idx_articles_source']
    (author_id)  [name: 'idx_articles_author']
    (date)       [name: 'idx_articles_date']
    (scraped_at) [name: 'idx_articles_scraped']
  }
}

Table article_tag {
  article_id int [not null]
  tag_id     int [not null]

  indexes {
```

```
  (article_id, tag_id) [pk, name: 'pk_article_tag']
  (tag_id)           [name: 'idx_article_tag_tag']
 }
}

/* Foreign Keys + Aktionen → gehören in Ref-Zeilen */
Ref:  articles.source_id  > sources.id  [delete: set null, update: cascade]
Ref:  articles.author_id  > authors.id  [delete: set null, update: cascade]
Ref:  article_tag.article_id > articles.id [delete: cascade, update: cascade]
Ref:  article_tag.tag_id   > tags.id     [delete: cascade, update: cascade]
```