Overview over what was done in the R course after lunch November 9 when the theme was how to get data from the databases at the institute.

A large number of packages exist in **R** to help in interfacing databases. Three of these packages, **DBI, ROracle and ROracleUI** have been installed at **MRI**, and are automatically attached in computers set up at the **MRI.**

Those packages contain a large number of functions but for most of the users the functions sql, tables, views and desc. Those functions are from the package ROracleUI and are relatively high

level functions that use function from DBI and ROracle. Only interface with the Oracle database at MRI was discussed so the lower level functions were not discussed at all.

The function tables lists all available Oracle tables. Typical use would be

```
x <- tables()
```

or

```
x <- tables(owner="fiskar")
```

```
names(x)

[1] "owner"    "table"    "space"    "rows"     "analyzed"
```

The column **rows** shows the number of records in the table. In some few tables the value of **rows** is NA (should not be). The number of records times number of columns is an indicator of the size of the table. The number of columns in a table can be found by the command **desc**
which is comparable to the **desc** command in **sql.**

x <- desc("fiskar.lengdir")

x

	name	Sclass	type	len	precision	scale	isVarLength	nullOK
1	synis_id	integer	number	4	6	0	FALSE	TRUE
2	tegund	integer	number	4	4	0	FALSE	TRUE
3	lengd	double	number	8	6	2	FALSE	TRUE
4	fjoldi	integer	number	4	5	0	FALSE	TRUE
5	kyn	integer	number	4	1	0	FALSE	TRUE
6	kynthroski	integer	number	4	2	0	FALSE	TRUE
7	sbt	character	date	64	0	0	TRUE	TRUE
8	sbn	character	varchar2	15	0	0	TRUE	TRUE
9	snt	character	date	64	0	0	TRUE	TRUE
10	snn	character	varchar2	15	0	0	TRUE	TRUE

7740714 rows on 07-FEB-07

With the function **sql** data can be obtained from the Oracle data base by an sql command. Requires an Oracle client to be running on the computer. An example where the number measured and number counted of saithe (tegund=3) in the cruise “B10-92” is.

```
x <- sql("select a.synis_id,fj_talid,fj_maelt from fiskar.stodvar a, fiskar.numer b where a.synis_id =
b.synis_id and tegund=3 and leidangur='B10-92'")
names(x)
[1] "synis.id" "fj.talid" "fj.maelt"
```

synis_id is the index that links the information in the station table (fiskar.stodvar, fiskar.kvarnir, fiskar.lengdir etc. The same sql command can be set up somewhat differently and more clearly as

```
x <- sql("select synis_id,fj_talid,fj_maelt from fiskar.numer b where tegund=3 and b.synis_id in
(select synis_id from fiskar.stodvar where leidangur='B10-92')")
```

The command **in** in sql is comparable to the command **%in%** in R. This kind of nested sql commands can be very useful for example if taking data from the stomach content data base when the number of layers become 3 instead of 2 . (fiskar.stodvar links to faeda.f_fiskar by the index synis_id and faeda.f_fiskar to the table faeda.f_hopar by the index flokk_id.)

