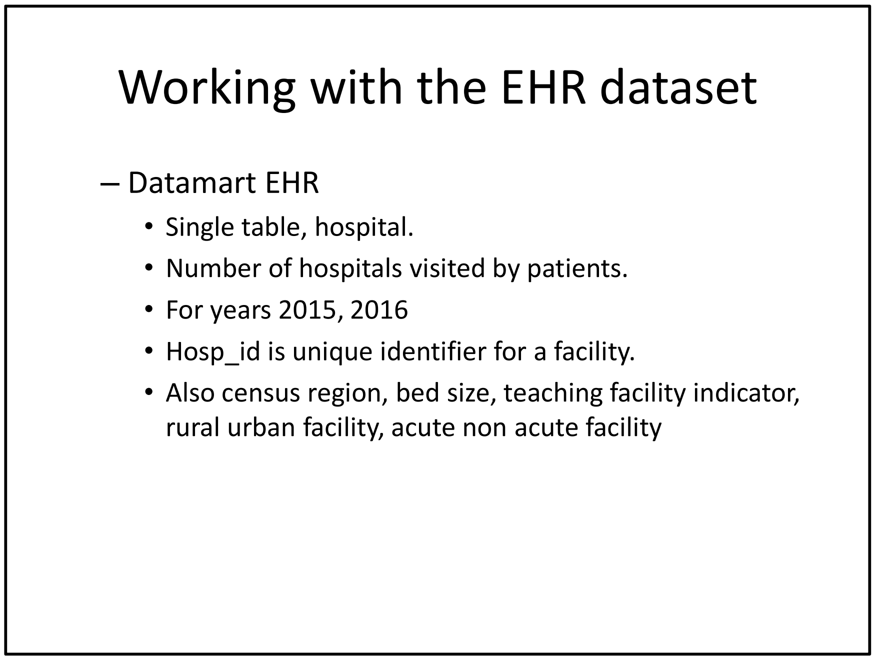
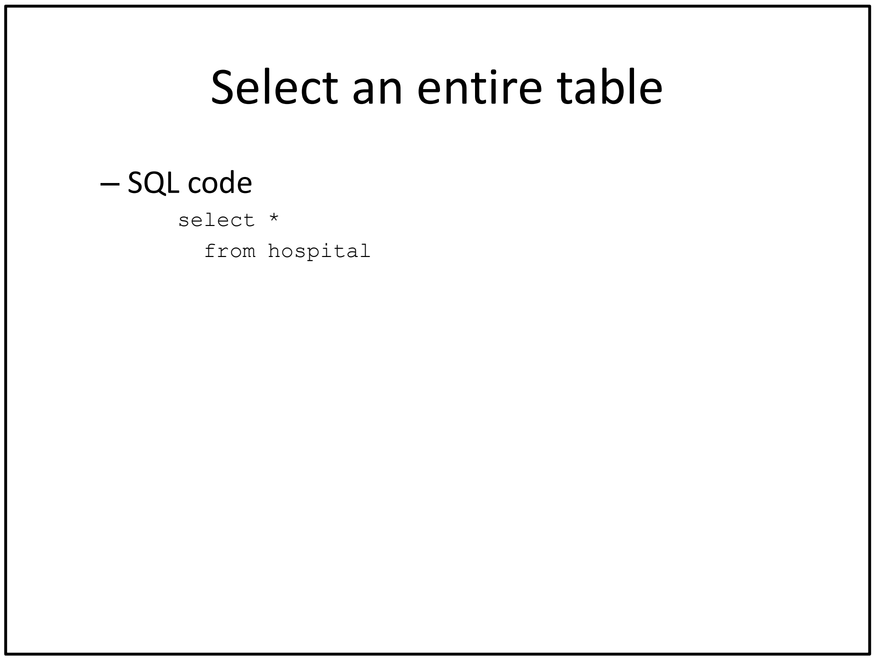
Connecting to Database & Running SQL from inside R and SAS

Suman Sahil, Steve Simon


Working with the EHR dataset

- Datamart EHR
  - Single table, hospital.
  - Number of hospitals visited by patients.
  - For years 2015, 2016
  - Hosp_id is unique identifier for a facility.
  - Also census region, bed size, teaching facility indicator, rural urban facility, acute non acute facility

The first data set is small. The EHR database has a table named hospital. The table has six fields and 108 records. It is a derived from deindentified electronic health records .

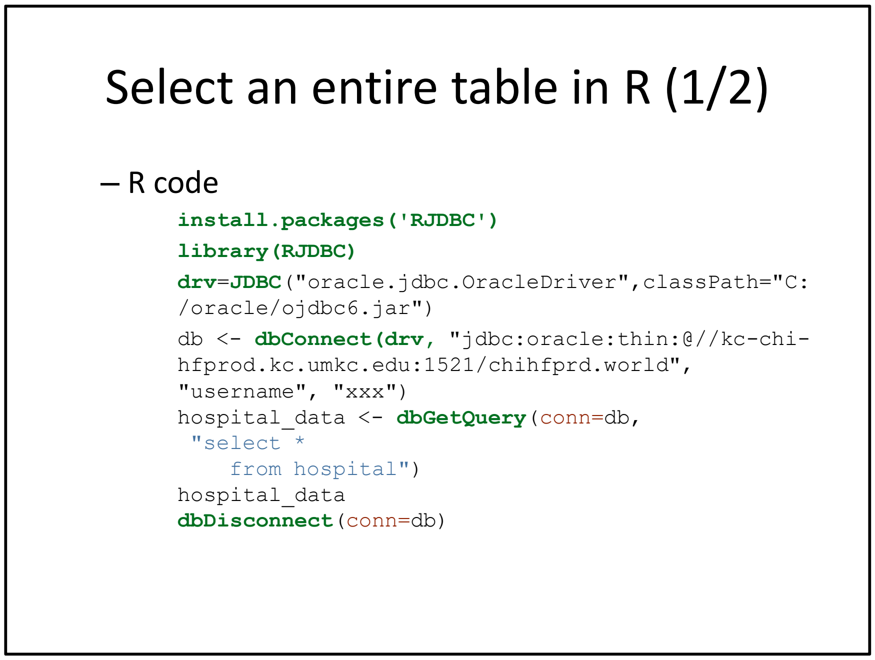


Select an entire table

– SQL code

```
select *
  from hospital
```

To select an entire table in SQL use the wild card symbol, asterisk. The asterisk is shorthand for “every field in the table.” If you are working directly with your database, you do not need any extra code, but in R and SAS there’s just a bit more to it. I wanted to illustrate extra SAS and R code to connect with the database.


