CS 325 Week 4-1

Intro to the relational model (DB Reading Packet 3)

Intro to the RELATIONAL model

- Some KEY definitions

Intro to the RELATIONAL model

- Some KEY definitions
- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation

Intro to the RELATIONAL model

- Some KEY definitions
- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - E.g., advisor = (advisor_id, advisor_lname, advisor_fname, advisor_phone)

Intro to the RELATIONAL model

- Some KEY definitions
- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - E.g., advisor = (advisor_id, advisor_lname, advisor_fname, advisor_phone)
  - (it MUST have at least ONE such SET of attributes!)

Intro to the RELATIONAL model

- Some KEY definitions
- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - E.g., advisor = (advisor_id, advisor_lname, advisor_fname, advisor_phone)
  - (it MUST have at least ONE such SET of attributes!)
  - consider this relation:
  - course_stats = (stu_id, class_id, class_grade, completion_date)
  - So, what may be our superkey in this relation?

Intro to the RELATIONAL model

- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - (it MUST have at least ONE such SET of attributes!)
  - consider this relation:
  - course_stats = (stu_id, class_id, class_grade, completion_date)
  - it is the case that (stu_id, class_id) is a superkey of course_stats
  - it is also the case the (stu_id, class_id, class_grade) is a superkey of course_stats

Intro to the RELATIONAL model

- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - (it MUST have at least ONE such SET of attributes!)
  - consider this relation:
  - course_stats = (stu_id, class_id, class_grade, completion_date)
  - it is the case that (stu_id, class_id) is a superkey of course_stats
  - it is also the case the (stu_id, class_id, class_grade) is a superkey of course_stats
  - Is stu_id alone a superkey? Is Class_id alone a superkey?

Intro to the RELATIONAL model

- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - (it MUST have at least ONE such SET of attributes!)
  - consider this relation:
  - course_stats = (stu_id, class_id, class_grade, completion_date)
  - it is the case that (stu_id, class_id) is a superkey of course_stats, it is also the case the (stu_id, class_id, class_grade) is a superkey of course_stats
  - Is the combination of class_id and class_grade a superkey? Is the combination of class_id and completion_date a superkey?

Intro to the RELATIONAL model

- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - (it MUST have at least ONE such SET of attributes!)
- PLEASE NOTE: the functional dependencies and superkeys of relations representing some setting or scenario are determined by that scenario, and the users' model of that scenario;

Intro to the RELATIONAL model

- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
  - a relation can have multiple superkeys!
  - (it MUST have at least ONE such SET of attributes!)
- PLEASE NOTE: the functional dependencies and superkeys of relations representing some setting or scenario are determined by that scenario, and the users' model of that scenario;
- And business rules - day-to-day rules in some setting or scenario, and such business rules can affect functional dependencies and superkeys;

Intro to the RELATIONAL model

- PLEASE NOTE: the functional dependencies and superkeys of relations representing some setting or scenario are determined by that scenario, and the users' model of that scenario;
- And business rules - day-to-day rules in some setting or scenario, and such business rules can affect functional dependencies and superkeys;
- So if you have a question about whether some relationship between attributes IS a functional dependency -- you should ask the users!!!

Question

Consider the relation:

dept(dept_name, DEPT_NUM, dept_loc).

Note: in this scenario, some departments share an office.

Is the attribute-set (dept_loc) also a **superkey** for this relation?

1) Yes
2) No

Intro to the RELATIONAL model

- Some KEY definitions
- A SUPERKEY or LOGICAL KEY is any set of one or more attributes that uniquely determine a tuple in a relation
- e.g., course_stats = (stu_id, class_id, class_grade, completion_date)
- SO -- a superkey may be a bit "more" than is needed to determine a tuple in a relation;

Intro to the RELATIONAL model

- Some KEY definitions
- e.g., course_stats = (stu_id, class_id, class_grade, completion_date)
- SO -- a superkey may be a bit "more" than is needed to determine a tuple in a relation;
- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey

Intro to the RELATIONAL model

- SO -- a superkey may be a bit "more" than is needed to determine a tuple in a relation;
- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - Consider relation advisor = (advisor_id, advisor_lname, advisor_fname)

Intro to the RELATIONAL model

- Another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - Consider relation advisor = (advisor_id, advisor_lname, advisor_fname)
  - for advisor
  - (advisor_id, advisor_lname, advisor_fname) is a superkey but not a minimal key, because subset (advisor_id) is also a superkey
  - BUT (advisor_id) IS a minimal key, because no subset of it is ALSO a superkey.

Intro to the RELATIONAL model

- Another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - Consider relation advisor = (advisor_id, advisor_lname, advisor_fname)
  - for course_stats,
  - (stu_id, class_id, class_grade) is a superkey but not a minimal key, because subset (stu_id, class_id) is also a superkey BUT (stu_id, class_id) IS a minimal key, because no subset of it is ALSO a superkey.

Intro to the RELATIONAL model

- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - a minimal key is also called a CANDIDATE key

Intro to the RELATIONAL model

- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - a minimal key is also called a CANDIDATE key
  - well...
  - a relation CAN have multiple minimal/candidate keys...!
  - student = (stu_id, stu_ssn, stu_lname, stu_fname, stu_email)
  - ...then (stu_id) is a minimal key
  - but (stu_ssn) is also a minimal key

Intro to the RELATIONAL model

- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - a minimal key is also called a CANDIDATE key
  - the primary key for a relation is the minimal/candidate key that we CHOOSE (from amongst its possible minimal/candidate keys) to serve as its primary key;

Intro to the RELATIONAL model

- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - a minimal key is also called a CANDIDATE key
  - the primary key for a relation is the minimal/candidate key that we CHOOSE (from amongst its possible minimal/candidate keys) to serve as its primary key;
  - student = (stu_id, stu_ssn, stu_lname, stu_fname, stu_email)

Intro to the RELATIONAL model

- another definition is then:
  - a MINIMAL key is a superkey in which NO (proper) SUBSET of the attributes is ALSO a superkey
  - a minimal key is also called a CANDIDATE key
  - the primary key for a relation is the minimal/candidate key that we CHOOSE (from amongst its possible minimal/candidate keys) to serve as its primary key;
  - artworks = (art_id, art_title, artist_name, year_created, price)

Question

Again consider the relation:

dept(dept_name, DEPT_NUM, dept_loc).