Number of

```
library(fjolst)
args(lesa.stodvar)
names(stodvar)
tmp <- lesa.stodvar(leidangur="B10-92")
nrow(tmp) # Number of stations
[1] 146

table(tmp$leidangur) # surveys only one here
names(tmp) # names of the columns in tmp
Oracle  # Variable describing the default behaviour Oracle or not.

tmp <- lesa.stodvar(leidangur="B10-92",oracle=F) #geting data from dataframe
x <- stodvar[stodvar$leidangur == "B10-92",]  #same the lesa.stodvar with oracle=F
x <- stodvar[stodvar$leidangur %in% "B10-92",]# same as before %in% instead of ==
# %in% is the same as in in sql and is very
nrow(x)
nrow(tmp)
saile <- lesa.lengdir(tmp$synis.id,3) # length distribution of saithe in B10-92
saikv <- lesa.kvarnir(tmp$synis.id,3,col.names=c("slaegt","oslaegt")) # otholith samples of saithe
#         in B10-92.  The columns slaegt and oslaegt added to the std synis_id, lengd, aldur
names(saikv)
[1] "synis.id" "lengd"    "aldur"    "slaegt"   "oslaegt"

sainu <- lesa.numer(tmp$synis.id,3) #
 names(sainu)

[1] "synis.id"    "ar"          "veidarfaeri" "tegund"      "fj.talid"
[6] "fj.maelt"    "afli"        "vigt.synis"

# columns ar and veidarfaeri are really from the station file.
hadnu <- lesa.numer(tmp$synis.id,2)
names(hadnu)
hadnu$totnumber <- hadnu$fj.maelt+hadnu$fj.talid  # Total number of haddock
tmp <- merge(tmp,hadnu[,c("synis.id","totnumber")]) # link station file to number file.
tmp[c("synis.id","totnumber")]
nrow(tmp)
[1] 100

#Only 100 of 146 stations have haddock and only those are included.  To get all 146 stations
requires outer join i.e merge with all = T as shown below.

tmp <- lesa.stodvar(leidangur="B10-92")
tmp <- merge(tmp,hadnu[,c("synis.id","totnumber")],all=T)
```

```
tmp[c("synis.id","totnumber")]
tmp$totnumber[is.na(tmp$totnumber)] <- 0 # No haddock record mean 0 haddock at the station not
        NA.  The home made function join would be better.
tmp <- join(tmp,hadnu[,c("synis.id","totnumber"),"synis.id",set=0) # has the parameter set
# the function fjperstod calculates number of given species at each station doing what has been #
described before.
tmp <- fjperstod(tmp,teg=2,name="ysa.stk")
names(tmp)
fjperstod
# Stomach samples
names(ffiskar) # the table f_fiskar i.e predator info
# Synaflokkur 35 is autumn survey
tmp <- stodvar[stodvar$synaflokkur %in% 35,]
table(tmp$ar)
# Greenland halibut stomachs.
grlstom <- ffiskar[ffiskar$synis.id %in% tmp$synis.id & ffiskar$ranfiskur==22,]
nrow(grlstom)
names(grlstom)
# get the preyinfo see how %in% is used.
grlf <- fhopar[fhopar$flokk.id %in% grlstom$flokk.id,]
names(fhopar)
names(ffiskar)
names(grlf)
# Sum the total amount by prey (faeduhopur)

x <- apply.shrink(grlf$thyngd,grlf$faeduhopur,sum,names=c("faeduhopur","amount"))
x
names(x)
#Take 10 random numbers between 1 and 20.  use to demonstrate order and sort.
x2 <- sample(1:20)[1:10]
x2
# Same without replacement
x2 <- sample(1:20,replace=F)[1:10]
x2
sort(x2)
args(sort)
sort(x2,T)
order(x2)
args(order)
names(x)
# look at x when the records are ordered from lowest to highest prey abundanc.
x[order(x$amount),]
# Order from highest to lowest.
x[order(x$amount),decreasing=T,]
x[order(x$amount,decreasing=T),]
ls() #list what is in the directory
names(grlf)
# Start geting the most common prey for each predator length group.  Start by linking predator
# length to prey info.
grlf <- join(grlf,grlstom[,c("lengd","flokk.id")],"flokk.id")
names(grlf)
# sum by length or predator and food item.
```

```
x <-
apply.shrink(grlf$thyngd,list(grlf$faeduhopur,grlf$lengd),sum,names=c("faeduhopur","lengd","am
ount"))
# Too many length groups might want to look at 10cm length groups.
x <-
apply.shrink(grlf$thyngd,list(grlf$faeduhopur,cut(grlf$lengd,seq(10,120,by=10))),sum,names=c("fa
eduhopur","lenfl","amount"))
# Demonstrate how the commands seq and cut work.
seq(10,120,by=10)
cut(1:10,c(0,5,9))
x2 <- cut(1:10,c(0,5,9))
x2
as.numeric(x2)
x2 <- cut(1:10,c(0.1,5.1,9.1))
x2
as.numeric(x2)
x <-
apply.shrink(grlf$thyngd,list(grlf$faeduhopur,cut(grlf$lengd,seq(10,120,by=10))),sum,names=c("fa
eduhopur","lenfl","amount"))
x[1:10,]
x$lenfl <- as.numeric(x$lenfl)
i <- x$lenfl==5
x[i,]
# Show and example of work from the package Logbooks that contains all logbook information ,
#one file for each gear except bottom trawl where there are three files botnv1, botnv2 and botnv3
#split according to period.  Botnv1 1973-1990, botnv2 1991-1996 and botnv3 since 1997.
# All files have information about position, time, effort and catch of most important species.
# the column visir is the unique index connecting the logbook data files similar to synis_id in the
# fiskar database.
library(Logbooks)
names(lina)
names(handf)
table(handf$ar)
names(flotv)
names(botnv1)
# Show how to add information about a species (hlyri) that does not exist in the dataframe for
#longlines.
desc("afli.afli")
 # name          Sclass  type   len precision scale isVarLength nullOK

#1 visir         double  number 8   38        127   FALSE       FALSE

#2 sokntegund    integer number 4    3         0    FALSE       TRUE

#3 tegund        integer number 4    5         0    FALSE       FALSE

#4 afli          integer number 4    7         0    FALSE       TRUE

#5 medalthyngd_gr integer number 4   6         0    FALSE       TRUE

#
```

```
#  8390782 rows on 19-JAN-05

# afli.afli is the table containing all the information about catch in the logbooks.
#select all records for species 13 they are not that many.
x <- sql("select * from afli.afli where tegund=13")
names(x)
[1] "visir"          "sokntegund"     "tegund"         "afli"

[5] "medalthyngd.gr"

find("x")
#[1] ".GlobalEnv"
 x only found here in the place where we are writing.
# join information abut hlyri to the longline data.
lina <- join(lina,x[,c("visir","afli")],"visir")
names(lina)
names(lina)[36] <- "hlyri" # the name of this column was afli that is not very descriptive.
find("lina")
#[1] ".GlobalEnv"       "package:Logbooks"

# we used the name lina again so we have this name now in 2 places.  Try to avoid this kind
# of situation as it can lea to confusing results.

savehistory() # save all commands in .Rhistory
```

