Die Schlei: Katalog der submarinen Funde und der Uferfunde.

A SQLite database from the original catalogue.

Christoph Rinne* Oliver Nakoinz†

Version 00 (2025-02-13)

Contents

Introduction

The data is taken from the catalogue provided and analysed by Oliver Nakoinz. The Offa journal is being retro-digitised and the publication will also be available digitally in the future. The current status is 2011.

> O. Nakoinz, Die Schlei: Katalog der submarinen Funde und der Uferfunde. Festgabe Joachim Reichstein. Offa 59/60, 2003/04, 2005, 167–218.

The data was collected, reorganized, reviewed and structured in this database in a course on computational archaeology held at Christian-Albrechts-University at Kiel (CAU Kiel) in the winter term 2024/25. The retrieval of data was the container to introduce text based data processing using regular expressions, basic concepts of a relational database, SQL, Markdown and the interoperability between some open software (Notepad++, DB Browser (SQLite) and QGIS). The overarching goal was to create an awareness of data structures in the context of reusable data and reproducible research.

Important note: In favour of simplicity and size, the database lacks important metadata to be a Spatialite database, you cannot open the SQLite file in QGIS and use the view "fundpunktgeo" directly. Furthermore, unfortunately the database manager of QGIS (3.34.8) cannot attach databases to an existing Spatialite database. Furthermore, you cannot execute multiple SQL statements within a `BEGIN TRANSACTION; [...] COMMIT;`. As a result, you would have to copy and paste each SQL statements into the query composer, an extremely laborious task. This means that although the QGIS (3.34.8) database manager is a great and indispensable tool, it lacks basic functionality. **To re-build a Spatialite database from the SQL statements** please load and execute the SQL script (UTF-8) in Spatialite.

The data collection is available via the LandMan portal of DFG CRC 1266[1] and opendata@uni-kiel.

*Kiel University, crinne@ufg.uni-kiel.de
†Kiel University, oliver.nakoinz@ufg.uni-kiel.de

[1] "Scales of Transformation - Human-Environmental Interaction in Prehistoric and Archaic Societies." Deutsche Forschungsgemeinschaft (DFG) - project number 290391021 https://gepris.dfg.de/gepris/projekt/290391021

Data presentation

All code chunks will be visible, this is part of the reproducibility. The documentation uses some R packages:

```
knitr::opts_chunk$set(echo = TRUE, include = TRUE)
require(pacman) || install.packages("pacman")
```

```
## Lade nötiges Paket: pacman
```

```
## [1] TRUE
```

```
pacman::p_load(dplyr,RSQLite)
```

Please set the working directory to the folder with the data, e.g.:

```
setwd('d:/data/folder/')
```

Set up the database connection.

```
db01 <- dbConnect(RSQLite::SQLite(), dbname = "./nakoinz.o_2005_schlei.v00.sqlite")
```

Data structure

The data is provided in six tables. Two tables are basic and contain the information about the location and related information (fundorte, infos). Two tables (gemeinden, kreise) contain the decoding of the indices for parish (Gemeinde) and district (Kreis). These could have been merged, with a third column added for the organisational hierarchy. The table 'fundpunkte' provides the coordinates. We have decided to separate this from the basic tables, e.g. infos, for two reasons. 1. The original catalogue only contains maps and no coordinates. 2. We decided to use wkt (well known text), which can offer different spatial objects in a text field and uses an additional spatial reference ID (srid) to provide ready-to-use geometry. The last table is needed to ensure a systematic and non-alphabetical order of the attributes (tags) of the information. The tables readme and metadata provide general information on the data set.