And, again, remember that, in this scenario, some departments share an office.

Is the attribute-set (dept_num, dept_loc) a **candidate** key for this relation?

a) Yes
b) No

Intro to the RELATIONAL model

Another KEY point

- do you see that, given these definitions, it MUST be the case that, for a relation scheme (e.g., R(A1, A2, a3, a4, a5)), the primary key attributes MUST functionally determine the remaining attributes?

Intro to the RELATIONAL model

Another KEY point

- do you see that, given these definitions, it MUST be the case that, for a relation scheme (e.g., R(A1, A2, a3, a4, a5)), the primary key attributes MUST functionally determine the remaining attributes?
- (that is, (A1, A2) -> (a3, a4, a5))

Question

Consider the following relation, expressed in relation structure form:

Relly(ATT_A, ATT_B, att_c, att_d, att_e)

Assume this is a "good", real relation. Based on our relational definitions so far, which of the following functional dependencies must be the case?

1) att_a -> att_c
2) att_c -> (att_a, att_b)
3) (att_a, att_b) -> att_e
4) (att_c, att_d, att_e) -> (att_a, att_b)
5) att_b -> (att_c, att_d, att_e)

Intro to the RELATIONAL model

ASIDE: the concept of physical keys/indexes

Intro to the RELATIONAL model

ASIDE: the concept of physical keys/indexes

- some DBMSs support identifying certain attributes in a table and it will "add" support to more quickly access those attributes;

Intro to the RELATIONAL model

ASIDE: the concept of physical keys/indexes

- some DBMSs support identifying certain attributes in a table and it will "add" support to more quickly access those attributes;
- Consider this relation:
- advisor = (advisor_id, advisor_lname, advisor_fname, advisor_phone)

Intro to the RELATIONAL model

ASIDE: the concept of physical keys/indexes

- some DBMSs support identifying certain attributes in a table and it will "add" support to more quickly access those attributes;
- these are NOT the same as primary keys! they are just for efficiency of certain searches...

Intro to the RELATIONAL model

ONE more important point... for future use:

- we've just said that every primary key must be the determinant of a functional dependency involving that relation, such that the primary key determines the rest of the attributes in that relation;

Intro to the RELATIONAL model

ONE more important point... for future use:

- we've just said that every primary key must be the determinant of a functional dependency involving that relation, such that the primary key determines the rest of the attributes in that relation;
  - BUT -- that is NOT to say that EVERY determinant in a relation is a primary key!

Intro to the RELATIONAL model

ONE more important point... for future use:

- we've just said that every primary key must be the determinant of a functional dependency involving that relation, such that the primary key determines the rest of the attributes in that relation;
  - BUT -- that is NOT to say that EVERY determinant in a relation is a primary key!
- Consider this relation:
  - student(STU_ID, stu_ssn, stu_lname, stu_fname, stu_email)
  - ...but stu_ssn is not the candidate key I chose for student's primary key.

Intro to the RELATIONAL model

- We've just said that every primary key must be the determinant of a functional dependency involving that relation, such that the primary key determines the rest of the attributes in that relation;
  - BUT -- that is NOT to say that EVERY determinant in a relation is a primary key!
- Consider this relation:
- ClassTaker(CLASSTAKER_ID, ctaker_lname, ctaker_fname, ctaker_phone, advisor_id, advisor_lname, advisor_phone)
  - Here, it IS the case that
  - (classtaker_id) -> (ctaker_lname, ctaker_fname, ctaker_phone, advisor_id, advisor_lname, advisor_phone)

Intro to the RELATIONAL model

- BUT -- that is NOT to say that EVERY determinant in a relation is a primary key!
- ClassTaker(CLASSTAKER_ID, ctaker_lname, ctaker_fname, ctaker_phone, advisor_id, advisor_lname, advisor_phone)
  - Here, it IS the case that
  - (classtaker_id) -> (ctaker_lname, ctaker_fname, ctaker_phone, advisor_id, advisor_lname, advisor_phone)
  - BUT it is also the case that
  - (advisor_id) -> (advisor_lname, advisor_phone)
  - ...so in this case, advisor_id IS a determinant of a functional dependency, BUT it is not the primary key or even a candidate key for the relation ClassTaker

Intro to the RELATIONAL model

what, then, is a FOREIGN key?

- (broader/theoretical definition)
- a foreign key is a set of attributes within one relation that is also a CANDIDATE key of another relation

Intro to the RELATIONAL model

what, then, is a FOREIGN key?

- (broader/theoretical definition)
- a foreign key is a set of attributes within one relation that is also a CANDIDATE key of another relation
- BUT that's hard for DBMSs to determine automatically! SO in practice, many/most DBMSs require a foreign key to be a set of attributes within one relation that is also a PRIMARY key of another relation

Intro to the RELATIONAL model

what, then, is a FOREIGN key?

- (broader/theoretical definition)
- a foreign key is a set of attributes within one relation that is also a CANDIDATE key of another relation
- BUT that's hard for DBMSs to determine automatically! SO in practice, many/most DBMSs require a foreign key to be a set of attributes within one relation that is also a PRIMARY key of another relation
- these let us relate the tuples/rows in different relations in USEFUL ways... 8-)

Intro to the RELATIONAL model

some INTEGRITY definitions:

- referential integrity is when the foreign key attributes in a table are either null or also values in the referenced table
- ...then your data has referential integrity;

Intro to the RELATIONAL model

some INTEGRITY definitions:

- referential integrity is when the foreign key attributes in a table are either null or also values in the referenced table
- ...then your data has referential integrity;
- The foreign key can be null in certain cases because not all relationships between tables are required or mandatory. In those situations, a null value indicates the absence of a relationship.

Intro to the RELATIONAL model

some INTEGRITY definitions:

- referential integrity is when the foreign key attributes in a table are either null or also values in the referenced table
- ...then your data has referential integrity;
- ...DBMSs frequently do support referential integrity! Oracle does!