Select an entire table in R (1/2)

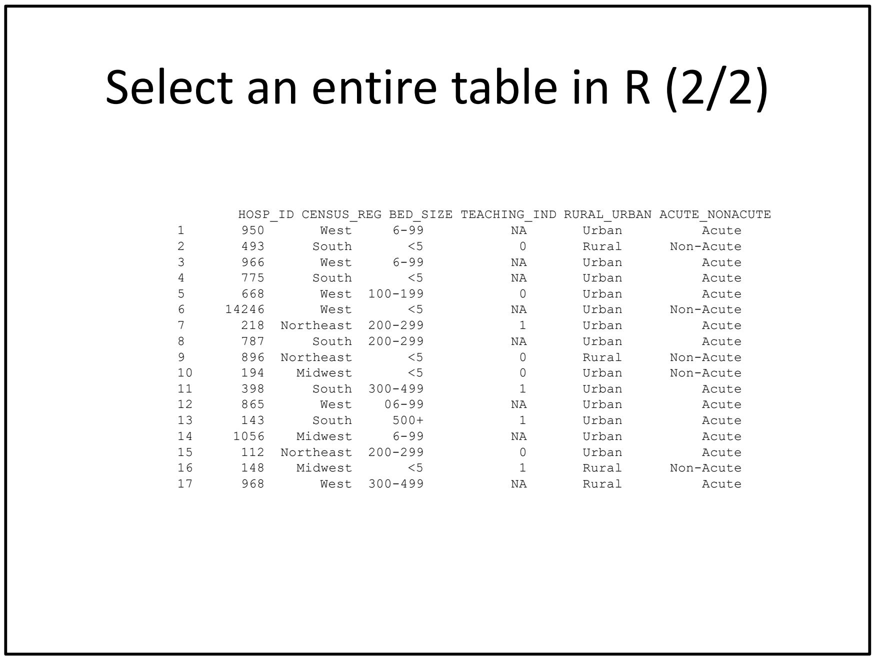
– R code

```
install.packages('RJDBC')
library(RJDBC)
drv=JDBC("oracle.jdbc.OracleDriver",classPath="C:
/oracle/ojdbc6.jar")
db <- dbConnect(drv, "jdbc:oracle:thin:@//kc-chi-
hfprod.kc.umkc.edu:1521/chihfprd.world",
"username", "xxx")
hospital_data <- dbGetQuery(conn=db,
 "select *
    from hospital")
hospital_data
dbDisconnect(conn=db)
```

In R, you need a third party extension. There is a generic library dbi and an extension to Oracle called RJDBC. RJDBC is a package implementing DBI in R on the basis of JDBC. This allows the use of any DBMS in R through the JDBC interface. The only requirement is working Java and a JDBC driver for the database engine to be accessed. The R package RJDBC is an implementation of the R DBI package – database interface – that uses JDBC as the back-end connection to the database. Any database that supports a JDBC driver can be used in connection with RJDBC.

You also need to connect to the database before you can extract any information. use the dbConnect function for this. For a password protected database, you would need extra arguments for the user name and password.

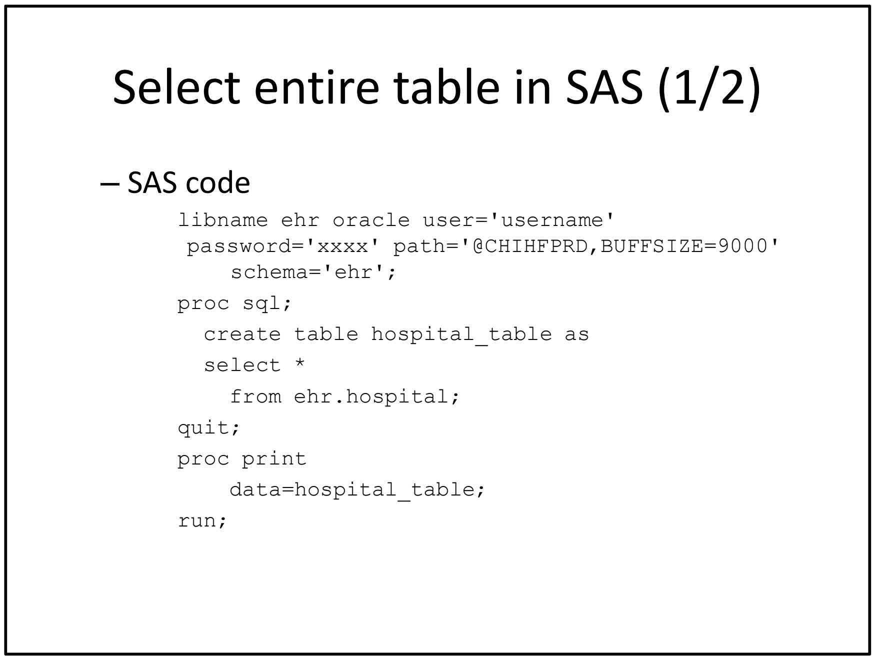
Finally, enclose your SQL code in quotes and pass it to the dbGetQuery function. This function produces a data frame which I have stored in hospital_data. Please remember to disconnect when you are done.



Select an entire table in R (2/2)

```
        HOSP_ID CENSUS_REG BED_SIZE TEACHING_IND RURAL_URBAN ACUTE_NONACUTE
1        950       West     6-99           NA       Urban          Acute
2        493      South       <5            0       Rural      Non-Acute
3        966       West     6-99           NA       Urban          Acute
4        775      South       <5           NA       Urban          Acute
5        668       West  100-199            0       Urban          Acute
6      14246       West       <5           NA       Urban      Non-Acute
7        218  Northeast  200-299            1       Urban          Acute
8        787      South  200-299           NA       Urban          Acute
9        896  Northeast       <5            0       Rural      Non-Acute
10       194    Midwest       <5            0       Urban      Non-Acute
11       398      South  300-499            1       Urban          Acute
12       865       West    06-99           NA       Urban          Acute
13       143      South     500+            1       Urban          Acute
14      1056    Midwest     6-99           NA       Urban          Acute
15       112  Northeast  200-299            0       Urban          Acute
16       148    Midwest       <5            1       Rural      Non-Acute
17       968       West  300-499           NA       Rural          Acute
```

Here’s what the data frame looks like.



