CS 325 Week 3-1

More database fundamentals (DB Reading Packet 2)

(and a little beyond)

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- self-describing: means the database contains METADATA as well as data –

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- self-describing: means the database contains METADATA as well as data –
  - that is, it also contains information about the structure of the data in that database;

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- self-describing: means the database contains METADATA as well as data –
  - that is, it also contains information about the structure of the data in that database;
  - this description of the data structure is called a data dictionary or a data directory

More database fundamentals

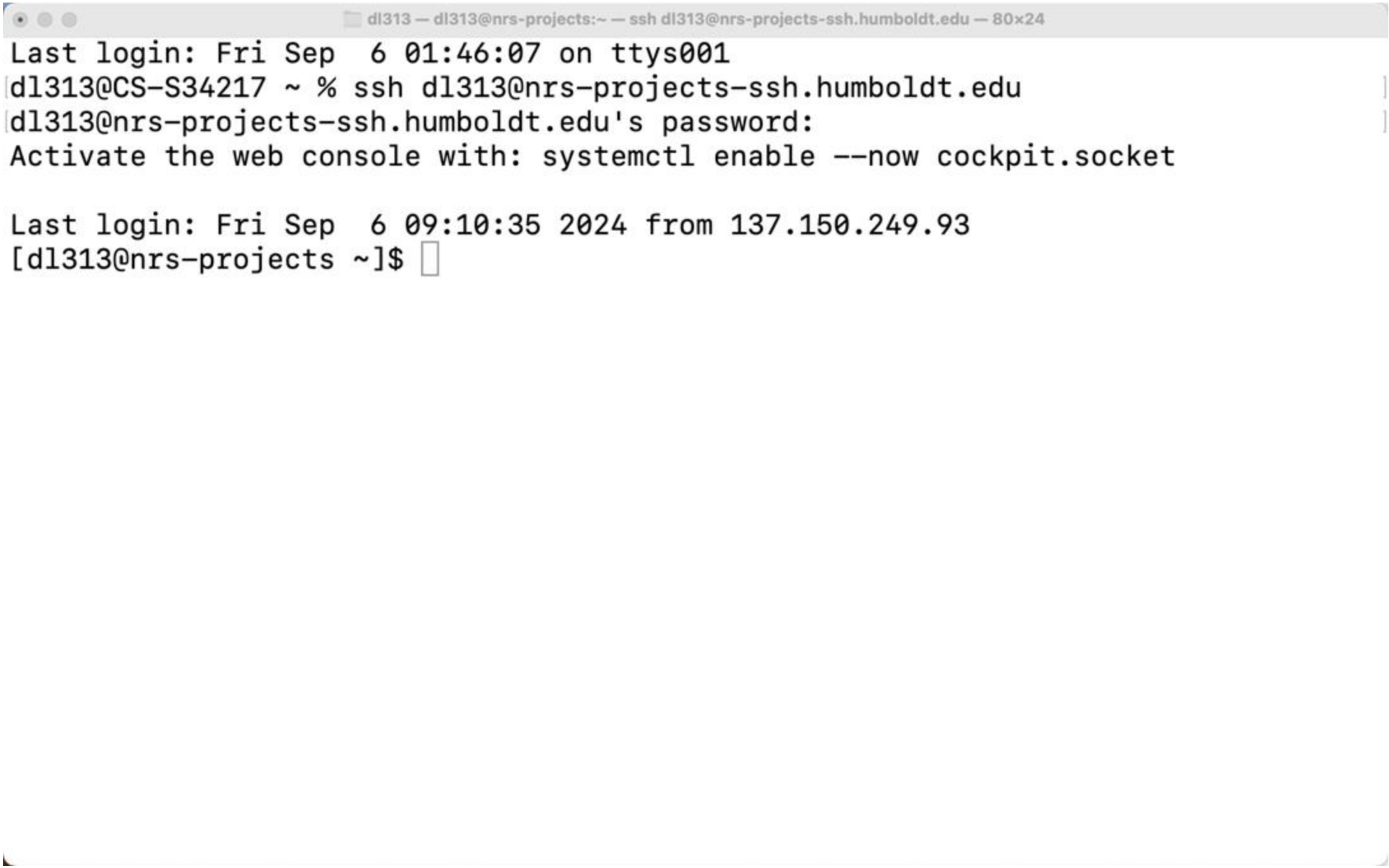
from Kroenke:

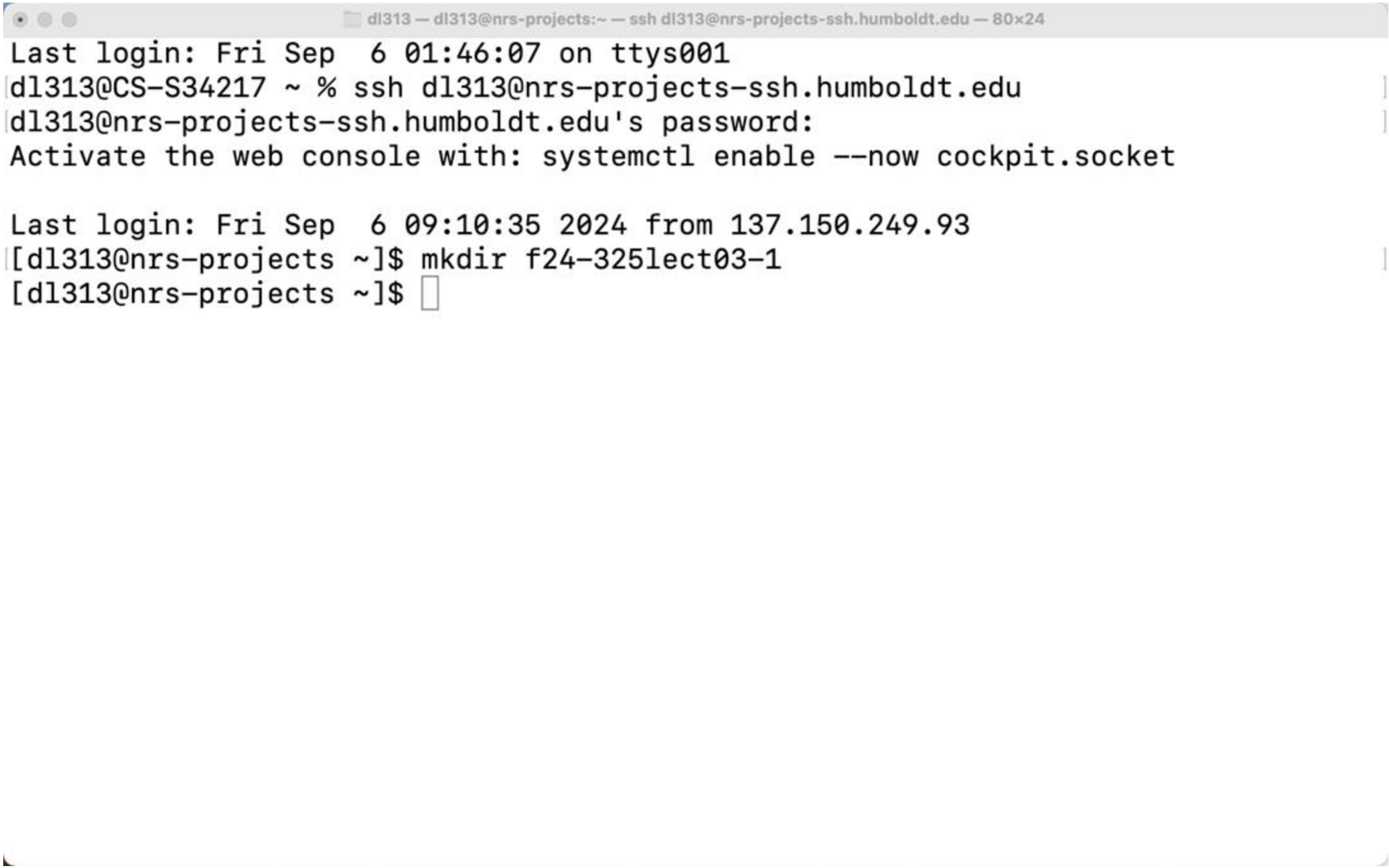
- "a database is a self-describing collection of integrated records“
- self-describing: means the database contains METADATA as well as data –
  - why would we want it to be self-describing?
    - ...for one thing, it can be used by a program to determine what a database contains;