Intro to the RELATIONAL model

some INTEGRITY definitions:

- referential integrity is when the foreign key attributes in a table are either null or also values in the referenced table
- ...then your data has referential integrity;
- ...DBMSs frequently do support referential integrity! Oracle does!
- ...you make a set of attribute/attributes a foreign key, and Oracle enforces referential integrity

Question

You have table parts, with relation structure:

parts(PART_NUM, part_name, part_quant, part_price)

and table part_orders, with relation structure:

part_orders(ORD_NUM, part_num, ord_quant)

foreign key (part_num) references part

You try to insert an order with a part number that does not exist in the parts table, and the DBMS does not allow it. Which kind of integrity is the DBMS enforcing in this case?

1) entity integrity
2) domain integrity
3) referential integrity

Intro to the RELATIONAL model

some INTEGRITY definitions:

- entity integrity is when in a relation, no attribute of the primary key can be null AND no two rows can have the SAME value for the primary key

Intro to the RELATIONAL model

some INTEGRITY definitions:

- entity integrity is when in a relation, no attribute of the primary key can be null AND no two rows can have the SAME value for the primary key
- ...then your data has entity integrity;

Intro to the RELATIONAL model

some INTEGRITY definitions:

- entity integrity is when in a relation, no attribute of the primary key can be null AND no two rows can have the SAME value for the primary key
- ...then your data has entity integrity;
- ...DBMSs frequently do support entity integrity! Oracle does!
- ...you make a set of attribute/attributes a primary key, and Oracle enforces entity integrity

Question

You have table parts, with relation structure:

parts(PART_NUM, part_name, part_quant, part_price)

and table part_orders, with relation structure:

part_orders(ORD_NUM, part_num, ord_quant)

foreign key (part_num) references part

You try to insert a part with no part number, and the DBMS does not allow it. Which kind of integrity is the DBMS enforcing in this case?

1) entity integrity
2) domain integrity
3) referential integrity

Intro to the RELATIONAL model

some CLASSIC relational algebra operations

Intro to the RELATIONAL model

some CLASSIC relational algebra operations

- in relational algebra, the relational operations manipulate relations to form new relations!
- and, the results of a relational operation on relations are also relations
- (and because relations are sets, set operations work on them also, but we'll get to most of those later)

Intro to the RELATIONAL model

some CLASSIC relational algebra operations

- these are just a useful subset of the relational algebra operations that there are:
  - Selection
  - Projection
  - (Cartesian) product
  - natural join
  - equi-join

Intro to the RELATIONAL model

- Selection
  - when you select JUST specified tuples/rows from a relation

Intro to the RELATIONAL model

- Selection

For example, say that you have a **Student** relation as follows:

STUDENT table:

Stud_ID	Stud_LName	Stud_Major	Stud_Grade_Level	Stud_Age
123	Jones	History	JR	21
158	Parks	Math	GR	26
105	Anderson	Management	SR	27
271	Smith	History	JR	19

Then the result of the **selection** operation on the **Student** table of rows in which **Stud_Age < 25** would be:

STUDENT WHERE Stud_Age < 25

Stud_ID	Stud_LName	Stud_Major	Stud_Grade_Level	Stud_Age
123	Jones	History	JR	21
271	Smith	History	JR	19

Intro to the RELATIONAL model

- Projection
  - when you select JUST specified attributes from a relation (and make sure the result has no duplicate rows)

Intro to the RELATIONAL model

- Projection

For example, say that you have a **Student** relation as follows:

STUDENT table:

Stud_ ID	Stud_LName	Stud_Major	Stud_Grade _Level	Stud_Age
123	Jones	History	JR	21
158	Parks	Math	GR	26
105	Anderson	Management	SR	27
271	Smith	History	JR	19

Then the result of the **projection** operation of the **Student_Major** and **Stud_Grade_Level** attributes of the **Student** table would be:

Stud_Major	Stud_Grade_Level
History	JR
Math	GR
Management	SR

Question

STUDENT table:

Stud_ ID	Stud_LName	Stud_Major	Stud_Grade _Level	Stud_Age
123	Jones	History	JR	21
158	Parks	Math	GR	26
105	Anderson	Management	SR	27
271	Smith	History	JR	19

Consider the displayed Student table. What would be the result of projecting the stud_major and stud_name column from the selection of rows in which stud_grade_level is GR?

(Assume each includes the appropriate column names/headings.)

1. Math Parks
2. Parks Math GR
3. 158 Parks Math GR 26
4. stud_major stud_name

Intro to the RELATIONAL model

- (Cartesian) product
  - the set of ALL pairs of ONE element from one set and ONE element from another set
  - ...but when relations are involved, the sets are sets of tuples!

Intro to the RELATIONAL model

- (Cartesian) product

For example, say that you have a **Price** relation as follows:

Prod_Code	Price
AA	5.99
BB	22.75

And, say that you have a **Location** table as follows:

Store	Aisle	Shelf
23	W	5
24	K	9
25	Z	6

Then the **Cartesian product** of Price and Location would be:

Prod_Code	Price	Store	Aisle	Shelf
AA	5.99	23	W	5
AA	5.99	24	K	9
AA	5.99	25	Z	6
BB	22.75	23	W	5
BB	22.75	24	K	9
BB	22.75	25	Z	6

Question

Assume that you have a table **Empl** with 20 rows. Which of the following is the case if you take the Cartesian product:

Empl x **Empl**

1. The resulting relation will have 20 rows.
2. The resulting relation will have 40 rows.
3. The resulting relation will have 400 rows.
4. The resulting relation will have no rows.

Intro to the RELATIONAL model

- equi-join, natural join
  - equi-join: when you take the Cartesian product of two relations, THEN you SELECT just the tuples in that product that meet some condition (usually an attribute value in one tuple being the same as an attribute value in another tuple from another relation)
  - MOST typically: you select the tuples in the Cartesian product that have common values for the attribute(s) serving as the foreign key relating those two tables

Intro to the RELATIONAL model

- equi-join, natural join
  - equi-join: when you take the Cartesian product of two relations, THEN you SELECT just the tuples in that product that meet some condition (usually an attribute value in one tuple being the same as an attribute value in another tuple from another relation)
  - MOST typically: you select the tuples in the Cartesian product that have common values for the attribute(s) serving as the foreign key relating those two tables
  - natural join: is the above, THEN you project ALL but one of the attributes/columns you joined on;

Intro to the RELATIONAL model

- equi-join, natural join

For example, say that you have **Student** and **Enrollment** tables as follows:

STUDENT table:

Stud_ ID	Stud_LName	Stud_Major	Stud_Grade_ Level	Stud_Age
123	Jones	History	JR	21
158	Parks	Math	GR	26
105	Anderson	Management	SR	27
271	Smith	History	JR	19

ENROLLMENT table:

Stud_ID	Class_Name	Position_Num
123	H350	1
105	BA490	3
123	BA490	7

Say that you wish to compute the **equi-join** and **natural join** of these tables based on the join condition (**Student.stud_id = Enrollment.stud_id**).