Select entire table in SAS (1/2)

– SAS code

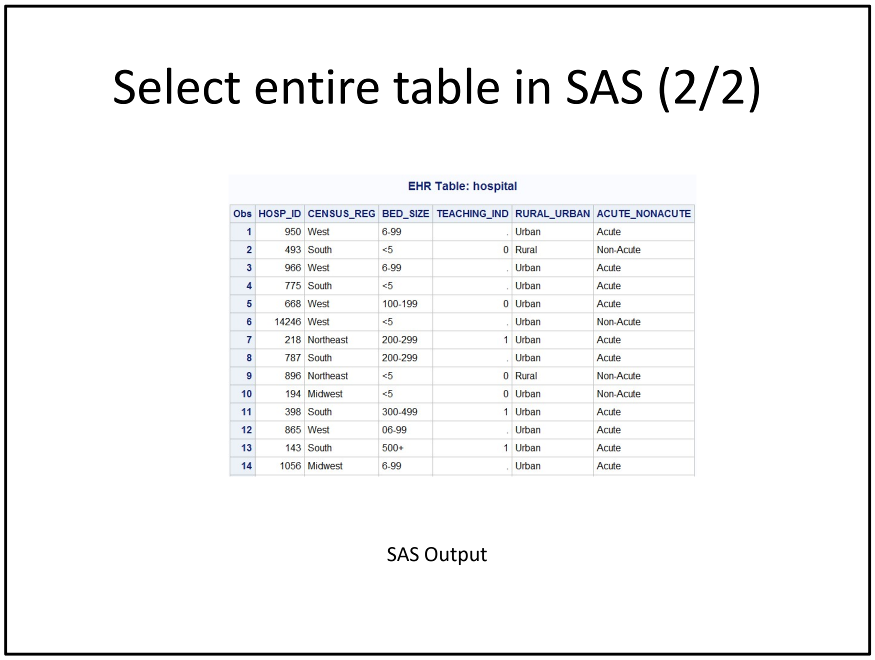
```
libname ehr oracle user='username'
 password='xxxx' path='@CHIHFPRD,BUFFSIZE=9000'
    schema='ehr';
proc sql;
  create table hospital_table as
  select *
    from ehr.hospital;
quit;
proc print
    data=hospital_table;
run;
```

This program shows how to use the SELECT statement for SQL within a SAS program. This code shows all the steps that you need for a simple query that selects every record and all fields within a single table.

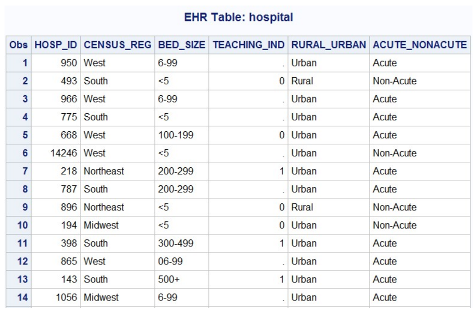
First you need to point to the database with a libname statement. Then you insert the code into proc sql.

By default, proc sql will just display the results of your query. To save a file for further work, use the create table as statement.

Notice that proc sql requires a quit statement rather than a run statement at the end.



Select entire table in SAS (2/2)

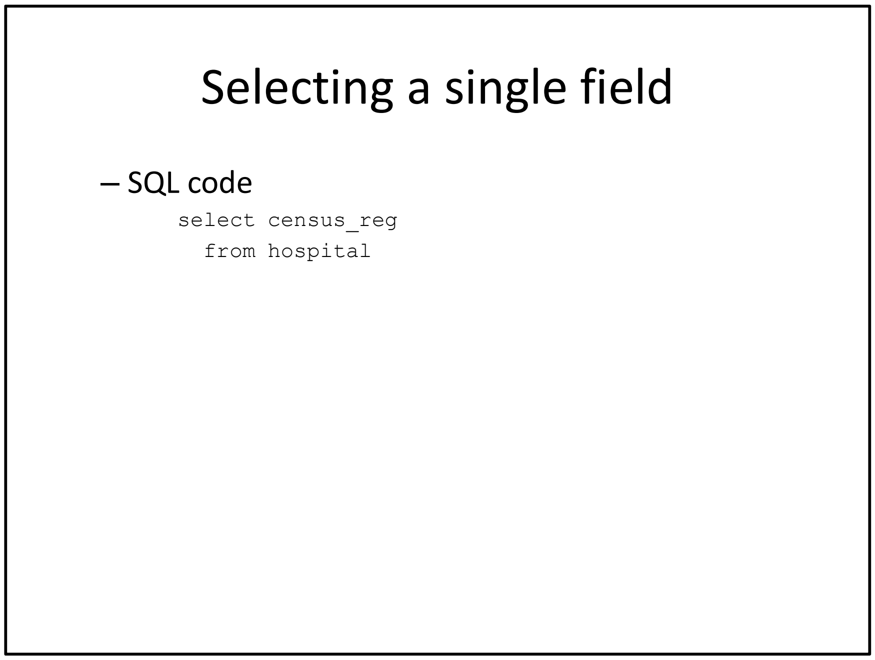


EHR Table: hospital

Obs	HOSP_ID	CENSUS_REG	BED_SIZE	TEACHING_IND	RURAL_URBAN	ACUTE_NONACUTE
1	950	West	6-99	.	Urban	Acute
2	493	South	<5	0	Rural	Non-Acute
3	966	West	6-99	.	Urban	Acute
4	775	South	<5	.	Urban	Acute
5	668	West	100-199	0	Urban	Acute
6	14246	West	<5	.	Urban	Non-Acute
7	218	Northeast	200-299	1	Urban	Acute
8	787	South	200-299	.	Urban	Acute
9	896	Northeast	<5	0	Rural	Non-Acute
10	194	Midwest	<5	0	Urban	Non-Acute
11	398	South	300-499	1	Urban	Acute
12	865	West	06-99	.	Urban	Acute
13	143	South	500+	1	Urban	Acute
14	1056	Midwest	6-99	.	Urban	Acute