Some older material about the same thing.

Notes on the Marine reasearch Institute databases and how to access them with R, Splus and sql.

What here follows is referred to the fish measurement database but the main points should be applicable to other databases at the institute. The material is not comprehensive, mostly limited by the authors knowledge. All colum names are in Icelandic but appendix A has an English translation of some names. Many of MRI databases are described on the internet under www.hafro.is/maps/hafrogogn2.map
In this description reference is often made to *stoðtöflur* (support tables). These are small tables including meaning of things like species number (orri.fisktegundir), maturity stages (fiskar.kynthroski) , gearcode (orri.veidarfaeri), name and number of ships (orri.skipaskrá) and sampling type (fiskar.synaflokkar) etc.

The databases at the MRI are all what is called "relational databases". Then different tables are related an index and the relation must always be one to many or one to 1. The simple example shown in figure 1, taken out of the MRI measurement database serves as an example. The figure shows two tables fiskar.stodvar and fiskar.lengdir The first table includes information on each station and the second information on the length measurements at each station. The column connecting those two is called "synis_id" (means sample_id). Of course there is only one station connected to each length measurment but there can be many length measurements associated with each station so the relation is **one to many**. So how do we select from these tables with sql.

First *sql* is started by the command *sqlplus /.* (another option is esc-X-sqlplus in emacs). In sql the column names of a table can be found by the command *decribe* for example if *describe fiskar.stodvar*. The answer to this question is quite commonly *Object does not exist* which either means that the table does not exist or the user does not have access to the table. The latter problem is quite common and can lead to strange error messages in when sql is called from another program.

Getting all stations in survey 'B10-92' take all the columns.

*select * from fiskar.stodvar where leidangur = 'B10-92':* 1)

Take only 3 columns synis_id, depth and station and add the condition that depth must be registered.

select synis,dypi_kastad,stod from fiskar.stodvar where leidangur = 'B10-92' and dypi_kastad is not NULL; 2)

In Oracle the expression NULL is associated with a missing value.

The next step is to find a way to select all length measurements associated with certain stations. One way to do so is to write

select synis_id,lengd,fjoldi from fiskar.lengdir where synis_id in (60103,60106,60107,60108); *3)*

This method will obviously become rather cumbersome when the stations are more than few so some method of selecting the right station out of the station file and using that selection will be neeeded. This is where joining of tables comes into the picture but the type of queries shown in 3 is common if queries are generated in R. (see later). Example of query when tables are joined is this query which selects all length measurements of cod (*tegund=1*) in survey 'B10-92' where the number of the statistical square is greater than 500. *(reitur > 500)*

select fiskar.stodvar.synis_id,lengd,fjoldi,reitur from fiskar.stodvar, fiskar.lengdir where where fiskar.stodvar.synis_id = fiskar.lengdir.synis_id and leidangur='B10-92' and reitur > 500 and tegund = 1; *4)*

Or equivalently to make the query a little simpler looking

select a.synis_id,lengd,fjoldi,reitur from fiskar.stodvar a, fiskar.lengdir b where a.synis_id = b.synis_id and leidangur='B10-92' and reitur > 500 and tegund = 1; *4a)*

The only difference here is that the abbreviations a and b are used for the tables fiskar.stodvar and fiskar.lengdir. The table name must be associated with the column synis_id as it exists in all tables. The main difference between this query and the former is the joining condition where *a.synis_id = b.synis_id.* This condition is to ensure that only those records of table be that match the selected records of table a will be selected.