RELATIONAL model

- equi-join, natural join

First, compute the **Cartesian product** of these two tables:

Student. Stud_ID	Stud_ LName	Stud_ Major	Stud_ Grade_ Level	Stud_ Age	Enrollment .Stud_ID	Class_ Name	Position _Num
123	Jones	History	JR	21	123	H350	1
123	Jones	History	JR	21	105	BA490	3
123	Jones	History	JR	21	123	BA490	7
158	Parks	Math	GR	26	123	H350	1
158	Parks	Math	GR	26	105	BA490	3
158	Parks	Math	GR	26	123	BA490	7
105	Anderson	Management	SR	27	123	H350	1
105	Anderson	Management	SR	27	105	BA490	3
105	Anderson	Management	SR	27	123	BA490	7
271	Smith	History	JR	19	123	H350	1
271	Smith	History	JR	19	105	BA490	3
271	Smith	History	JR	19	123	BA490	7

Second, perform a selection on this result of only those rows for which (**Student.stud_id = Enrollment.stud_id)**:

Student. Stud_ID	Stud_ LName	Stud_ Major	Stud_ Grade_ Level	Stud_ Age	Enrollment .Stud_ID	Class_ Name	Position _Num
123	Jones	History	JR	21	123	H350	1
123	Jones	History	JR	21	123	BA490	7
105	Anderson	Management	SR	27	105	BA490	3

This is the **equi-join** of these two tables on the join condition **(Student.stud_id = Enrollment.stud_id)**.

Then, the **natural join** of these two tables on this join condition would include the **third** step of now projecting all of the columns in this result except for one of the "duplicate-contents" columns (and it doesn't matter which one of the two is omitted).

So, for example, the resulting **natural join** in this case could be:

Stud_ID	Stud_ LName	Stud_ Major	Stud_ Grade_ Level	Stud_ Age	Class_ Name	Position_ Num
123	Jones	History	JR	21	H350	1
123	Jones	History	JR	21	BA490	7
105	Anderson	Management	SR	27	BA490	3

Question

Which has more columns -- the natural join of two tables, or the equi-join of two tables?

1) The natural join
2) The equi-join
3) Neither -- both always have the same number of columns
4) In general, one cannot say for sure.

```
dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
Last login: Fri Sep 13 16:33:10 on ttys002
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 failed login: Fri Sep 13 16:45:07 PDT 2024 from 137.150.82.116 on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Fri Sep 13 16:33:27 2024 from 137.150.82.116
[dl313@nrs-projects ~]$ mkdir f24-325lect04-1
[dl313@nrs-projects ~]$ chmod 700 f24-325lect04-1
[dl313@nrs-projects ~]$ cd f24-325lect04-1
[dl313@nrs-projects f24-325lect04-1]$
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

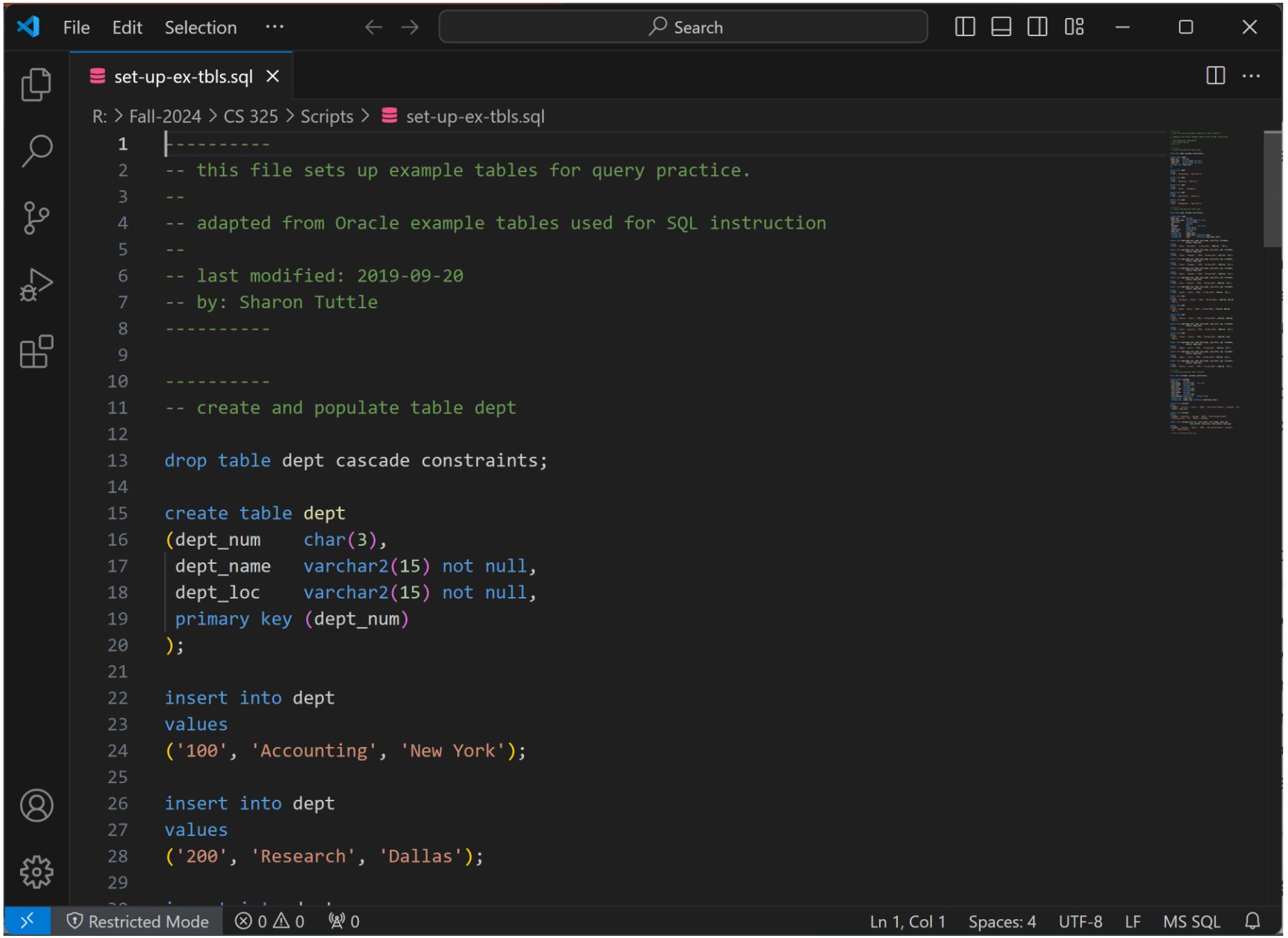
```
Last login: Fri Sep 13 16:33:10 on ttys002
dl313@CS-S34217 ~ % ssh dl313@nrs-projects-ssh.humboldt.edu
dl313@nrs-projects-ssh.humboldt.edu's password:
Activate the web console with: systemctl enable --now cockpit.socket