SAS Output

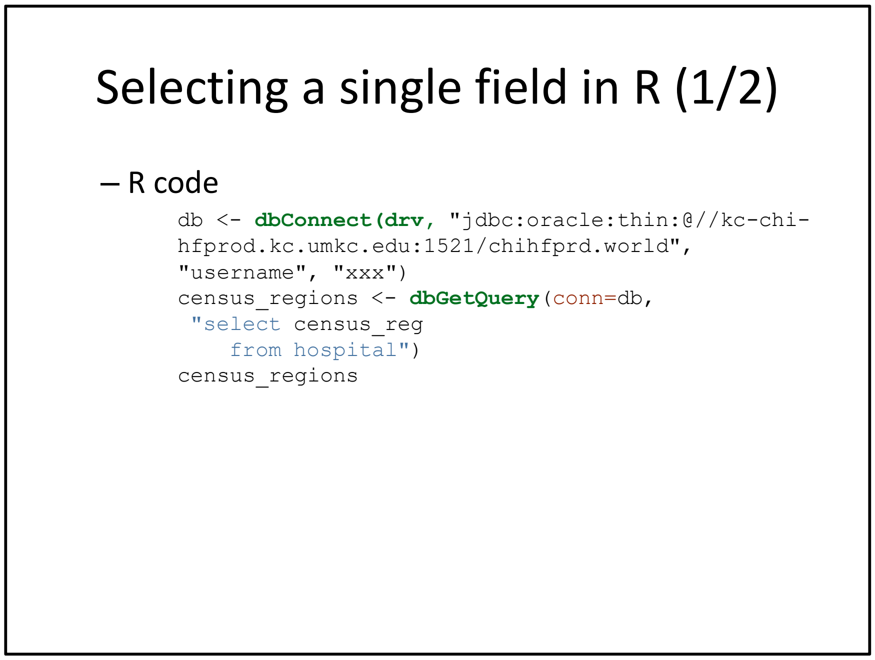
Here's what the output looks like.


Selecting a single field

– SQL code

```
select census_reg
  from hospital
```

To select a single field, list that field’s name after the select statement.

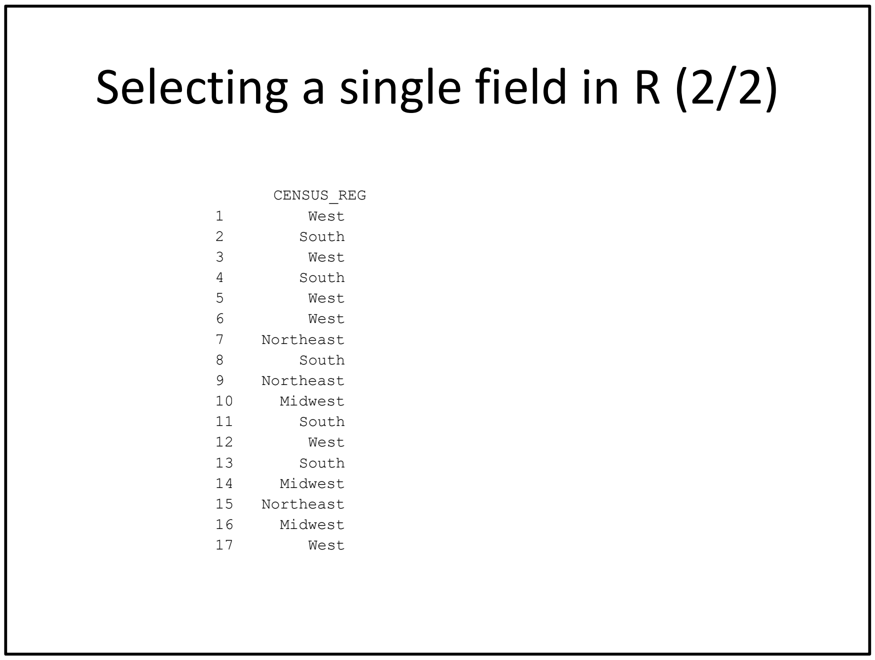


Selecting a single field in R (1/2)

– R code

```
db <- dbConnect(drv, "jdbc:oracle:thin:@//kc-chi-
hfprod.kc.umkc.edu:1521/chihfprd.world",
"username", "xxx")
census_regions <- dbGetQuery(conn=db,
 "select census_reg
    from hospital")
census_regions
```

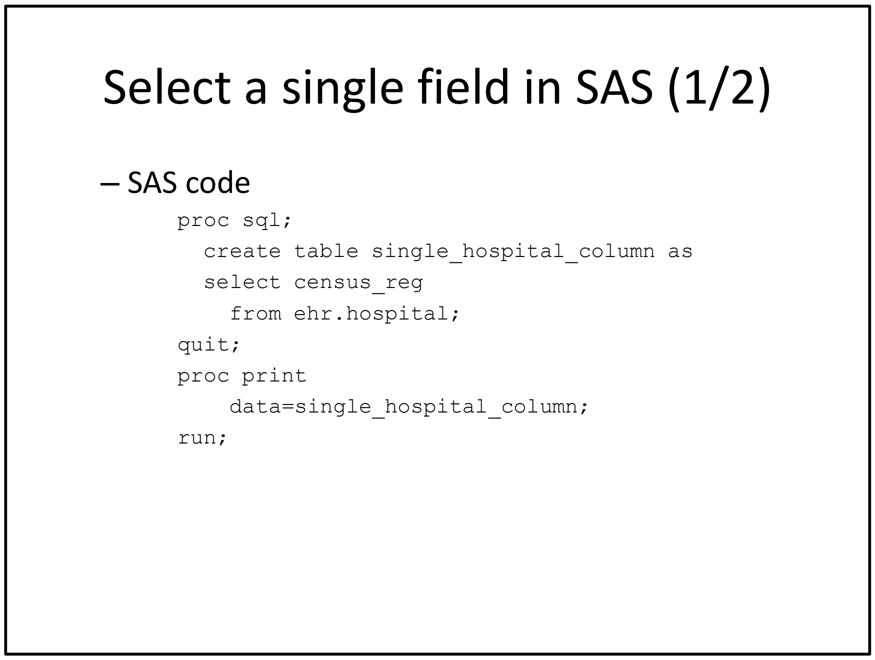
In R, you need to connect again (skip this step if you didn't disconnect earlier). Then call the dbGetQuery function with the SQL code inserted. Keep the connection open for now to save time with the next couple of queries.



Selecting a single field in R (2/2)

```
      CENSUS_REG
1         West
2        South
3         West
4        South
5         West
6         West
7    Northeast
8        South
9    Northeast
10     Midwest
11       South
12        West
13       South
14     Midwest
15   Northeast
16     Midwest
17        West
```

Here’s what the output looks like.