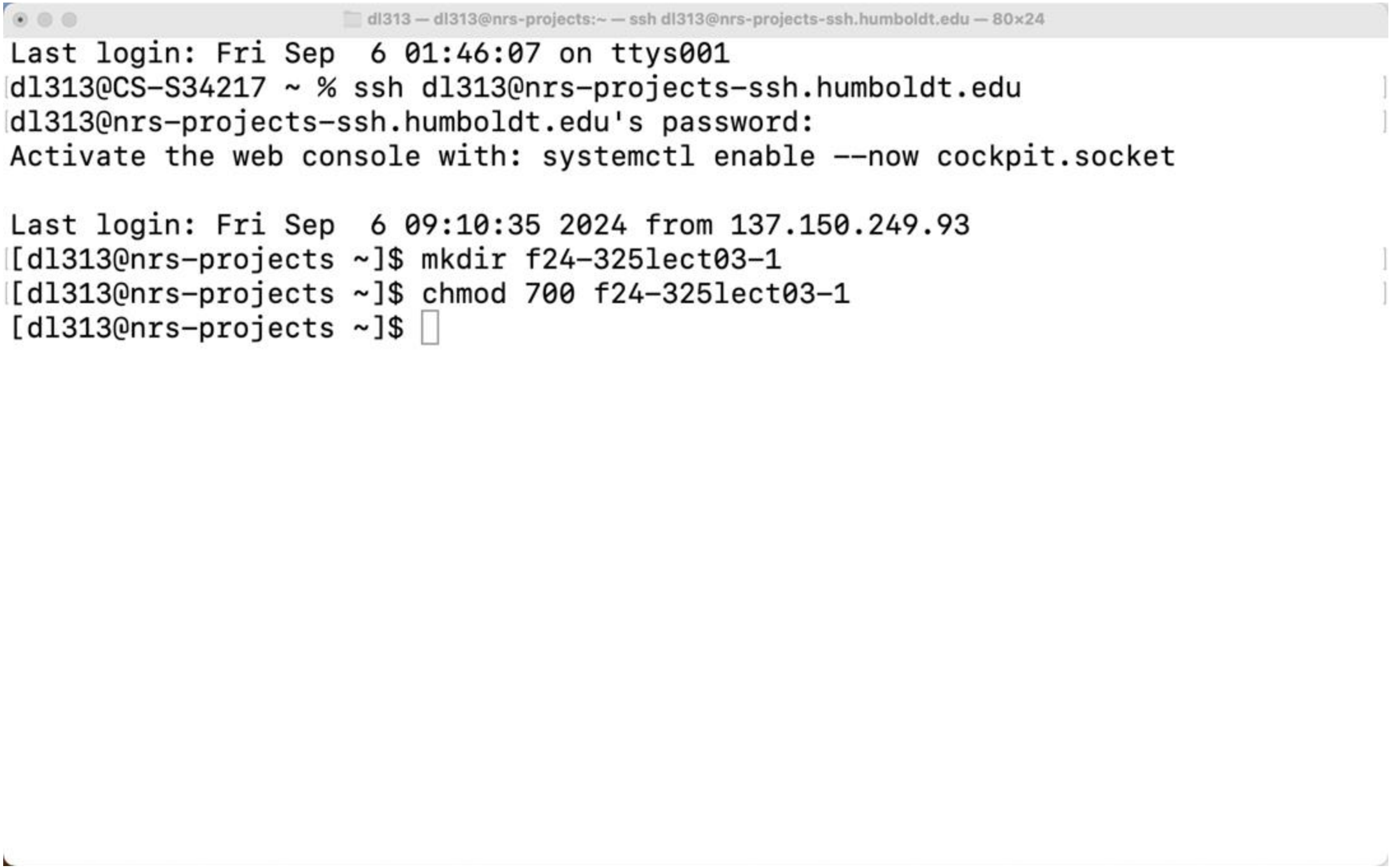
More database fundamentals

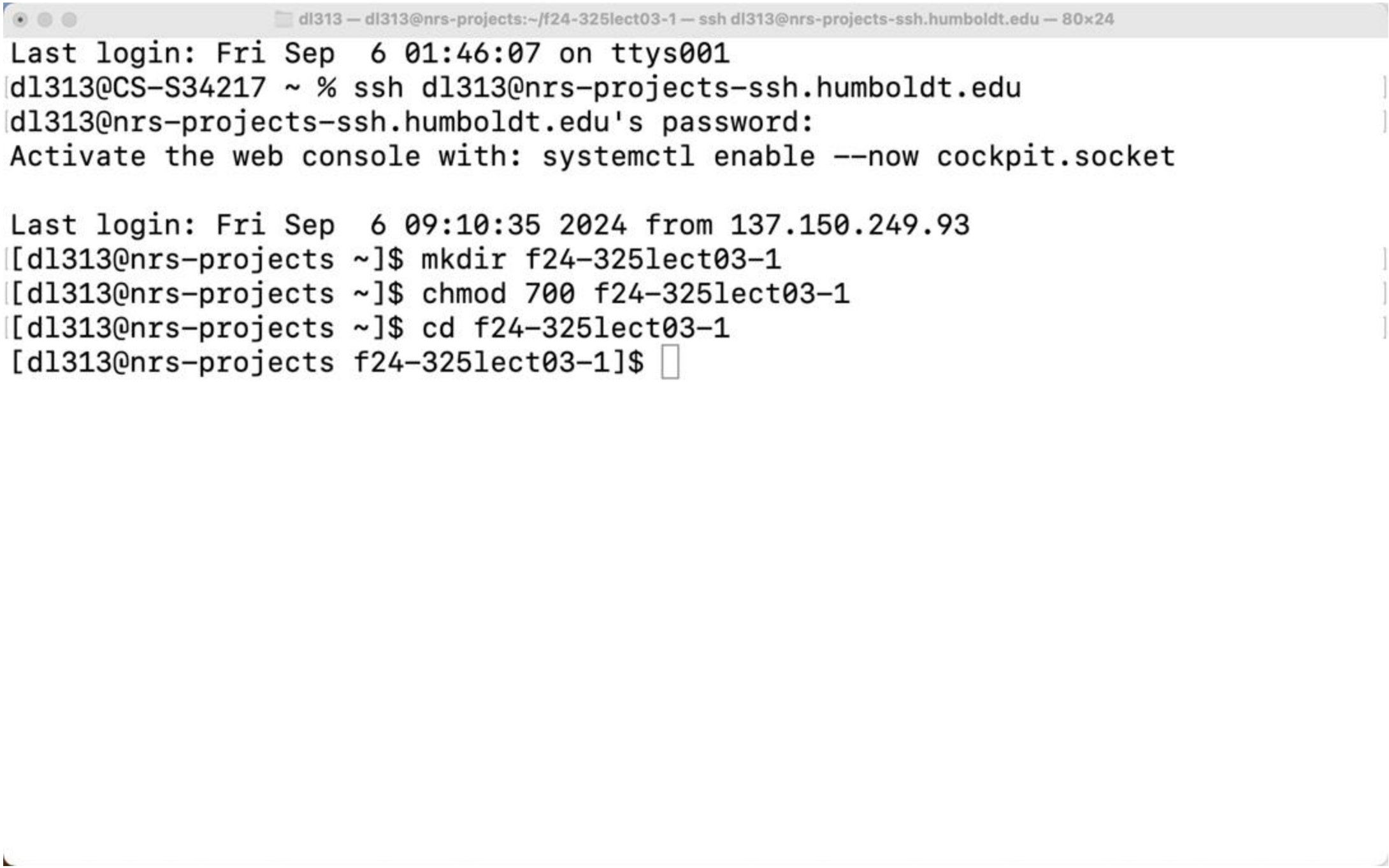
from Kroenke:

- "a database is a self-describing collection of integrated records“
- self-describing: means the database contains METADATA as well as data –
  - why would we want it to be self-describing?
    - ...for one thing, it can be used by a program to determine what a database contains;
    - ...that is, the self-describing aspect can help promote application program/data INdependence

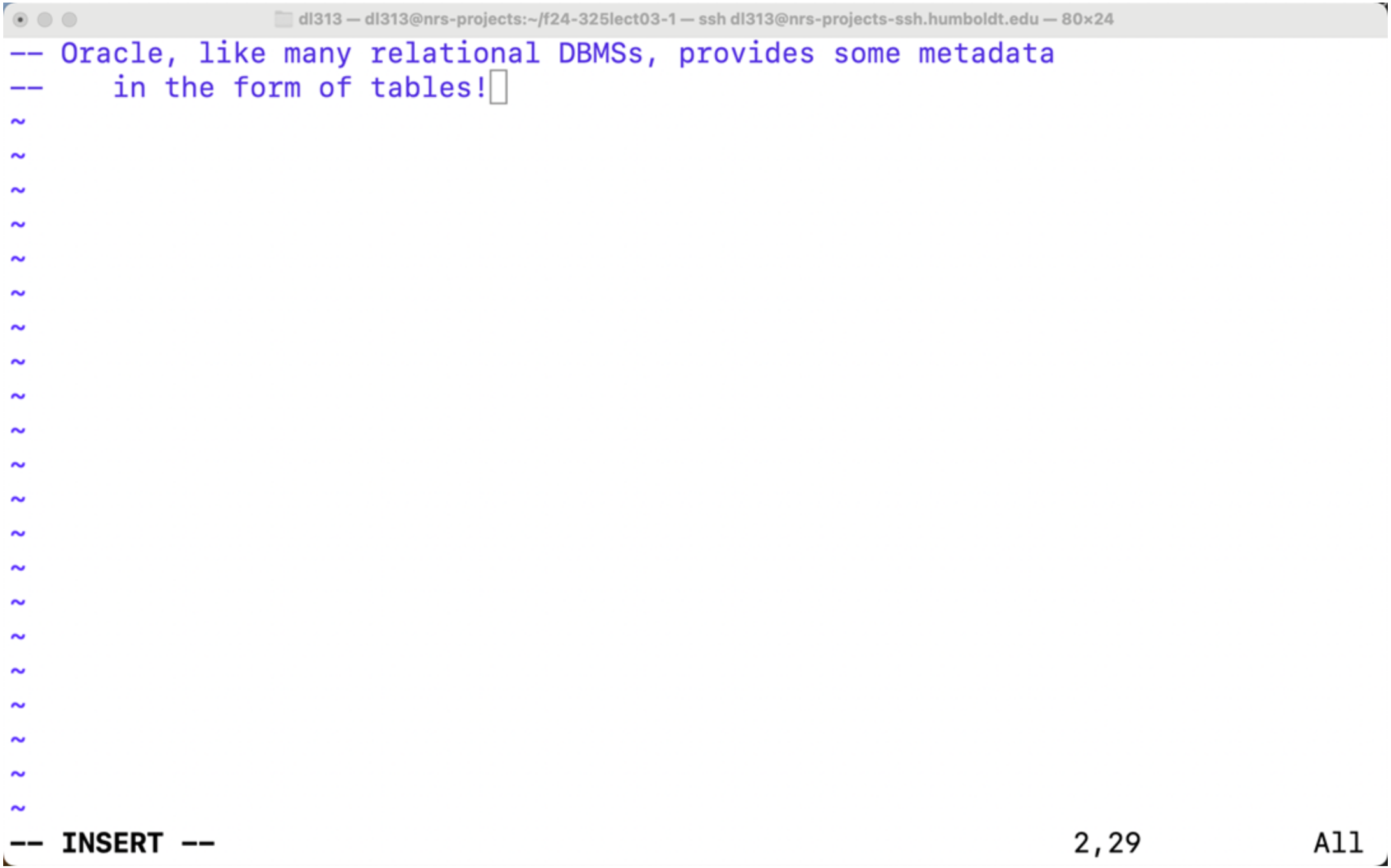

dl313 — dl313@nrs-projects:~ — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Fri Sep 6 01:46:07 on ttys001
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket
Last login: Fri Sep 6 09:10:35 2024 from 137.150.249.93
[dl313@nrs-projects ~]$