Last failed login: Fri Sep 13 16:45:07 PDT 2024 from 137.150.82.116 on ssh:notty
There was 1 failed login attempt since the last successful login.
Last login: Fri Sep 13 16:33:27 2024 from 137.150.82.116
[dl313@nrs-projects ~]$ mkdir f24-325lect04-1
[dl313@nrs-projects ~]$ chmod 700 f24-325lect04-1
[dl313@nrs-projects ~]$ cd f24-325lect04-1
[dl313@nrs-projects f24-325lect04-1]$ vim 325lect04-1.sql
```

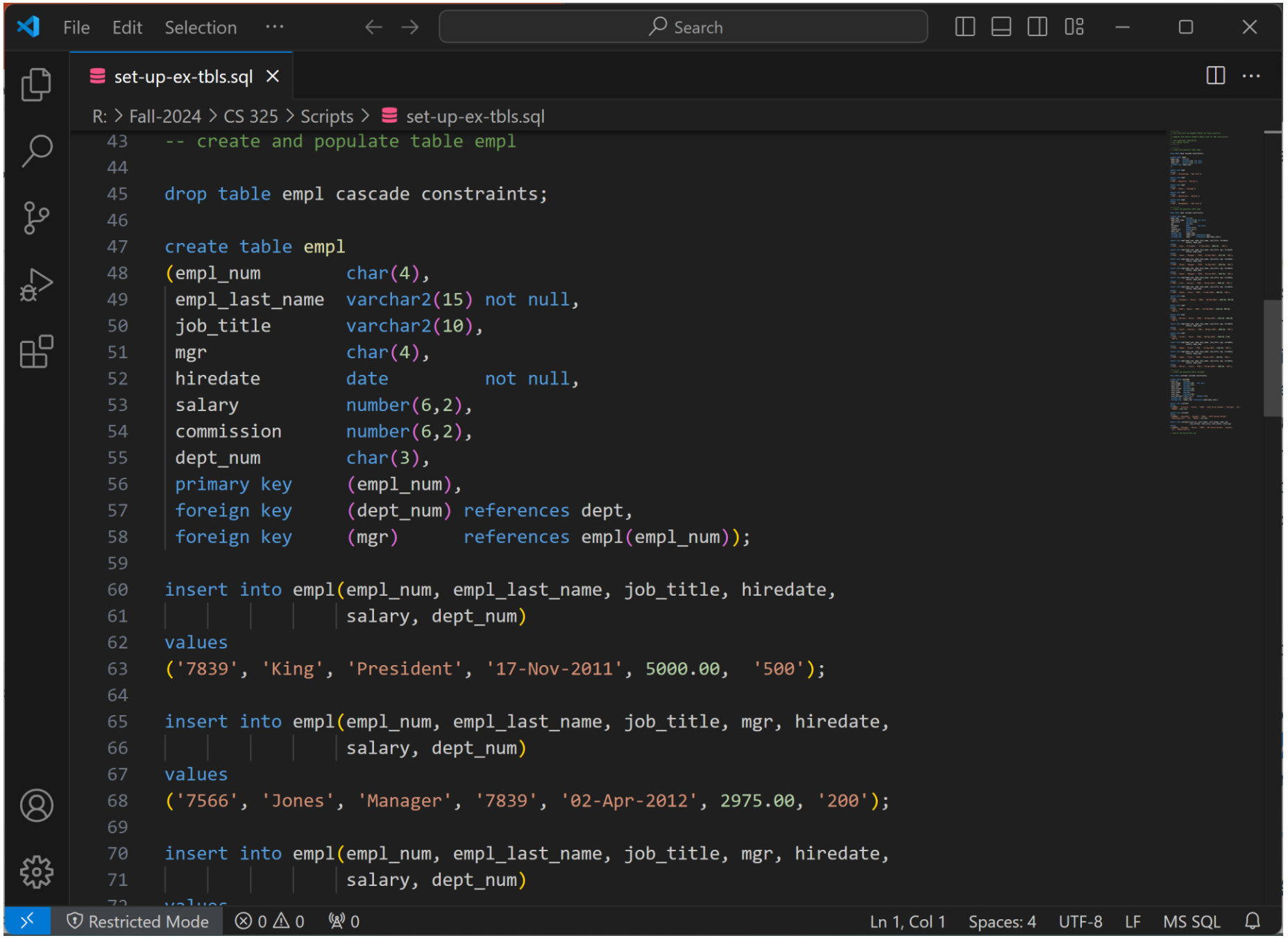
dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

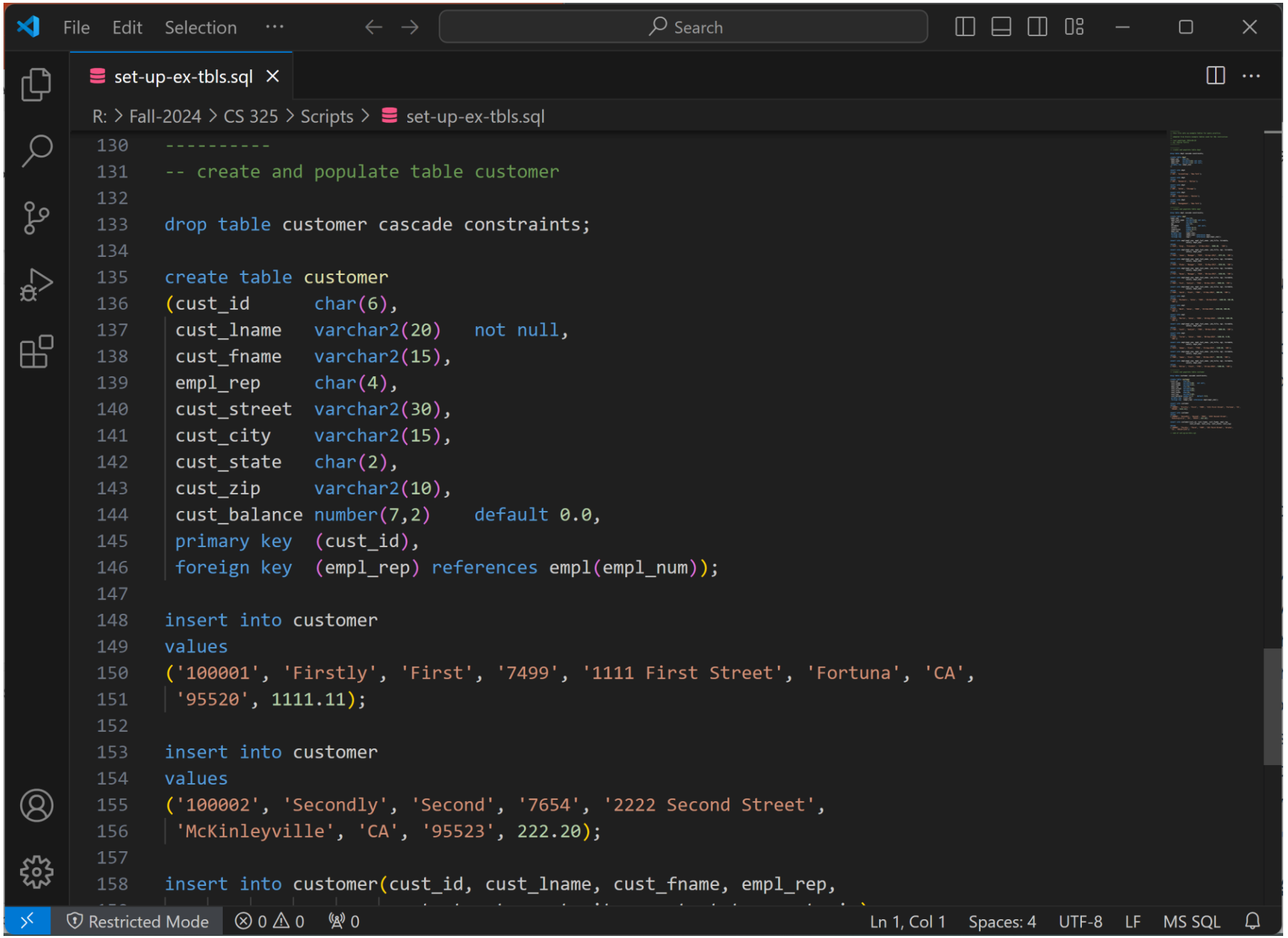
```
/* let's express relational operations in SQL */