The main relation between the archaeological information is the combination of 'gemeinde_nr' and (site_) 'nr'.

```
select row_number() over (order by name) as 'nr',
      name  as 'table name' from sqlite_master where type = 'table' order by 1;
```

Table 1: 8 records

nr	table name
1	fundorte
2	fundpunkte
3	gemeinden
4	infos
5	kreise
6	metadata
7	readme
8	tags

As part of the teaching outcome there are some views, providing reorganised data for different tasks. For a more detailed explanation on the SQL-syntax see below.

```
select row_number() over (order by name) as 'nr',
       name  as 'view name' from sqlite_master where type = 'view' order by 2;
```

Table 2: 7 records

nr	view name
1	Tabelle
2	fo_infos
3	fundpunktgeo
4	katalog_mit_tabelle_1
5	katalog_mit_tabelle_2
6	katalog_schritt_1
7	katalog_schritt_2

The first view reorganises the list of attributes (tags) into a table as an often desired structure. The second summarises all information in a text field that can be used as pop-up information (html) in a QGIS map of the locations. 3. create a ready-to-use geometry object (st_GeometryFromText()). 4 & 5 Create a Markdown text with a table of aggregated (counted) information per location. 6 & 7 Create a Markdown text to create a catalogue with hierarchical grouping and layout.

Import the data into the R environment.

```
tables<- dbGetQuery(db01, "select name from sqlite_master
                    where type = 'table' order by 1;")[,1]
for (tbl in tables){
  assign(tbl, dbReadTable(db01, tbl))
}
```

Most SQLite columns are set to TEXT due to type affinity in SQLite, exception are id fields (nr, gemeinde_nr, kreis_nr.

fundorte

The column names are self-explanatory in general. We first left the numbers as provided (gemeinde_nr/(site_)nr) but separated them in a further step due to 'order by' reasons. The original combination can easily be achieved: `gemeinde_nr || '/' || nr`.

```
paste(colnames(get(tables[1])), collapse = ", ")
```

[1] "gemeinde_nr, nr, fundort"

infos

The field 'wert' provides the information for each 'tag' (attribute) for each site ('gemeinde_nr', 'nr'). Not all tags are provided for each site. In contrast to a table with a column for each attribute, there are no empty cells.

```
paste(colnames(get(tables[4])), collapse = ", ")
```

[1] "gemeinde_nr, nr, tag, wert"

fundpunkte

The table 'fundpunkte' provides coordinates (wkt, srid: 4647) digitised from the georeferenced maps of the original publication.

```
paste(colnames(get(tables[2])), collapse = ", ")
```

[1] "gemeinde_nr, nr, wkt, srid"

Content

fundorte, fundpunkte

The table 'fundorte' provides the original site numbers, a combination of parish number and site number (1/1) in separated columns. In consequence the key value is a combination of both. In addition, some sites have an overarching information and thus a text instead of a single site number, e.g. 1/1-3.

The table 'fundpunkte' provides coordinates for the 218 sites of the catalogue.

```
cities <- rbind.data.frame(
    c("Kappeln", 6057300, 32560000),
    c("Eckernförde", 6036000, 32554000),
    c("Schleswig", 6040800, 32536800),
    c("Haithabu", 6038400, 32536300)
)
colnames(cities) <-c("name", "h", "r")
cities$r<-as.numeric(cities$r)
cities$h<-as.numeric(cities$h)
plot(as.numeric(substr(fundpunkte$wkt,7,14)),
     as.numeric(gsub('^.* (\\d{1,8}).*','\\1',fundpunkte$wkt)),
     pch = 19, cex = 0.4,
     xlab = "UTM32 r", ylab = "UTM32 h",
     xlim = c (32534000,32566000), ylim = c(6035000,6060500));
text(cities$r, cities$h, cities$name, cex = 0.8)
```