Select a single field in SAS (1/2)

– SAS code

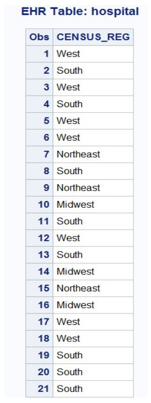
```
proc sql;
  create table single_hospital_column as
  select census_reg
    from ehr.hospital;
quit;
proc print
    data=single_hospital_column;
run;
```

This is how you select a single field in SAS.

As before, place the SQL query inside proc sql and use the create table as statement to store the results in a SAS data set.

Watch your semicolons carefully in SAS!

Select a single field in SAS (2/2)

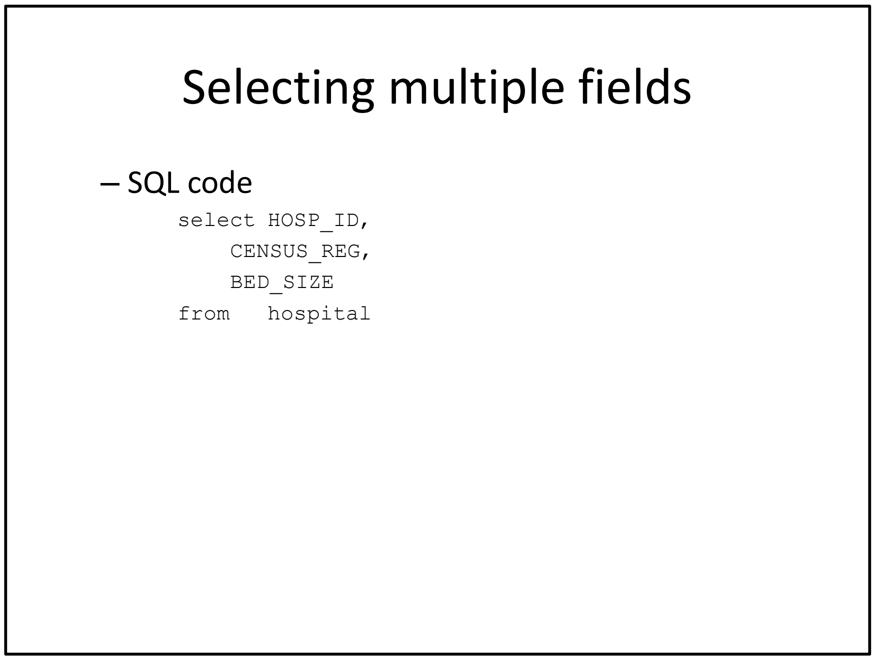


EHR Table: hospital

Obs	CENSUS_REG
1	West
2	South
3	West
4	South
5	West
6	West
7	Northeast
8	South
9	Northeast
10	Midwest
11	South
12	West
13	South
14	Midwest
15	Northeast
16	Midwest
17	West
18	West
19	South
20	South
21	South

SAS Output

Here’s what the output looks like.

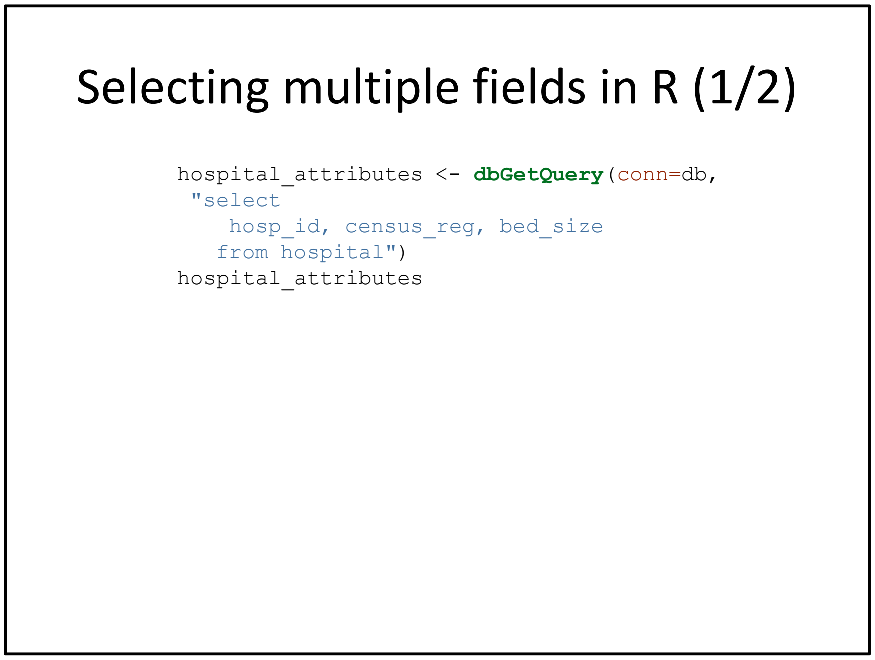


Selecting multiple fields

– SQL code

```
select HOSP_ID,
       CENSUS_REG,
       BED_SIZE
from   hospital
```

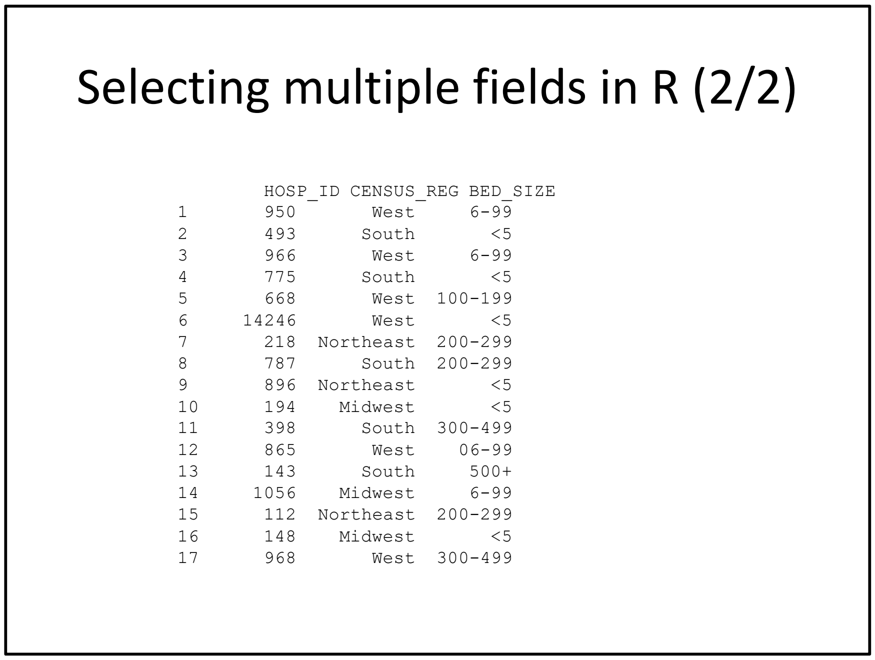
To select multiple fields, list them after the select statement separated by commas. Don’t leave out the commas.