/* pure selection! in SQL

   select *
   from   desired_table
   where  desired_bool_expr;

*/
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                              9,3           All
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/* let's express relational operations in SQL */

-- get some example tables to play with
start set-up-ex-tbls.sql

/* pure selection! in SQL

   select *
   from   desired_table
   where  desired_bool_expr;

*/
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --                                          4,25          All
```


File
Edit
Selection
Search
set-up-ex-tbls.sql
R: > Fall-2024 > CS 325 > Scripts > set-up-ex-tbls.sql
1 ----------
2 -- this file sets up example tables for query practice.
3 --
4 -- adapted from Oracle example tables used for SQL instruction
5 --
6 -- last modified: 2019-09-20
7 -- by: Sharon Tuttle
8 ----------
9
10 ----------
11 -- create and populate table dept
12
13 drop table dept cascade constraints;
14
15 create table dept
16 (dept_num char(3),
17 dept_name varchar2(15) not null,
18 dept_loc varchar2(15) not null,
19 primary key (dept_num)
20);
21
22 insert into dept
23 values
24 ('100', 'Accounting', 'New York');
25
26 insert into dept
27 values
28 ('200', 'Research', 'Dallas');
29
Restricted Mode
0 0
0
Ln 1, Col 1
Spaces: 4
UTF-8
LF
MS SQL


File Edit Selection ···
Search
set-up-ex-tbls.sql
R: > Fall-2024 > CS 325 > Scripts > set-up-ex-tbls.sql
-- create and populate table empl
drop table empl cascade constraints;
create table empl
(empl_num char(4),
empl_last_name varchar2(15) not null,
job_title varchar2(10),
mgr char(4),
hiredate date not null,
salary number(6,2),
commission number(6,2),
dept_num char(3),
primary key (empl_num),
foreign key (dept_num) references dept,
foreign key (mgr) references empl(empl_num));
insert into empl(empl_num, empl_last_name, job_title, hiredate,
salary, dept_num)
values
('7839', 'King', 'President', '17-Nov-2011', 5000.00, '500');
insert into empl(empl_num, empl_last_name, job_title, mgr, hiredate,
salary, dept_num)
values
('7566', 'Jones', 'Manager', '7839', '02-Apr-2012', 2975.00, '200');
insert into empl(empl_num, empl_last_name, job_title, mgr, hiredate,
salary, dept_num)
Restricted Mode 0 0 0
Ln 1, Col 1 Spaces: 4 UTF-8 LF MS SQL


File Edit Selection ···
Search
set-up-ex-tbls.sql
R: > Fall-2024 > CS 325 > Scripts > set-up-ex-tbls.sql
130 ----------
131 -- create and populate table customer
132
133 drop table customer cascade constraints;
134
135 create table customer
136 (cust_id char(6),
137 cust_lname varchar2(20) not null,
138 cust_fname varchar2(15),
139 empl_rep char(4),
140 cust_street varchar2(30),
141 cust_city varchar2(15),
142 cust_state char(2),
143 cust_zip varchar2(10),
144 cust_balance number(7,2) default 0.0,
145 primary key (cust_id),
146 foreign key (empl_rep) references empl(empl_num));
147
148 insert into customer
149 values
150 ('100001', 'Firstly', 'First', '7499', '1111 First Street', 'Fortuna', 'CA',
151 '95520', 1111.11);
152
153 insert into customer
154 values
155 ('100002', 'Secondly', 'Second', '7654', '2222 Second Street',
156 'McKinleyville', 'CA', '95523', 222.20);
157
158 insert into customer(cust_id, cust_lname, cust_fname, empl_rep,
Restricted Mode
0 0
0
Ln 1, Col 1
Spaces: 4
UTF-8
LF
MS SQL