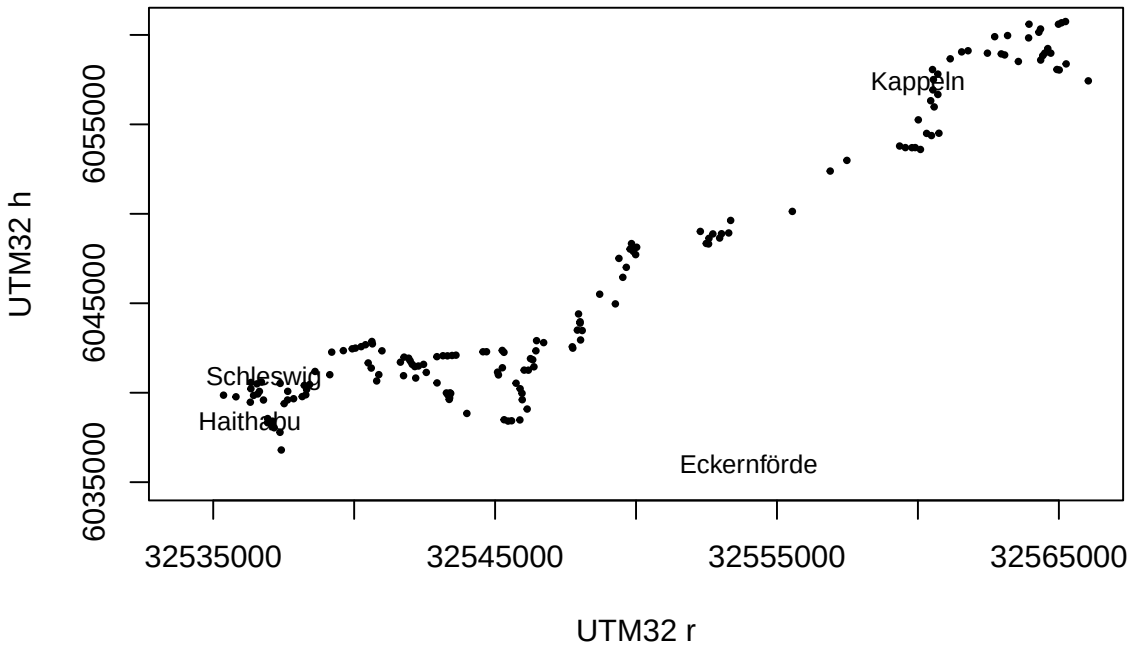

Figure 1: Sites along the Schlei in Schleswig-Holstein.

Content of infos

The table provides 876 information records for the sites.

```
infos %>%
  group_by(., tag) %>%
  summarise(count = n())
```

```
## # A tibble: 5 x 2
##   tag       count
##   <chr>     <int>
## 1 Datierung   114
## 2 Info        218
## 3 LA-Nr       214
## 4 Literatur   134
## 5 Verbleib    196
```

The values for each tag are provided as text and can vary in length from 1 up to 9834 characters. Important note: The literature is provided as short citation with author(s) and year. You need the original text to resolve the full citation, as this is not part of the data provided here.

Views

As already mentioned, the views are created to provide information for various tasks and with increasing complexity in the SQL statements.

```
views <- dbGetQuery(db01, "select name from sqlite_master
                    where type = 'view' order by 1;")[,1]
paste(views, collapse = ', ')
```

```
## [1] "Tabelle, fo_infos, fundpunktgeo, katalog_mit_tabelle_1, katalog_mit_tabelle_2, katalog_schri
```

Tabelle

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'Tabelle';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW Tabelle as
## select k.kreis, g.gemeinde, i.nr,
##  group_concat(case when i.tag="LA-Nr" then i.wert end) as LANr,
##  group_concat(case when i.tag="Datierung" then i.wert end) as Dat,
##  group_concat(case when i.tag="Info" then i.wert end) as Info,
##  group_concat(case when i.tag="Literatur" then i.wert end) as Literatur
## from infos as i
## left join gemeinden as g on i.gemeinde_nr = g.nr
## left join kreise as k on g.kreis_nr = k.nr
## group by 1, 2, 3
## order by g.nr, cast(i.nr as integer)
```

The aim is to obtain a typical tabular structure of the data. In this case, this is less useful for long texts, but may be necessary for measurements or lists of objects. The SQL statement joins the table infos with *left joins* to the tables 'gemeinden' and 'kreise' in order to obtain the respective full names. The resulting list is grouped according to the first three columns: 'kreis', 'gemeinde' and (site) 'nr'. For the following columns, the *group_concat()* function is used to concatenate the corresponding values. These values, i.e. all tags, are selected by the expression *case when* to match only a specific tag. If necessary, you should specify a collapse-string, e.g. ', ' instead of the default value ','. The entire list is sorted by 'gemeinde_nr' and the (site) 'nr', converted to an integer.

fo_infos

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'fo_infos';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW fo_infos as
## select gemeinde_nr || '/' || nr as fonr, group_concat(info, char(10)) as 'infos'
## from (select i.gemeinde_nr, i.nr, i.tag || ': ' || i.wert as info
##   from infos as i
##   left join tags as t on i.tag = t.tag
##   order by i.gemeinde_nr, cast(i.nr as integer), t.nr)
## group by 1
## order by gemeinde_nr, cast(nr as integer)
```