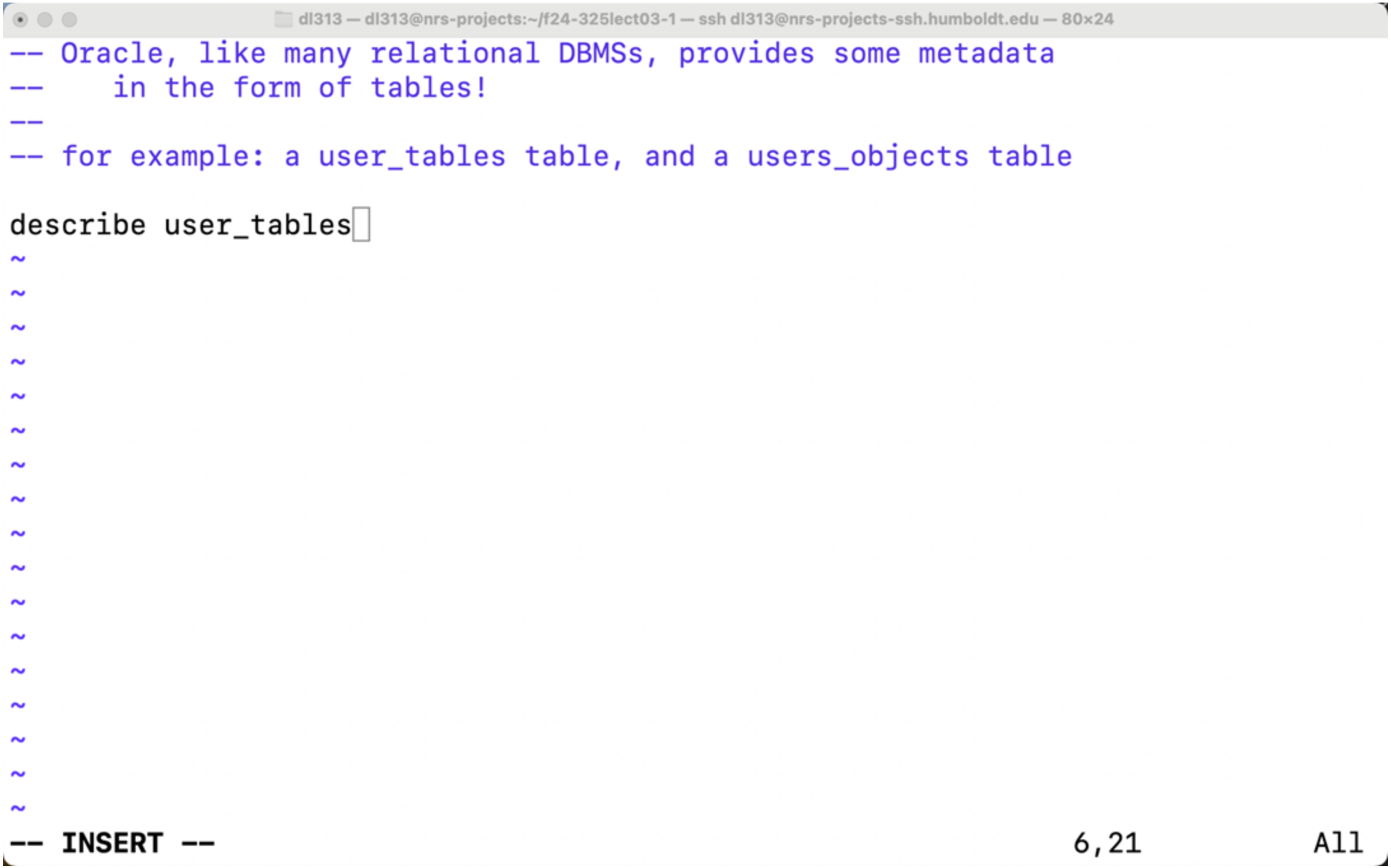

dl313 — dl313@nrs-projects:~ — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Fri Sep 6 01:46:07 on ttys001
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket
Last login: Fri Sep 6 09:10:35 2024 from 137.150.249.93
[dl313@nrs-projects ~]$ mkdir f24-325lect03-1
[dl313@nrs-projects ~]$


dl313 — dl313@nrs-projects:~ — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Fri Sep 6 01:46:07 on ttys001
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket
Last login: Fri Sep 6 09:10:35 2024 from 137.150.249.93
[dl313@nrs-projects ~]$ mkdir f24-325lect03-1
[dl313@nrs-projects ~]$ chmod 700 f24-325lect03-1
[dl313@nrs-projects ~]$


dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Fri Sep 6 01:46:07 on ttys001
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket
Last login: Fri Sep 6 09:10:35 2024 from 137.150.249.93
[dl313@nrs-projects ~]$ mkdir f24-325lect03-1
[dl313@nrs-projects ~]$ chmod 700 f24-325lect03-1
[dl313@nrs-projects ~]$ cd f24-325lect03-1
[dl313@nrs-projects f24-325lect03-1]$

```
dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Fri Sep  6 01:46:07 on ttys001
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Fri Sep  6 09:10:35 2024 from 137.150.249.93
[dl313@nrs-projects ~]$ mkdir f24-325lect03-1
[dl313@nrs-projects ~]$ chmod 700 f24-325lect03-1
[dl313@nrs-projects ~]$ cd f24-325lect03-1
[dl313@nrs-projects f24-325lect03-1]$ vim 325lect03-1.sql
```

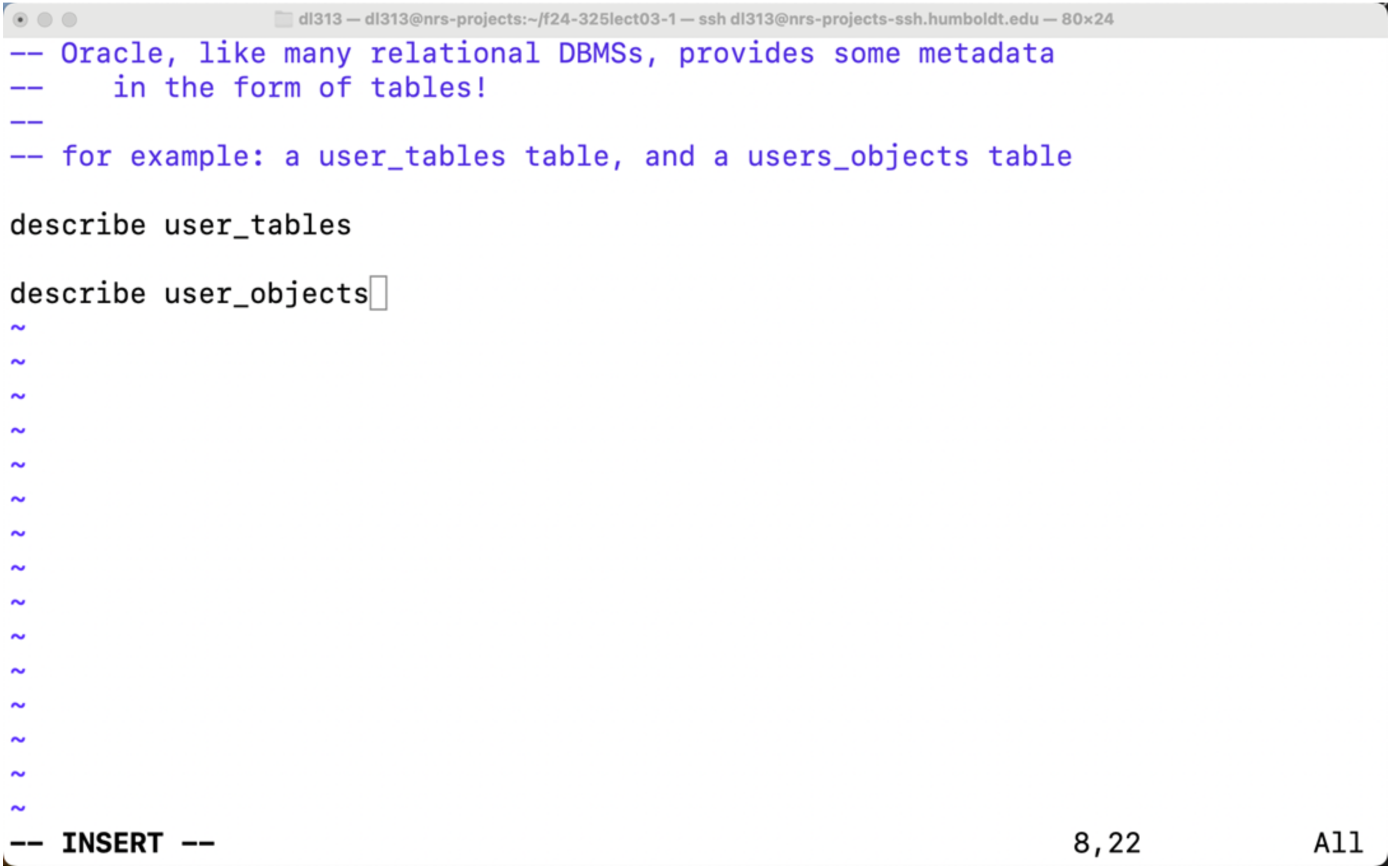

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
-- Oracle, like many relational DBMSs, provides some metadata
--   in the form of tables!
-- INSERT --
2,29
All


dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
-- Oracle, like many relational DBMSs, provides some metadata
--   in the form of tables!
--
-- for example: a user_tables table, and a users_objects table
describe user_tables
-- INSERT --
6,21
All

```
dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Sun Sep  8 21:10:42 on ttys000
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Sun Sep  8 21:11:11 2024 from 137.150.231.63
[dl313@nrs-projects ~]$ cd f24-325lect03-1
[dl313@nrs-projects f24-325lect03-1]$ sqlplus /

SQL*Plus: Release 19.0.0.0.0 - Production on Sun Sep 8 21:13:35 2024
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Last Successful login time: Fri Sep 06 2024 09:10:49 -07:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL> @ 325lect03-1.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
SQL> @ 325lect03-1.sql
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 TABLE_NAME                                NOT NULL VARCHAR2(128)
 TABLESPACE_NAME                                    VARCHAR2(30)
 CLUSTER_NAME                                       VARCHAR2(128)
 IOT_NAME                                           VARCHAR2(128)
 STATUS                                             VARCHAR2(8)
 PCT_FREE                                           NUMBER
 PCT_USED                                           NUMBER
 INI_TRANS                                          NUMBER
 MAX_TRANS                                          NUMBER
 INITIAL_EXTENT                                     NUMBER
 NEXT_EXTENT                                        NUMBER
 MIN_EXTENTS                                        NUMBER
 MAX_EXTENTS                                        NUMBER
 PCT_INCREASE                                       NUMBER
 FREELISTS                                          NUMBER
 FREELIST_GROUPS                                    NUMBER
 LOGGING                                            VARCHAR2(3)
 BACKED_UP                                          VARCHAR2(1)
 NUM_ROWS                                           NUMBER
 BLOCKS                                             NUMBER
 EMPTY_BLOCKS                                       NUMBER
```


dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
-- Oracle, like many relational DBMSs, provides some metadata
--   in the form of tables!
--
-- for example: a user_tables table, and a users_objects table
describe user_tables
describe user_objects
-- INSERT --
8,22
All

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
INMEMORY_DISTRIBUTE                            VARCHAR2(15)
INMEMORY_COMPRESSION                           VARCHAR2(17)
INMEMORY_DUPLICATE                             VARCHAR2(13)
DEFAULT_COLLATION                              VARCHAR2(100)
DUPLICATED                                     VARCHAR2(1)
SHARDED                                        VARCHAR2(1)
EXTERNAL                                       VARCHAR2(3)
HYBRID                                         VARCHAR2(3)
CELLMEMORY                                     VARCHAR2(24)
CONTAINERS_DEFAULT                             VARCHAR2(3)
CONTAINER_MAP                                  VARCHAR2(3)
EXTENDED_DATA_LINK                             VARCHAR2(3)
EXTENDED_DATA_LINK_MAP                         VARCHAR2(3)
INMEMORY_SERVICE                               VARCHAR2(12)
INMEMORY_SERVICE_NAME                          VARCHAR2(1000)
CONTAINER_MAP_OBJECT                           VARCHAR2(3)
MEMOPTIMIZE_READ                               VARCHAR2(8)
MEMOPTIMIZE_WRITE                              VARCHAR2(8)
HAS_SENSITIVE_COLUMN                           VARCHAR2(3)
ADMIT_NULL                                     VARCHAR2(3)
DATA_LINK_DML_ENABLED                          VARCHAR2(3)
LOGICAL_REPLICATION                            VARCHAR2(8)

SQL> @ 325lect03-1.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

Name	Null?	Type
OBJECT_NAME		VARCHAR2(128)
SUBOBJECT_NAME		VARCHAR2(128)
OBJECT_ID		NUMBER
DATA_OBJECT_ID		NUMBER
OBJECT_TYPE		VARCHAR2(23)
CREATED		DATE
LAST_DDL_TIME		DATE
TIMESTAMP		VARCHAR2(19)
STATUS		VARCHAR2(7)
TEMPORARY		VARCHAR2(1)
GENERATED		VARCHAR2(1)
SECONDARY		VARCHAR2(1)
NAMESPACE		NUMBER
EDITION_NAME		VARCHAR2(128)
SHARING		VARCHAR2(18)
EDITIONABLE		VARCHAR2(1)
ORACLE_MAINTAINED		VARCHAR2(1)
APPLICATION		VARCHAR2(1)
DEFAULT_COLLATION		VARCHAR2(100)
DUPLICATED		VARCHAR2(1)
SHARDED		VARCHAR2(1)
CREATED_APPID		NUMBER

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
-- Oracle, like many relational DBMSs, provides some metadata
--    in the form of tables!
--
-- for example: a user_tables table, and a users_objects table

describe user_tables

describe user_objects

-- another form of the SQL select statement lets you specify
--     just what columns you want to see from a table;
-- for example, here's how I can see JUST the table_name
--     column from the user_tables metadate table:
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                          13,51         All
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
-- Oracle, like many relational DBMSs, provides some metadata
--    in the form of tables!
--
-- for example: a user_tables table, and a users_objects table

describe user_tables

describe user_objects

-- another form of the SQL select statement lets you specify
--     just what columns you want to see from a table;
-- for example, here's how I can see JUST the table_name
--     column from the user_tables metadate table:

select table_name
from   user_tables;
~
~
~
~
~
~
~
-- INSERT --                                          16,20         All
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
DATA_OBJECT_ID                                    NUMBER
OBJECT_TYPE                                       VARCHAR2(23)
CREATED                                           DATE
LAST_DDL_TIME                                     DATE
TIMESTAMP                                         VARCHAR2(19)
STATUS                                            VARCHAR2(7)
TEMPORARY                                         VARCHAR2(1)
GENERATED                                         VARCHAR2(1)
SECONDARY                                         VARCHAR2(1)
NAMESPACE                                         NUMBER
EDITION_NAME                                      VARCHAR2(128)
SHARING                                           VARCHAR2(18)
EDITIONABLE                                       VARCHAR2(1)
ORACLE_MAINTAINED                                 VARCHAR2(1)
APPLICATION                                       VARCHAR2(1)
DEFAULT_COLLATION                                 VARCHAR2(100)
DUPLICATED                                        VARCHAR2(1)
SHARDED                                           VARCHAR2(1)
CREATED_APPID                                     NUMBER
CREATED_VSNID                                     NUMBER
MODIFIED_APPID                                    NUMBER
MODIFIED_VSNID                                    NUMBER

SQL> @ 325lect03-1.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
TABLE_NAME
------------------------------------------------------------------------------
CHOCOLATE
CLIENT
COMIC
COMIC_LOAN
CUSTOMER
DEPT
EMPL
MOVIE
MOVIE_CATEGORY
PARTS
PART_ORDERS