```
dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24
[dl313@nrs-projects f24-325lect04-1]$ sqlplus /

SQL*Plus: Release 19.0.0.0.0 - Production on Mon Sep 16 07:34:21 2024
Version 19.3.0.0.0

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

Last Successful login time: Mon Sep 16 2024 07:30:28 -07:00

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

SQL> @ 325lect04-1.sql
```

```
1 row created.


1 row created.


1 row created.


Table dropped.


Table created.


1 row created.


1 row created.


1 row created.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
/* let's express relational operations in SQL */

-- get some example tables to play with
start set-up-ex-tbls.sql

/* pure selection! in SQL

   select *
   from   desired_table
   where  desired_bool_expr;

*/

-- selection of empls whose salary is more than 1500

select *
from   empl
where  salary > 1500;
~
~
~
~
~
-- INSERT --                                          18,22         All
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
1 row created.


1 row created.


1 row created.


Table dropped.


Table created.


1 row created.


1 row created.


1 row created.

SQL> @ 325lect04-1.sql
```

```
1 row created.


1 row created.


1 row created.


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE       SALARY COMMISSION DEP
---- --------------- ---------- ---- --------- ---------- ---------- ---
7839 King            President       17-NOV-11        5000                500
7566 Jones           Manager    7839 02-APR-12        2975                200
7698 Blake           Manager    7839 01-MAY-13        2850                300
7782 Raimi           Manager    7839 09-JUN-12        2450                100
7902 Ford            Analyst    7566 03-DEC-12        3000                200
7499 Michaels        Sales      7698 20-FEB-18        1600         300    300
7788 Scott           Analyst    7566 09-NOV-18        3000                200

7 rows selected.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
-- selection of empls whose salary is more than 1500

select *
from   empl
where  salary > 1500;

/* pure projection!

   select distinct attrib1, attrib2, ...
   from desired_tbl;
*/

-- projection of dept_num and job_title
```

-- INSERT -- 30,1 92%

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
-- selection of empls whose salary is more than 1500

select *
from   empl
where  salary > 1500;

/* pure projection!

   select distinct attrib1, attrib2, ...
   from desired_tbl;
*/

-- projection of dept_num and job_title

select distinct dept_num, job_title
from   empl;
```

-- INSERT -- 30,1 92%

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
1 row created.


1 row created.


1 row created.


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE       SALARY COMMISSION DEP
---- --------------- ---------- ---- --------- ---------- ---------- ---
7839 King            President       17-NOV-11        5000                500
7566 Jones           Manager    7839 02-APR-12        2975                200
7698 Blake           Manager    7839 01-MAY-13        2850                300
7782 Raimi           Manager    7839 09-JUN-12        2450                100
7902 Ford            Analyst    7566 03-DEC-12        3000                200
7499 Michaels        Sales      7698 20-FEB-18        1600         300    300
7788 Scott           Analyst    7566 09-NOV-18        3000                200

7 rows selected.

SQL> @ 325lect04-1.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
7782 Raimi           Manager    7839 09-JUN-12       2450                100
7902 Ford            Analyst    7566 03-DEC-12       3000                200
7499 Michaels        Sales      7698 20-FEB-18       1600         300    300
7788 Scott           Analyst    7566 09-NOV-18       3000                200

7 rows selected.


DEP JOB_TITLE
--- ----------
200 Analyst
300 Clerk
500 President
100 Manager
200 Clerk
300 Manager
200 Manager
100 Clerk
300 Sales
400 Clerk

10 rows selected.

SQL>
```

Question

product_id	product_name	price	category
1	Laptop	1000	Electronics
2	Smartphone	800	Electronics
3	Chair	200	Furniture
4	Desk	400	Furniture
5	TV	1200	Electronics

Which SQL query will return only the product_name and category for products priced above 500, without duplicates?

1) SELECT product_name, category FROM products WHERE price > 500;
2) SELECT DISTINCT product_name, category FROM products WHERE price > 500;
3) SELECT product_name, category FROM products WHERE price < 500;
4) SELECT DISTINCT * FROM products WHERE price > 500;

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
*/

-- projection of dept_num and job_title

select distinct dept_num, job_title
from   empl;

/* pure Cartesian product!

   select *
   from   tbl1, tb2;

*/

-- Cartesian product of empl and dept (ugly!)

select *
from   empl, dept;
```