The goal is to get a text with all the information for a global search and a pop-up information in QGIS, but each tag (attribute) with its value in a new line. The SQL statement consists of two queries. The internal or sub-query returns the data, the outer query summarises the data for each site. The sub-query links the infos table with the tags table to ensure a consistent order of the tags. The first and second columns are the identifiers for the parish and the site, the third column combines the tag, ':' and the value into a string labelled 'info'. The outer query groups this list according to the first column, the combination of 'gemeinde_nr', '/' and (site_)'nr'. The second column uses the *group_concat()* function to concatenate the respective strings of the info column in one cell, separated by a line break = char(10).

fundpunktgeo

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'fundpunktgeo';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW fundpunktgeo as
## select "gemeinde_nr"||'/'||"nr" as id, "gemeinde_nr", "nr",
##    st_geomfromtext(wkt,srid) as geometry
## from fundpunkte
```

The aim is to provide a ready-to-use geometry including the srid into a GIS by using the *st_GeometryFromText()* function to convert the WKT string into a binary object.

##katalog_mit_tabelle_1 & katalog_mit_tabelle_2

The goal is to provide an SQL statement that returns a catalogue in Markdown including a table with summary information. In our example, we count the first character of the tags for each site, which is useless, but just an example. For simplicity, we split the task into two separate views with increasing complexity. Both views provide inline comments (–).

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'katalog_mit_tabelle_1';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW katalog_mit_tabelle_1 as
## -- Beispiel für Markdown mit einer Tabelle je Fundplatz.
## -- Erstell eine Überschrift H1, dann eine Kopfzeile und
## -- hängt dann die konkatenierten Elemente einer Zählung (s.u.) an.
## -- Alles wird mit char(10) = neue Zeile = LF = \n getrennt.
## select '# ' || fon || char(10) as DS,
##    '| Buchstabe | Anzahl |' || char(10) ||
##    '|:---------:|-------:|' || char(10) ||
##    group_concat('| ' || tag1 || ' | ' || anzahl || ' |', char(10))  || char(10) as tab
## -- Unterabfrage für die zusammenstellung der Daten.
## -- Die Funktion substr() liefert den ersten Buchstaben als zu zählende Variable.
## from (select gemeinde_nr || '/' || nr as fon,
##           substr(tag, 1, 1) as tag1,
```

```
##           count(*) as anzahl
##       from infos
##       group by 1, 2
##    order by 1, 2)
## group by 1
## order by 1
```

The first sql statement has a sub-query to collect and aggregate the data by site and the first character of each tag. The outer query aggregates the provided data for each site into two columns. The first column concatenates '#' and the site identifier with a subsequent newline (char(10)). The second column concatenates a table header in markdown with the result of a subsequent *group_concat()* of the corresponding records.

```
sql <- dbGetQuery(db01, "select * from katalog_mit_tabelle_1 limit 1;")
cat(gsub('\r\n', '  \n', sql))
```

```
## # 1/1
##  | Buchstabe | Anzahl |
## |:---------:|-------:|
## | D | 1 |
## | I | 1 |
## | L | 2 |
## | V | 1 |
```

The second query simply adds another outer query to aggregate the parish as the first hyphenation (H1), and sets the site including its name as the second hyphenation (H2). This may look confusing, but it works pretty well. Important note: When exporting the result of this query, please omit string delimiters, e.g. "'' and column separation, e.g. ';' or '\t'.