TABLE_NAME
------------------------------------------------------------------------------
RENTAL
STUDENT
STUFF
VIDEO

15 rows selected.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
--
-- for example: a user_tables table, and a users_objects table

describe user_tables

describe user_objects

-- another form of the SQL select statement lets you specify
--     just what columns you want to see from a table;
-- for example, here's how I can see JUST the table_name
--     column from the user_tables metadate table:

select table_name
from   user_tables;

-- and here's how I can see just the object_name and object_type
--     columns from the user_objects metadata table:

~
~
~
~
-- INSERT --                                          21,1          Bot
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
--
-- for example: a user_tables table, and a users_objects table

describe user_tables

describe user_objects

-- another form of the SQL select statement lets you specify
--     just what columns you want to see from a table;
-- for example, here's how I can see JUST the table_name
--     column from the user_tables metadate table:

select table_name
from   user_tables;

-- and here's how I can see just the object_name and object_type
--     columns from the user_objects metadata table:

select object_name, object_type
from   user_objects;
~
~
~
-- INSERT --                                          22,21         Bot
```

```
dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
[dl313@nrs-projects f24-325lect03-1]$ sqlplus /

SQL*Plus: Release 19.0.0.0.0 - Production on Sun Sep 8 21:25:35 2024
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle.  All rights reserved.

Last Successful login time: Sun Sep 08 2024 21:13:35 -07:00

Connected to:
Oracle Database 19c Enterprise Edition Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL> @ 325lect03-1.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect03-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
OBJECT_NAME
--------------------------------------------------------------------------------
OBJECT_TYPE
-----------------------
CATEGORY_STATS
VIEW

CHOCOLATE
TABLE

CLIENT
TABLE


OBJECT_NAME
--------------------------------------------------------------------------------
OBJECT_TYPE
-----------------------
COMIC
TABLE

COMIC_LOAN
TABLE
```

Question (3-1-Q1):

What does the term **"self-describing"** mean in the context of a database?

1) The database only stores data without any additional information.
2) The database includes metadata, which is information about the data and its structure.
3) The database cannot describe its own structure.
4) The database stores only user-entered data and not system information.

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- self-describing: means the database contains METADATA as well as data.
- Integrated: you not only have data that's self-describing, you also have a description of RELATIONSHIPS among the data records;

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- Integrated: you not only have data that's self-describing, you also have a description of RELATIONSHIPS among the data records;
  - foreign keys are one way to describe such relationships!

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- NOTE that a database is a model, not of reality, but of the user's model or view of what's important in some scenario;

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- NOTE that a database is a model, not of reality, but of the user's model or view of what's important in some scenario;
  - model - "simplified abstraction of real-world events or conditions"

More database fundamentals

from Kroenke:

- "a database is a self-describing collection of integrated records“
- NOTE that a database is a model, not of reality, but of the user's model or view of what's important in some scenario;
  - model - "simplified abstraction of real-world events or conditions“
  - the degree of detail provided by a database should be based on the needs and desires of the users involved in the scenario that database is modeling or representing;

More database fundamentals

a LITTLE more database history:

More database fundamentals

- a LITTLE more database history:
- those early 1960's databases tended to be "organizational" in scope -- large companies, an important subset of such a large company;

More database fundamentals

- a LITTLE more database history:
- those early 1960's databases tended to be "organizational" in scope -- large companies, an important subset of such a large company;
  - and these often were VERY oriented towards TRANSACTIONS - it can be as simple as a representation of a class of significant events

More database fundamentals

- those early 1960's databases tended to be "organizational" in scope -- large companies, an important subset of such a large company;
  - and these often were VERY oriented towards TRANSACTIONS - it can be as simple as a representation of a class of significant events
  - these early organizational databases were VERY good at keeping track of regular (predicatable, regularly-occurring) transactions

More database fundamentals

- those early 1960's databases tended to be "organizational" in scope -- large companies, an important subset of such a large company;
  - and these often were VERY oriented towards TRANSACTIONS - it can be as simple as a representation of a class of significant events
  - these early organizational databases were VERY good at keeping track of regular (predicatable, regularly-occurring) transactions
  - also good at creating regularly scheduled reports;

More database fundamentals

- those early 1960's databases tended to be "organizational" in scope -- large companies, an important subset of such a large company;
  - also good at creating regularly scheduled reports;
    - BUT! they were not very flexible,

More database fundamentals

- those early 1960's databases tended to be "organizational" in scope -- large companies, an important subset of such a large company;
  - also good at creating regularly scheduled reports;
    - BUT! they were not very flexible,
    - and application programs tended to need to be written in procedural languages such as COBOL and PL/I
    - ...users might think of a "new" question the data COULD conceivably answer -- but they might not be able or willing to wait for a programmer to get around to writing a procedural program to answer their questions;

Question (3-1-Q2):

Why were early databases not very flexible?

1) They were designed for small-scale, personal use
2) They could not handle scheduled tasks
3) Users needed custom programs to analyze data in new ways, which took time
4) They had built-in flexibility to handle any new queries instantly

More database fundamentals

- ...enter THE RELATIONAL MODEL!

More database fundamentals

- ...enter THE RELATIONAL MODEL!
- E.F. Codd developed this in 1970;
  - It is based on relational algebra

More database fundamentals

- ...enter THE RELATIONAL MODEL!
- E.F. Codd developed this in 1970;
  - It is based on relational algebra
  - instead of organizing a database using a hierarchical or network structure, this considers data to be stored in the form of RELATIONS, table-like structures that meet certain mathematical requirements;

More database fundamentals

- ...enter THE RELATIONAL MODEL!
- E.F. Codd developed this in 1970;
  - It is based on relational algebra
  - instead of organizing a database using a hierarchical or network structure, this considers data to be stored in the form of RELATIONS, table-like structures that meet certain mathematical requirements;
  - relations have rows, and relations have columns, and relationship between rows of tables are visible in the data

More database fundamentals

- ...enter THE RELATIONAL MODEL!
- E.F. Codd developed this in 1970;
  - it turns out that relational algebra's relation operations provide a very useful way to express operations on relational tables that can map very nicely to QUESTIONS user might want to ask about the data;

More database fundamentals

- ...enter THE RELATIONAL MODEL!
- E.F. Codd developed this in 1970;
  - it turns out that relational algebra's relation operations provide a very useful way to express operations on relational tables that can map very nicely to QUESTIONS user might want to ask about the data;
  - and asking and answering questions over time remains very reasonable! And you can do so with relatively short SQL select statements (compared to procuedural programs written in COBOL or PL/I or C++ or Java or JavaScript or Python, etc.

More database fundamentals

- Potential benefits of Codd's relational model:

More database fundamentals

- Potential benefits of Codd's relational model:
  - in a well-deigned relational database, data are stored in a way that minimizes unnecessarily duplicated data and eliminates certain kinds of day-to-day processing errors that can occur compared to some other data-storage approaches;

More database fundamentals

- Potential benefits of Codd's relational model:
  - in a well-deigned relational database, data are stored in a way that minimizes unnecessarily duplicated data and eliminates certain kinds of day-to-day processing errors that can occur compared to some other data-storage approaches;
  - columns can be used to contain data that relate one row to another row
  - it is easier for people to think about relational/tabular data than that is earlier hierarchical/network database approaches;

More database fundamentals

- Potential benefits of Codd's relational model:
  - …
  - It provide much better support for ad-hoc queries, spur-of-the-moment questions over time and can encourage more-creative use of one's data;

More database fundamentals

- There was initial resistance to the relational model (and there is a little now, also)

More database fundamentals

- There was initial resistance to the relational model (and there is a little now, also)
  - there IS overhead here! providing this relational abstraction and performing these relational operations;

More database fundamentals

- there was initial resistance to the relational model (and there is a little now, also)
  - there IS overhead here! providing this relational abstraction and performing these relational operations;
  - fortunately, computer hardware and memory speed and power increased, and prices for these decreased, to where relational implementations could be practical;

Question (3-1-Q3):

What is a key characteristic of data organization in the relational model?

1) Data is stored in a hierarchy, with parent-child relationships between records.
2) Data is stored in tables (relations), with rows and columns, and relationships between rows are managed through foreign keys.
3) Data is organized in a network, with multiple parent and child nodes.
4) Data is stored sequentially, with no relationships between different tables.

More database fundamentals

- 1970s: Relational Model (E.F. Codd)
- Microcomputers (e.g., Apple II, Commodore 64) made computers accessible to more people.

More database fundamentals

- 1970s: Relational Model (E.F. Codd)
- Microcomputers (e.g., Apple II, Commodore 64) made computers accessible to more people.
- Early Personal Databases.

More database fundamentals

- 1970s: Relational Model (E.F. Codd)
- Microcomputers (e.g., Apple II, Commodore 64) made computers accessible to more people.
- Early Personal Databases.
- 1980s: Client-Server Architecture

More database fundamentals

- 1970s: Relational Model (E.F. Codd)
- Microcomputers (e.g., Apple II, Commodore 64) made computers accessible to more people.
- Early Personal Databases.
- 1980s: Client-Server Architecture
- 1990s: Internet and Distributed Databases

More database fundamentals

- 1970s: Relational Model (E.F. Codd)
- Microcomputers (e.g., Apple II, Commodore 64) made computers accessible to more people.
- Early Personal Databases.
- 1980s: Client-Server Architecture
- 1990s: Internet and Distributed Databases
- Object-Oriented Databases

More database fundamentals

- 1970s: Relational Model (E.F. Codd)
- Microcomputers (e.g., Apple II, Commodore 64) made computers accessible to more people.
- Early Personal Databases.
- 1980s: Client-Server Architecture
- 1990s: Internet and Distributed Databases
- Object-Oriented Databases
- Modern Era: NoSQL Databases

More database fundamentals

- Four main elements of a (relational) database:
  - user data
  - metadata
  - indexes of various kinds - especially foreign keys
  - application metadata

More database fundamentals

- Four main elements of a (relational) database:
  - user data - the most obvious element!
  - in a relational database, user data is represented as relational tables;

More database fundamentals

- Four main elements of a (relational) database:
  - user data - the most obvious element!
  - in a relational database, user data is represented as relational tables;
  - we can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:

More database fundamentals

- Four main elements of a (relational) database:
  - user data - the most obvious element!
  - in a relational database, user data is represented as relational tables;
  - we can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
    - tabular: actually list a table's rows and columns!

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
    - tabular: actually list a table's rows and columns!

STUDENT:

Stu_id	Stu_lname	Stu_phone	Adviser_id
123456	Jones	123-4567	234567
123457	Nguyen	234-5678	234568
123458	Garza	345-6789	234569

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - tabular: actually list a table's rows and columns!
    - straightforward!
    - can see contents!
    - you can see the basic table structure,

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - tabular: actually list a table's rows and columns!
    - straightforward! can see contents! you can see the basic table structure,
  - But
    - you can infer domain information for column, but you don't know its physical domain

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - tabular: actually list a table's rows and columns!
    - straightforward! can see contents! you can see the basic table structure,
  - But
    - you can infer domain information for column, but you don't know its physical domain
    - you might be able to infer how tables are related, but not explicit;
    - can be a pain to type;

Question (3-1-Q4):

Which of the following is **NOT** one of the limitations of the **tabular form** in relational databases?

A) It does not explicitly show relationships between tables

B) It does not display the data types of columns

C) It does not allow for easy navigation of data (actual rows of data)

D) It can be tedious to manually enter large amounts of data

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - tabular: actually list a table's rows and columns!
  - there are times when you care more about the table structure and less about the current rows in a table - relation structure form can be useful!

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - tabular: actually list a table's rows and columns!
  - there are times when you care more about the table structure and less about the current rows in a table - relation structure form can be useful!