Selecting multiple fields in R (1/2)

```
hospital_attributes <- dbGetQuery(conn=db,
 "select
    hosp_id, census_reg, bed_size
   from hospital")
hospital_attributes
```

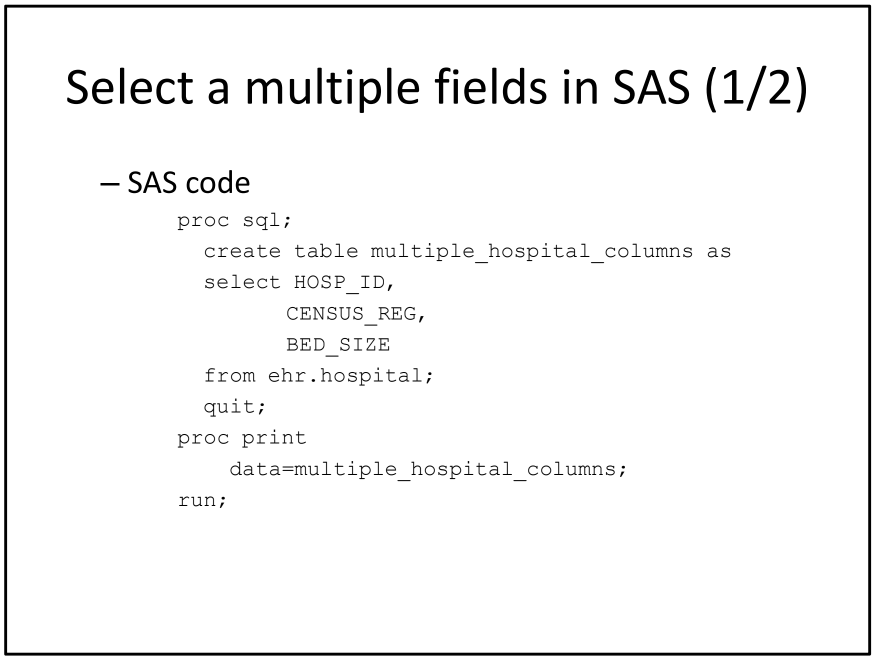
Here’s the code in R.



Selecting multiple fields in R (2/2)

```
        HOSP_ID CENSUS_REG BED_SIZE
1       950       West     6-99
2       493      South       <5
3       966       West     6-99
4       775      South       <5
5       668       West  100-199
6     14246       West       <5
7       218  Northeast  200-299
8       787      South  200-299
9       896  Northeast       <5
10      194    Midwest       <5
11      398      South  300-499
12      865       West    06-99
13      143      South     500+
14     1056    Midwest     6-99
15      112  Northeast  200-299
16      148    Midwest       <5
17      968       West  300-499
```

Here’s what the output looks like.

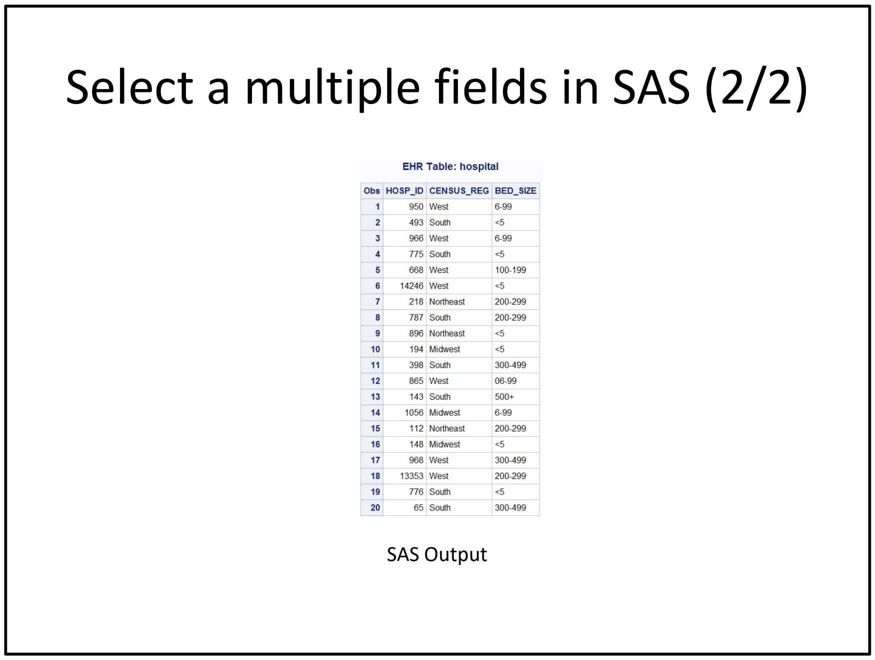


Select a multiple fields in SAS (1/2)

– SAS code

```
proc sql;
  create table multiple_hospital_columns as
  select HOSP_ID,
         CENSUS_REG,
         BED_SIZE
  from ehr.hospital;
  quit;
proc print
    data=multiple_hospital_columns;
run;
```

This is how you select multiple fields in SAS.