The final step is what to do if we want to take sum, mean or some other function disaggregated by something. Then the expression group by in sql becomes important but it tells by which variables to disaggregate. An example is this queary which counts the number of fishes measured in survey B10-92 disaggregated by square, length, and species. Note that all the variables to be used in the disaggregation must be included in the list of selected variables as anything else would not make sense.

select reitur,lengd,tegund,sum(fjoldi),count(*) from fiskar.stodvar a, fiskar.lengdir b where a.synis_id = b.synis_id and leidangur='B10-92' group by reitur,lengd,tegund; 5)

The function *count(*)* counts the number of records for each disaggregation. Other variables than those used in the disaggregation can usually not be included. A query like

select reitur,lengd,tegund,sum(fjoldi),count(*) a.synis_id from fiskar.stodvar a, fiskar.lengdir b where a.synis_id = b.synis_id and leidangur='B10-92' group by reitur,lengd,tegund; 5a)

is a nonsense. *synis_id* can not be associated with the selected disaggregation.

One important thing is how to treat missing values in sql. Missing values present difficulties in many applications and the user must know how to treat them. In **R** missing values are called **NA** but in sql **NULL**. The most common actions taken when missing value is present are:

1. Skip the record. An example could be if temperature was recorded at each station in a survey but was missing at some stations. Those stations would then not be used in calculations of mean temperature.
2. Put the value to 0. An example would be if the mean catch per area was to be found from a database with each tow registered. Areas with no tows registered would then be included with zero catch.

An example of a query where only records with age, maturity and sex registered are selected is.

select a.synis_id,kyn,kynthroski,lengd,aldur from fiskar.stodvar a, fiskar.kvarnir b where a.synis_id = b.synis_id and leidangur='B10-92' and reitur > 500 and tegund = 1 and aldur is not NULL and kyn is not NULL and kynthroski is not NULL; (6).

In sql two types of joining exist, join and outer join. In outer join all the records are included but else only the records that exist in both tables. To take an example the tables fiskar.stodvar and fiskar.numer will be connected. In fiskar.numer the number measured and counted fishes are stored. (fj_maelt, fj_talid)

select a.synis_id,kastad_n_breidd,kastad_v_lengd,NVL(fj_maelt,0)+NVL(fj_talid,0) from fiskar.stodvar a, fiskar.numer b where a.synis_id = b.synis_id and leidangur='B10-92' and tegund = 1; (7)

Using R to acess Oracle.

The following section describes ways to get data from Oracle into R or Splus. Most of commands do only apply to unix systems or windows computers running R and Oracle client. Working on a network connected PC the user can get the data by starting Splus on unix in the reflection window, and then transferring the data to the PC as described in section,

There are various ways to acess Oracle from Splus and R. One is to use a function called ImportData that can read from Oracle tables. This function which can also read excel tables and ascii files is described in the help. It is though relatively limited for reading from Oracle. If an Oracle client is installed on a windows computer R has no problem in getting data from the Oracle database.

function called **sql** is available on the unix and computers and windows computers running Oracle client. The function takes one required argument which is the sql command. An example is this command which finds all cod larger than 100cm and stores in an object called Codgt100

*Codgt100 <- sql("select * from fiskar.lengdir where tegund = 1 and lengd > 150")*

8)

One important thing about the function sql is that the command should not end with *;* as usually in sql. If the *;* is included an error message will appear. The function sql changes _ in column names to . so a column with the name *synis_id* in Oracle will be *synis.id* in Splus. This is for convenience as _ has be reserved for assignment in Splus and a column with the name *synis_id* would have to be referred to within quotemarks. (*data$synis_id* would not work but *data$"synis_id"* would be needed).

Writing sql commands where a number of tables are joined can at times be rather cumbersome. Even reading from the station file can be cumbersome as it is currently split in 3 parts that have to be joined. (fiskar.stodvar, fiskar.togstodvar, fiskar.umhverfi). Hopefully these tables will be merged in the future. It is relatively easy to generate sql commands of the form shown in 3) in Splus. Then the stations of interest are selected stored in a dataframe. In Splus a powerful function called paste is available to generate the sql. To take an example if the stations of interest is stored in a data frame called *st* with *synis.id* one column the command

l*e <- sql(paste("select synis_id,lengd,fjoldi from fiskar.lengdir where tegund = 1 and synis_id in ("paste(st$synis.id,collapse=","),")")) 9)*

will select all length measurements of species 1 (cod) from those stations. The expression is ugly looking but the user can see the effects of the inner command *paste(st$synis.id,collapse=",")* and also print out the command. What is confusing here is also that some of the brackets are for the sql (those within quotemarks) and some for the Splus command. Users are encouraged to read about the *paste* command and the difference between the arguments *sep* and *collapse* and make a vector

```
command <- paste("select synis_id,lengd,fjoldi from fiskar.lengdir where tegund = 1
and synis_id in ("paste(st$synis.id,collapse=","),")")
print(command)
le <- sql(command)              9a)
```

Splus function for the fish measurement and stomach content database.