```
name_of_relation(PRIM_KEY_COL1, PRIM_KEY_COL2, another_col2,
    another_col2, ...)
    foreign key (another_col2) references another_relation
```

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - tabular: actually list a table's rows and columns!
  - there are times when you care more about the table structure and less about the current rows in a table - relation structure form can be useful!

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id)
```

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - there are times when you care more about the table structure and less about the current rows in a table - relation structure form can be useful!
    - still pretty straightforward; nice and concise; pretty easy to type; CAN see the primary key; can often see the foreign key.

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - there are times when you care more about the table structure and less about the current rows in a table - relation structure form can be useful!
    - still pretty straightforward; nice and concise; pretty easy to type; CAN see the primary key; can often see the foreign key
  - But
    - don't see the rows in this form
    - don't see information about the column domains!

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - create-table-statement form:

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - create-table-statement form:
    - represent a relation as the SQL create table statement you could use to create that table or relation!

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - create-table-statement form:
    - represent a relation as the SQL create table statement you could use to create that table or relation!

```
create table student
(stu_id         char(6),
 stu_lname      varchar2(30),
 stu_phone      char(8),
 adviser_id     char(6),
 primary key    (student_id),
 foreign key    (adviser_id) references adviser
);
```

More database fundamentals

- We can depict these tables and/or their structure in quite a few ways -- here are THREE such ways:
  - create-table-statement form: represent a relation as the SQL create table statement you could use to create that table or relation!
    - definitely shows the columns; gives SOME physical domain information about each column; OUGHT to give primary key and foreign keys explicitly; can even create actual tables from them!
  - But
    - don't see rows in this form; this is a LITTLE more a pain to type...
    - maybe a little less easy for people to read than relation structure form...

Question (3-1-Q5):

Consider the different ways of depicting a relation that we discussed. Which **most explicitly** shows the **data type/physical domain** of the elements within a particular column?

1) tabular form
2) relation structure form
3) create-table statement form
4) metadata form

More database fundamentals

- Relational Databases Have Potential: However, poor table structure can waste this potential.
- An example of poorly-structures vs. "better"-structured relations.

More database fundamentals

- Relational Databases Have Potential: However, poor table structure can waste this potential.
- An example of poorly-structures vs. "better"-structured relations.
  - The quality of structure depends on what you want to achieve and how you use the data.

More database fundamentals

- Relational Databases Have Potential: However, poor table structure can waste this potential.
- An example of poorly-structures vs. "better"-structured relations.
  - The quality of structure depends on what you want to achieve and how you use the data.
  - Operational vs. Read-Only Databases.
  - The best structure depends on the context.

More database fundamentals

- An example of poorly-structures vs. "better"-structured relations;
  - because some database structures will be harder to live with over time, or will be easier to maintain data integrity over time, or will make representing some information easier over time, etc.

More database fundamentals

- An example of poorly-structures vs. "better"-structured relations;

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id,
        Adviser_lname, Adviser_phone)
```

More database fundamentals

- An example of poorly-structures vs. "better"-structured relations;

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id,
        Adviser_lname, Adviser_phone)
```

vs.