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'katalog_mit_tabelle_2';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW katalog_mit_tabelle_2 as
## -- Beispiel für Markdown mit einer Tabelle je Fundplatz und Gemeinde-Nr.
## -- Betrachten Sie ggf. erst katalog_mit_tabelle_1
## select '# ' || gemeinde_nr || char(10) as H1,
##       group_concat(DS || char(10) || tab, char(10)) as info
## from
## (select gemeinde_nr,
##    '## ' || nr || ': ' || fundort || char(10) as DS,
##    '| Buchstabe | Anzahl |' || char(10) ||
##    '|:---------:|-------:|' || char(10) ||
##    group_concat('| ' || tag1 || ' | ' || anzahl || ' |', char(10))
##    || char(10) as tab
## from (select f.gemeinde_nr, f.fundort, i.nr,
##           substr(i.tag, 1, 1) as tag1,
##           count(*) as anzahl
##       from infos as i
##    left join fundorte as f on i.gemeinde_nr||'/'||i.nr = f.gemeinde_nr||'/'||f.nr
##       group by 1, 2, 3, 4
##    order by 1, 2, 3, 4)
## group by 1, 2
## order by 1, 2)
## group by 1
## order by 1
```

##katalog_schritt_1 & katalog_schritt_2

The goal is to provide an SQL statement that returns a catalogue in Markdown that is similar to the published version. The strategy is similar to the previous views, but in this case the views are

interdependent. Both views provide inline comments (–). Important note: When exporting the result of this query, please omit string delimiters, e.g. "'' and column separation, e.g. ';' or '\t'.

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'katalog_schritt_1';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW katalog_schritt_1 as
## --Katalog in Markdown
## --Unterabfrage für die Daten
## select i.gemeinde_nr, i.nr,
##       group_concat('**' || i.tag || '**: ' || i.wert, char(10)||char(10)) as fo_info
## from infos as i
## left join tags as t on i.tag = t.tag
## group by 1, 2
## order by 1, cast(i.nr as integer), t.nr
```

The fist view collects and provides the information grouped by parish and site and ordered by parish, site and tag.

```
sql<-dbGetQuery(db01, "select * from katalog_schritt_1 limit 1;")
cat(gsub('\r\n', '  \n', sql))
```

```
## 1 1 **LA-Nr**: ohne LA-Nr.
##
## **Info**: Bei den Ausgrabungen in der Altstadt von Schleswig wurden hölzerne Schiffsteile entdeck
##
## **Datierung**: Sl, Dendro ca. 1100 n.Chr.; S3, Strat. ca. 1100; S4, Dendro 1256(?) n.Chr.; S10, S
##
## **Verbleib**: ALM
##
## **Literatur**: Crumlin-Pedersen 1973; 1997.
```

The second query adds the site name and the parish using *left join* and concatenates the respective strings with '##' or '###' to a markdown hyphenation.

```
sql<-dbGetQuery(db01, "select sql from sqlite_master
                    where type = 'view' and name = 'katalog_schritt_2';")[,1]
cat(gsub('\r\n', '  \n', sql))
```

```
## CREATE VIEW katalog_schritt_2 as
## --Katalog in Markdown
## --Mit Daten aus der Abfrage (view) katalog_schritt_1
## select '## ' || g.gemeinde || char(10) as H2_Gemeinde,
##        group_concat('### ' || f.gemeinde_nr || '/' || f.nr || ': ' ||
##     f.fundort || char(10) ||
##        k1.fo_info,
##     char(10)
##     ) as H3_Infos
## from katalog_schritt_1 as k1
## left join fundorte as f on k1.gemeinde_nr||'/'||k1.nr = f.gemeinde_nr||'/'||f.nr
## left join gemeinden as g on f.gemeinde_nr = g.nr
## group by 1
## order by g.kreis_nr desc, g.nr
```