39,0-1 Bot

```
7782 Raimi           Manager    7839 09-JUN-12        2450                100
7902 Ford            Analyst    7566 03-DEC-12        3000                200
7499 Michaels        Sales      7698 20-FEB-18        1600         300    300
7788 Scott           Analyst    7566 09-NOV-18        3000                200

7 rows selected.


DEP JOB_TITLE
--- ----------
200 Analyst
300 Clerk
500 President
100 Manager
200 Clerk
300 Manager
200 Manager
100 Clerk
300 Sales
400 Clerk

10 rows selected.

SQL> @ 325lect04-1.sql
```

```
---- --------------- ---------- ---- --------- ---------- ---------- --- ---
DEPT_NAME       DEPT_LOC
--------------- --------------
7844 Turner          Sales      7698 08-SEP-19       1500          0 300 500
Management      New York

7876 Adams           Clerk      7788 23-SEP-18       1100            400 500
Management      New York

7900 James           Clerk      7698 03-DEC-17        950            300 500
Management      New York


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE      SALARY COMMISSION DEP DEP
---- --------------- ---------- ---- --------- ---------- ---------- --- ---
DEPT_NAME       DEPT_LOC
--------------- --------------
7934 Miller          Clerk      7782 23-JAN-16       1300            100 500
Management      New York


70 rows selected.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
   from   tbl1, tb2;

*/

-- Cartesian product of empl and dept (ugly!)

select *
from   empl, dept;

/* ONE form of pure equi-join!

   select *
   from  tbl1, tbl2
   where tbl1.attrib = tbl2.attrib;

*/

select *
from   empl, dept
where  empl.dept_num = dept.dept_num;
```

-- INSERT -- 53,38 Bot

```
---- --------------- ---------- ---- --------- ---------- ---------- --- ---
DEPT_NAME       DEPT_LOC
--------------- --------------
7844 Turner          Sales      7698 08-SEP-19       1500          0 300 500
Management      New York

7876 Adams           Clerk      7788 23-SEP-18       1100            400 500
Management      New York

7900 James           Clerk      7698 03-DEC-17        950            300 500
Management      New York


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE      SALARY COMMISSION DEP DEP
---- --------------- ---------- ---- --------- ---------- ---------- --- ---
DEPT_NAME       DEPT_LOC
--------------- --------------
7934 Miller          Clerk      7782 23-JAN-16       1300            100 500
Management      New York


70 rows selected.

SQL> @ 325lect04-1.sql
```

```
7788 Scott           Analyst    7566 09-NOV-18        3000              200 200
Research        Dallas

7844 Turner          Sales      7698 08-SEP-19        1500            0 300 300
Sales           Chicago

7876 Adams           Clerk      7788 23-SEP-18        1100              400 400
Operations      Boston


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE      SALARY COMMISSION DEP DEP
---- --------------- ---------- ---- --------- ---------- ---------- --- ---
DEPT_NAME       DEPT_LOC
--------------- --------------
7900 James           Clerk      7698 03-DEC-17         950              300 300
Sales           Chicago

7934 Miller          Clerk      7782 23-JAN-16        1300              100 100
Accounting      New York


14 rows selected.

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
DEPT_NAME        DEPT_LOC
---------------- ----------------
7900 James            Clerk      7698 03-DEC-17         950              300 300
Sales            Chicago

7934 Miller           Clerk      7782 23-JAN-16        1300              100 100
Accounting       New York


14 rows selected.

SQL> describe empl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPL_NUM                                  NOT NULL CHAR(4)
 EMPL_LAST_NAME                            NOT NULL VARCHAR2(15)
 JOB_TITLE                                          VARCHAR2(10)
 MGR                                                CHAR(4)
 HIREDATE                                  NOT NULL DATE
 SALARY                                             NUMBER(6,2)
 COMMISSION                                         NUMBER(6,2)
 DEPT_NUM                                           CHAR(3)

SQL>
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
14 rows selected.

SQL> describe empl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPL_NUM                                  NOT NULL CHAR(4)
 EMPL_LAST_NAME                            NOT NULL VARCHAR2(15)
 JOB_TITLE                                          VARCHAR2(10)
 MGR                                                CHAR(4)
 HIREDATE                                  NOT NULL DATE
 SALARY                                             NUMBER(6,2)
 COMMISSION                                         NUMBER(6,2)
 DEPT_NUM                                           CHAR(3)

SQL> describe dept
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPT_NUM                                  NOT NULL CHAR(3)
 DEPT_NAME                                 NOT NULL VARCHAR2(15)
 DEPT_LOC                                  NOT NULL VARCHAR2(15)

SQL>
```

```
*/

select *
from   empl, dept
where  empl.dept_num = dept.dept_num;

/* natural join is just... tedious!

   select <all attribs of tbl1 and tbl2 except ONE of tbl1's or
           tbl2's join_attrib>
   from   tbl1, tbl2
   where  tbl1.join_attrib = tbl2.join_attrib;

*/

select empl_num, empl_last_name, job_title, mgr, hiredate, salary,
       commission, empl.dept_num, dept_name, dept_loc
from   empl, dept
where  empl.dept_num = dept.dept_num;
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
14 rows selected.

SQL> describe empl
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPL_NUM                                  NOT NULL CHAR(4)
 EMPL_LAST_NAME                            NOT NULL VARCHAR2(15)
 JOB_TITLE                                          VARCHAR2(10)
 MGR                                                CHAR(4)
 HIREDATE                                  NOT NULL DATE
 SALARY                                             NUMBER(6,2)
 COMMISSION                                         NUMBER(6,2)
 DEPT_NUM                                           CHAR(3)

SQL> describe dept
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DEPT_NUM                                  NOT NULL CHAR(3)
 DEPT_NAME                                 NOT NULL VARCHAR2(15)
 DEPT_LOC                                  NOT NULL VARCHAR2(15)

SQL> @ 325lect04-1.sql
```

dl313 — dl313@nrs-projects:~/f24-325lect04-1 — ssh dl313@nrs-projects-ssh.humboldt.edu — 80×24

```
7788 Scott           Analyst    7566 09-NOV-18        3000              200
Research        Dallas

7844 Turner          Sales      7698 08-SEP-19        1500            0 300
Sales           Chicago

7876 Adams           Clerk      7788 23-SEP-18        1100              400
Operations      Boston


EMPL EMPL_LAST_NAME  JOB_TITLE  MGR  HIREDATE      SALARY COMMISSION DEP
---- --------------- ---------- ---- --------- ---------- ---------- ---
DEPT_NAME       DEPT_LOC
--------------- --------------
7900 James           Clerk      7698 03-DEC-17         950              300
Sales           Chicago

7934 Miller          Clerk      7782 23-JAN-16        1300              100
Accounting      New York


14 rows selected.

SQL>
```

Question

Which of the following are relational operations from relational algebra?

1) create table, drop table
2) selection, projection, natural join
3) insert, select
4) spool, describe