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id)
    foreign key (Adviser_id) references Adviser

Adviser(ADVISER_ID, Adviser_lname, Adviser_phone)
```

More database fundamentals

- First Approach (Single Table)
  - Data duplication (e.g., repeating advisor details for each student).
  - Updates (e.g., phone number changes) must be made in multiple places, increasing risk of errors.

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id,
        Adviser_lname, Adviser_phone)
```

More database fundamentals

- First Approach (Single Table)
  - Data duplication (e.g., repeating advisor details for each student).
  - Updates (e.g., phone number changes) must be made in multiple places, increasing risk of errors.
  - May have quick access to advisor information from the student table

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id,
        Adviser_lname, Adviser_phone)
```

More database fundamentals

- Second Approach (Two Related Tables)
  - Advisor information stored once, avoiding duplication and ensuring data integrity.

```
Student(STU_ID, Stu_lname, Stu_phone, Adviser_id)
    foreign key (Adviser_id) references Adviser

Adviser(ADVISER_ID, Adviser_lname, Adviser_phone)
```

More database fundamentals

- First Approach (Single Table)
  - Data duplication (e.g., repeating advisor details for each student).
  - Updates (e.g., phone number changes) must be made in multiple places, increasing risk of errors.
  - May have quick access to advisor information from the student table
  - Advisor information is lost if all students are deleted.
- Second Approach (Two Related Tables)
  - Advisor information stored once, avoiding duplication and ensuring data integrity.
  - Requires an extra step to link student and advisor data.

More database fundamentals

- First Approach (Single Table)
  - Data duplication (e.g., repeating advisor details for each student).
  - Updates (e.g., phone number changes) must be made in multiple places, increasing risk of errors.
  - May have quick access to advisor information from the student table
  - Advisor information is lost if all students are deleted.
- Second Approach (Two Related Tables)
  - Advisor information stored once, avoiding duplication and ensuring data integrity.
  - Requires an extra step to link student and advisor data.
  - Better for long-term data management in day-to-day operations.

Question (3-1-Q6)

What is a major **disadvantage** of storing both student and advisor information in a **single table**?

A) It requires more tables to be created.

B) It can result in unnecessary duplication of advisor information, leading to potential data inconsistency.

C) It makes retrieving advisor information impossible.

More database fundamentals

- going BEYOND the DB Reading Packet 2 a little bit:

More database fundamentals

- going BEYOND the DB Reading Packet 2 a little bit:
    - a bit more about some capabilities a DBMS should provide
    - -- these can vary a lot, BUT:
    - if it is going to provide an interface between users/applications and the data,
    - abstracting out physical storage details,
    - then at the very least, it needs to provide:

More database fundamentals

- going BEYOND the DB Reading Packet 2 a little bit:
  - a bit more about some capabilities a DBMS should provide, these can vary a lot, BUT: if it is going to provide an interface between users/applications and the data, abstracting out physical storage details, then at the very least, it needs to provide:
    - some kind of data DEFINITION language (DDL)

More database fundamentals

- going BEYOND the DB Reading Packet 2 a little bit:
  - a bit more about some capabilities a DBMS should provide, these can vary a lot, BUT: if it is going to provide an interface between users/applications and the data, abstracting out physical storage details, then at the very least, it needs to provide:
    - some kind of data DEFINITION language (DDL)
    - some kind of data MANIPULATION language (DML)

More database fundamentals

- going BEYOND the DB Reading Packet 2 a little bit:
  - a bit more about some capabilities a DBMS should provide, these can vary a lot, BUT: if it is going to provide an interface between users/applications and the data, abstracting out physical storage details, then at the very least, it needs to provide:
    - some kind of data DEFINITION language (DDL)
    - some kind of data MANIPULATION language (DML)
    - some kind of data CONTROL language (DCL)

More database fundamentals

- going BEYOND the DB Reading Packet 2 a little bit:
  - a bit more about some capabilities a DBMS should provide
    - some kind of data DEFINITION language (DDL)
    - some kind of data MANIPULATION language (DML)
    - some kind of data CONTROL language (DCL)
  - many DBMSs implement these by providing an implementation of SQL! (some also provide various GRAPHICAL DDL, DML and/or DCL)

More database fundamentals

- many DBMSs implement these by providing an implementation of SQL! (some also provide various GRAPHICAL DDL, DML and/or DCL)
  - DDL - let users define the database!
    - for a relational database, this lets you define tables!
    - e.g. SQL's create table statement

More database fundamentals

- DDL - let users define the database!
  - for a relational database, this lets you define tables!
  - e.g. SQL's create table statement
- DML - lets users insert, update, delete, and retrieve data
  - e.g. SQL's insert statement
  - and its update statement
  - and its delete statement
  - and its select statement

More database fundamentals

- DDL - let users define the database!
  - e.g. SQL's create table statement
- DML - lets users insert, update, delete, and retrieve data
  - e.g. SQL's insert statement
- DCL - lets users control access to the database

More database fundamentals

- DDL - let users define the database!
  - e.g. SQL's create table statement
- DML - lets users insert, update, delete, and retrieve data
  - e.g. SQL's insert statement
- DCL - lets users control access to the database
  - security features, data integrity features, concurrency control, recovery system, and more

More database fundamentals

- DDL - let users define the database!
  - e.g. SQL's create table statement
- DML - lets users insert, update, delete, and retrieve data
  - e.g. SQL's insert statement
- DCL - lets users control access to the database
  - security features, data integrity features, concurrency control, recovery system, and more
  - SQL provides *some* DCL support - creation of views, ability to GRANT or REVOKE privileges on tables

More database fundamentals

- DCL - lets users control access to the database
  - security features, data integrity features, concurrency control, recovery system, and more
  - SQL provides *some* DCL support - creation of views, ability to GRANT or REVOKE privileges on tables
    - GRANT lets you say particular users can be GRANTed select, insert, update, and/or delete access to a particular table

More database fundamentals

- DCL - lets users control access to the database
  - security features, data integrity features, concurrency control, recovery system, and more
  - SQL provides *some* DCL support - creation of views, ability to GRANT or REVOKE privileges on tables
    - GRANT lets you say particular users can be GRANTed select, insert, update, and/or delete access to a particular table

```
GRANT select, insert, update
ON abc123.stuff
TO fg3, gh5, st10;
```

    - -- says users fg3, gh5, st10 can select, insert, and update, -- (but NOT delete) rows from user abc123's stuff table

More database fundamentals

- DCL - lets users control access to the database
  - security features, data integrity features, concurrency control, recovery system, and more
  - SQL provides *some* DCL support - creation of views, ability to GRANT or REVOKE privileges on tables
    - GRANT lets you say particular users can be GRANTed select, insert, update, and/or delete access to a particular table.
    - REVOKE lets you remove previously-GRANTed access

```
REVOKE select
ON abc124.stuff
FROM gh5;
```

    - -- says user gh5 can NO LONGER select rows from user abc123's -- stuff table (but they can still insert rows and update -- them...!)

Question (3-1-Q7):

Consider the SQL create table statement. Which DBMS capability is this most reasonably considered to be part of?

1) its DDL, data definition language
2) its DML, data manipulation language
3) its DCL, data control language
4) its MDL, metadata language