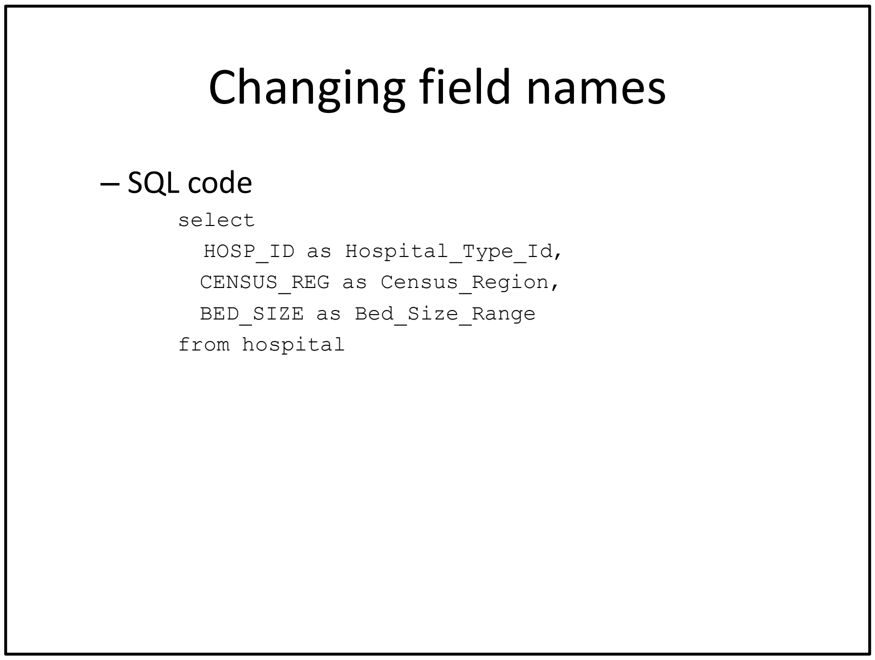
Select a multiple fields in SAS (2/2)

EHR Table: hospital

Obs	HOSP_ID	CENSUS_REG	BED_SIZE
1	950	West	6-99
2	493	South	<5
3	966	West	6-99
4	775	South	<5
5	668	West	100-199
6	14246	West	<5
7	218	Northeast	200-299
8	787	South	200-299
9	896	Northeast	<5
10	194	Midwest	<5
11	398	South	300-499
12	865	West	06-99
13	143	South	500+
14	1056	Midwest	6-99
15	112	Northeast	200-299
16	148	Midwest	<5
17	968	West	300-499
18	13353	West	200-299
19	776	South	<5
20	65	South	300-499

SAS Output

Here’s what the output looks like.

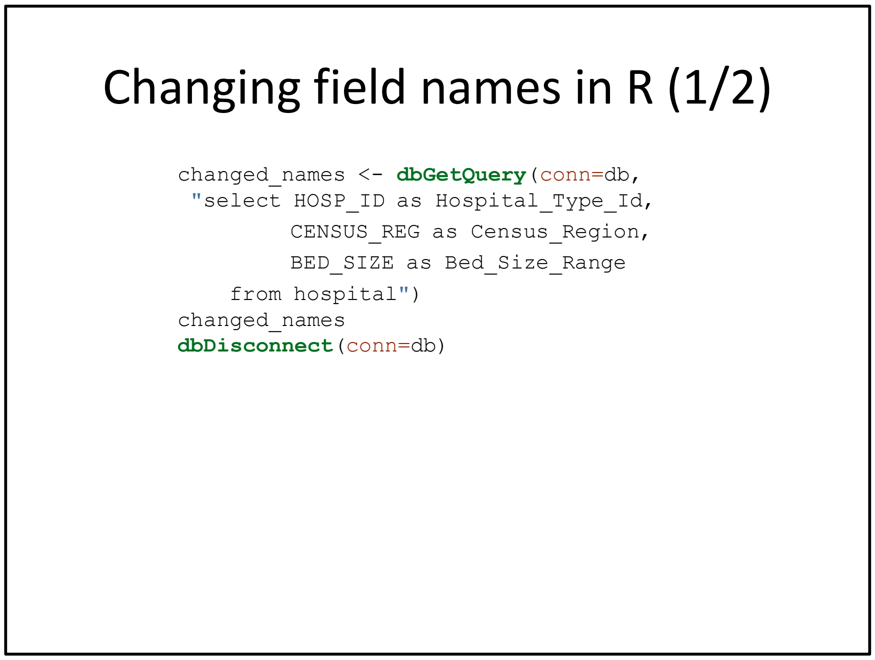


Changing field names

– SQL code

```
select
  HOSP_ID as Hospital_Type_Id,
  CENSUS_REG as Census_Region,
  BED_SIZE as Bed_Size_Range
from hospital
```

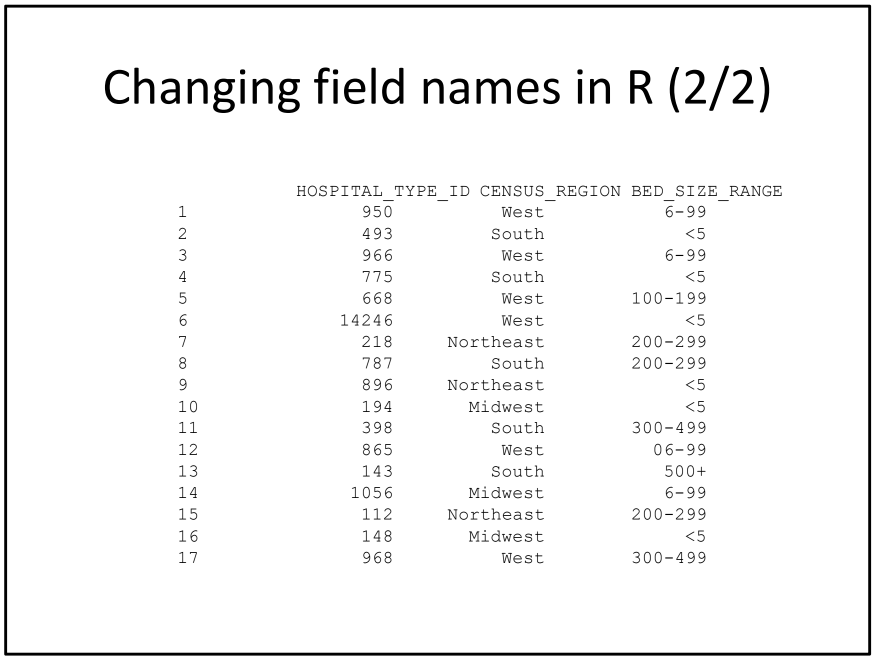
Use the AS keyword to change the name of a field. This code renames the fields in the output, but the names in the original database remain the same. ALIASES can be used to create a temporary name for columns or tables. COLUMN ALIASES are used to make column headings in your result set easier to read.