The fish measurement database is one of the most commonly used database at the marine reasearch institute and a number of special purpose programs have been made to read from that database. The name of the programs are.

1. lesa.stodvar. Reading station information from fiskar.stodvar, fiskar.togstodvar and fiskar.umhverfi.
2. lesa.numer. Reading information on number of measured and counted fishes, total catch etc. from the table fiskar.numer.
3. lesa.lengdir. Reading information on length distribution from the file fiskar.lengdir.
4. lesa.kvarnir, Reading information on otholith samples from the table fiskar.kvarnir. Information on weighed fish is also stored in this table even though the otholiths are not sampled (age is then NULL)

5. join Connect two different tables by an index. Similar to merge but has some added capability but also a number of limitations compared to merge.
6. lesa.stakir. Read information from the predator table faeda.f_fiskar.
7. lesa.flokkar. Read from the old predator table containing the bulked stomach content data (faeda.f_flokkar) .
8. lesa.hopar. Reading diet information from specified stomachs.
9. add.hopar . Adding information on specified preys to a dataframe generated by lesa.stakir or lesa.flokkar.
10. lesa.f.lengdir read information on length measurements of specified prey at specified stations.
11. Lesa.allir.hopar Getting all stomatch content information from specified stomachs (by flokk.id)
12. fjperstod. Getting the number of fishes at each station. A column with the number of fishes is added to the stationfile.
13. afli.per.stod Getting the amount caught at each station. A column with the number of fishes is added to the stationfile.

All the programs have and argument called *oracle*. If it is TRUE data are obtained from the Oracle database but if it is false the data are obtained from **R** data frames that contain complete Oracle tables. The syntax of the calls is the same whether the data are obtained from the Oracle data base or the data frames. The data frames are a mixture of the tables fiskar.leidr_stodvar,

The name of the dataframes fiskar.stodvar,fiskar.lengdir, fiskar.kvarnir, fiskar.numer and fiskar.leidr_stodvar, fiskar.leidr_lengdir, fiskar.leidr_kvarnir, fiskar.leidr_numer. For few problematic stations 1e6 has been added to the synis_id from the **leidr** tables but those are stations where the synis_id of a old station in the **leidr** tables is the same as of a recent stations in the traditional tables. All synis_id since 1985 are though unchanged. When reading from oracle the function lesa.lengdir , lesa.numer and lesa.kvarnir have the argument leidrett reading from the table fiskar.leidr_*** when this argument is T. A function called lesa.leidr.stodvar does also exist.

When working with stomach content data base many things that are not logical show up. The reason is historical but the data used to be stored in a database completely seperate from the measurment data base and therfore the owner of the tables is "faeda" instead of "fiskar". The most obvious problem is that the predator table (faeda.f_fiskar) is not the same table as fiskar.kvarnir. (the old predator table would for the bulked stomachs would still have to exits)

All data obtained by the function has dot in column names where we have _ in R. (synis_id becomes synis.id etc). The reason is that in Splus _ was interpreted as an assignment operator and could therefore not be used in column names except accessed within quotemarks. If the user does want to keep _ in names **sql** can be called with **dots=F**.

The names of the dataframes in the fjolst package are

stodvar	tables fiskar.stodvar, fiskar.togstodvar and fiskar.umhverfi and fiskar.stodvar_leidr
all.nu	table fiskar.numer and fiskar.numer_leidr

all.le	table fiskar.lengdir and fiskar.lengdir_leidr
all.kv	table fiskar.kvarnir and fiskar.kvarnir_leidr
leidr.stodvar	table fiskar.stodvar_leidr
fflokkar	table faeda.f_flokkar (predator file for bulked stomachs)
ffiskar	table faeda.f_fiskar (predator file for individual stomachs)
fhopar	table faeda.f_hopar (information about prey items)
flengdir	table faeda.f_lengdir(length distribution of prey old samples)
fstaerdir	table faeda.f_staerdir (length distribution of prey new samples)
fkynthroski	table faeda.f_kynthroski. (detailed info about pre ysparimæling)