Changing field names in R (1/2)

```
changed_names <- dbGetQuery(conn=db,
 "select HOSP_ID as Hospital_Type_Id,
         CENSUS_REG as Census_Region,
         BED_SIZE as Bed_Size_Range
    from hospital")
changed_names
dbDisconnect(conn=db)
```

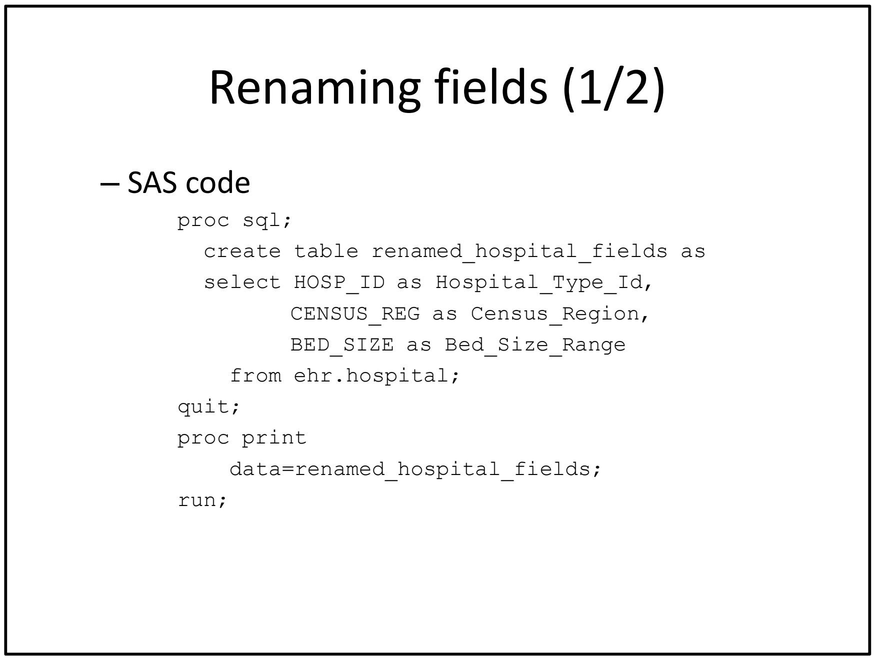
Here's the R code. Since this is the last query in R, you need to disconnect here.



Changing field names in R (2/2)

```
          HOSPITAL_TYPE_ID CENSUS_REGION BED_SIZE_RANGE
1                      950          West           6-99
2                      493         South             <5
3                      966          West           6-99
4                      775         South             <5
5                      668          West        100-199
6                    14246          West             <5
7                      218     Northeast        200-299
8                      787         South        200-299
9                      896     Northeast             <5
10                     194       Midwest             <5
11                     398         South        300-499
12                     865          West          06-99
13                     143         South           500+
14                    1056       Midwest           6-99
15                     112     Northeast        200-299
16                     148       Midwest             <5
17                     968          West        300-499
```

Here's what the output looks like.



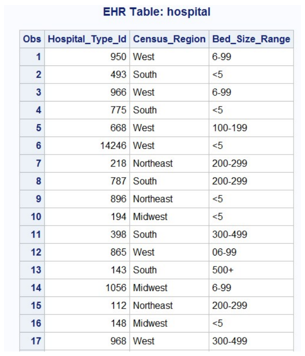
Renaming fields (1/2)

– SAS code

```
proc sql;
  create table renamed_hospital_fields as
  select HOSP_ID as Hospital_Type_Id,
         CENSUS_REG as Census_Region,
         BED_SIZE as Bed_Size_Range
    from ehr.hospital;
quit;
proc print
    data=renamed_hospital_fields;
run;
```

You can rename fields in proc sql, but be careful. Sometimes SAS retains the original name as the variable label. If you have trouble with renaming, you may want to do the renaming in SAS itself.

Renaming fields (2/2)

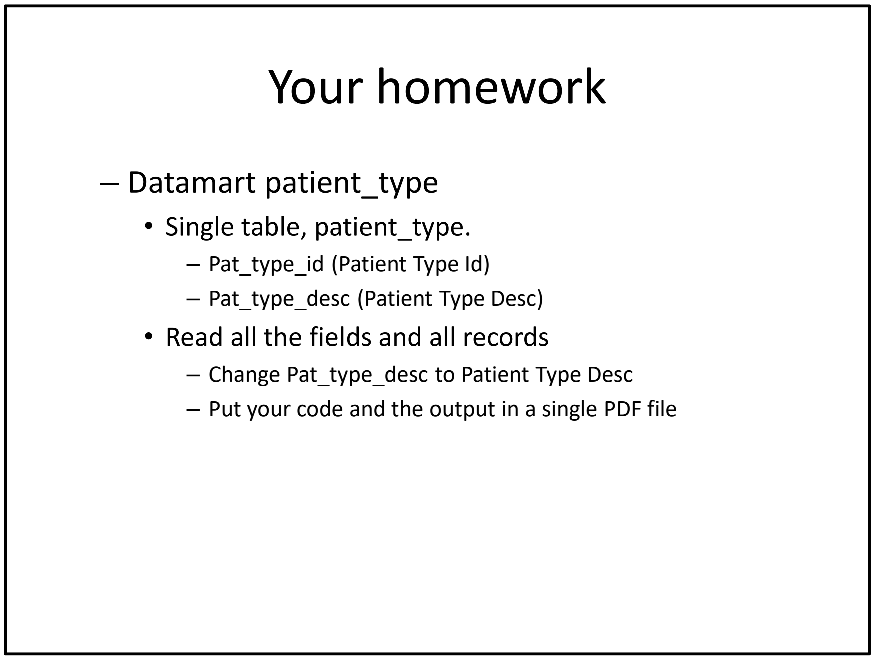


EHR Table: hospital

Obs	Hospital_Type_Id	Census_Region	Bed_Size_Range
1	950	West	6-99
2	493	South	<5
3	966	West	6-99
4	775	South	<5
5	668	West	100-199
6	14246	West	<5
7	218	Northeast	200-299
8	787	South	200-299
9	896	Northeast	<5
10	194	Midwest	<5
11	398	South	300-499
12	865	West	06-99
13	143	South	500+
14	1056	Midwest	6-99
15	112	Northeast	200-299
16	148	Midwest	<5
17	968	West	300-499

SAS Output

Here’s what the output looks like.



Your homework

– Datamart patient_type
    - Single table, patient_type.
        – Pat_type_id (Patient Type Id)
        – Pat_type_desc (Patient Type Desc)
    - Read all the fields and all records
        – Change Pat_type_desc to Patient Type Desc
        – Put your code and the output in a single PDF file

The patient type data for your first homework assignment is small. It has two fields and twelve records. This data came from deindentified electronic health record.