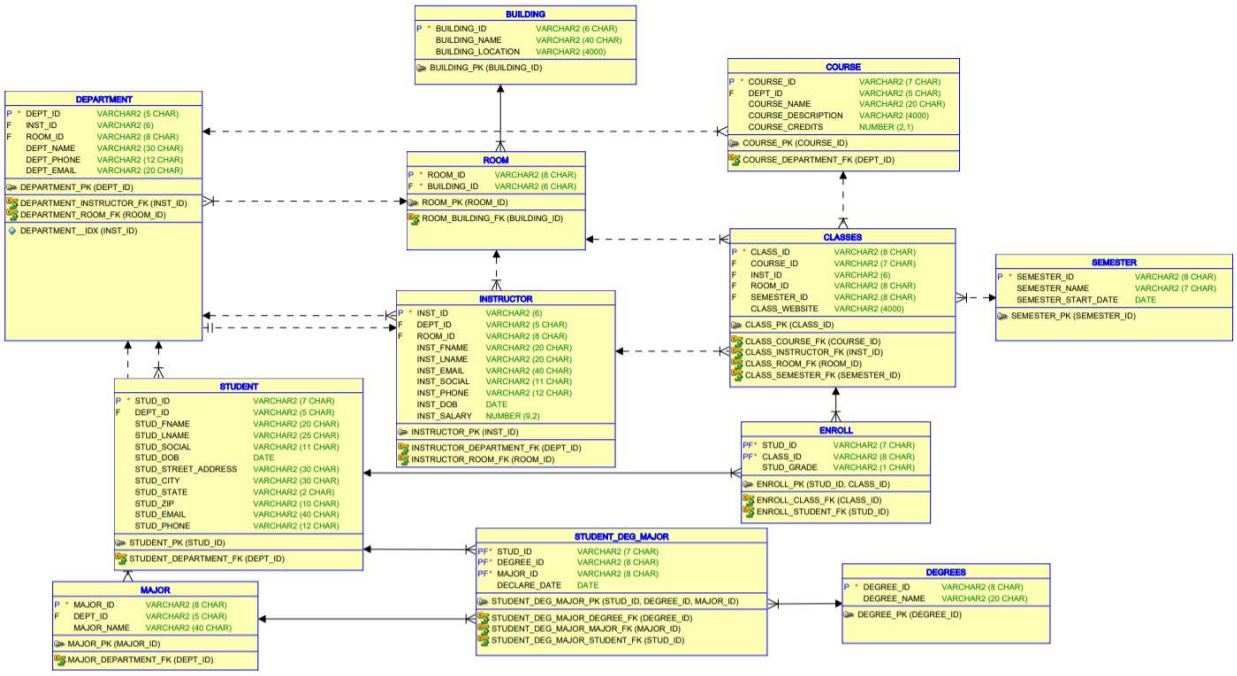
Kevin Nolasco

MCIS 510

Milestone 4

Using SQL

The purpose of this milestone is to gain experience using SQL to create tables and populate the tables with data. Once the tables are created and populated, we can query the tables to answer questions. The physical model that is used to create the DDL is shown below:



DDL (Data Definition Language)

To add data to our database, each table must be created. The code that follows will create all tables and their primary/foreign keys.

BUILDING

```
CREATE TABLE building (
    building_id        VARCHAR2(6 CHAR) NOT NULL,
    building_name      VARCHAR2(40 CHAR),
    building_location  VARCHAR2(4000)
);

ALTER TABLE building ADD CONSTRAINT building_pk PRIMARY KEY ( building_id );
```

CLASSES (I had to modify the name of this table from CLASS to CLASSES because CLASS is a reserved word in SQL. The textbook advises against using plural names for the tables but I viewed this change as a quick fix).

```
CREATE TABLE classes (
    class_id       VARCHAR2(8 CHAR) NOT NULL,
    course_id      VARCHAR2(7 CHAR),
    inst_id        VARCHAR2(6),
    room_id        VARCHAR2(8 CHAR),
    semester_id    VARCHAR2(8 CHAR),
    class_website  VARCHAR2(4000)
);

ALTER TABLE classes ADD CONSTRAINT class_pk PRIMARY KEY ( class_id );
```

COURSE

```
CREATE TABLE course (
    course_id           VARCHAR2(7 CHAR) NOT NULL,
    dept_id             VARCHAR2(5 CHAR),
    course_name         VARCHAR2(20 CHAR),
    course_description  VARCHAR2(4000),
    course_credits      NUMBER(2, 1)
);

ALTER TABLE course ADD CONSTRAINT course_pk PRIMARY KEY ( course_id );
```

DEGREES (I had to modify the name of this table from DEGREE to DEGREES because DEGREE is a reserved word in SQL. The textbook advises against using plural names for the tables but I viewed this change as a quick fix).

```
CREATE TABLE degrees (
    degree_id    VARCHAR2(8 CHAR) NOT NULL,
    degree_name  VARCHAR2(20 CHAR)
);

ALTER TABLE degrees ADD CONSTRAINT degree_pk PRIMARY KEY ( degree_id );
```

DEPARTMENT

```
CREATE TABLE department (
    dept_id     VARCHAR2(5 CHAR) NOT NULL,
    inst_id     VARCHAR2(6),
    room_id     VARCHAR2(8 CHAR),
    dept_name   VARCHAR2(30 CHAR),
    dept_phone  VARCHAR2(12 CHAR),
    dept_email  VARCHAR2(20 CHAR)
);

CREATE UNIQUE INDEX department__idx ON
    department (
        inst_id
    ASC );

ALTER TABLE department ADD CONSTRAINT department_pk PRIMARY KEY ( dept_id );
```

Since there is a 1:1 relationship between DEPARTMENT and INSTRUCTOR, the unique index is used here to describe the INST_ID.

ENROLL

```
CREATE TABLE enroll (
    stud_id     VARCHAR2(7 CHAR) NOT NULL,
    class_id    VARCHAR2(8 CHAR) NOT NULL,
    stud_grade  VARCHAR2(1 CHAR)
);

ALTER TABLE enroll ADD CONSTRAINT enroll_pk PRIMARY KEY ( stud_id,
                                                          class_id );
```

ENROLL's primary key is a composite key and is defined in a single line using parenthesis.

INSTRUCTOR

```
CREATE TABLE instructor (
    inst_id      VARCHAR2(6) NOT NULL,
    dept_id      VARCHAR2(5 CHAR),
    room_id      VARCHAR2(8 CHAR),
    inst_fname   VARCHAR2(20 CHAR),
    inst_lname   VARCHAR2(20 CHAR),
    inst_email   VARCHAR2(40 CHAR),
    inst_social  VARCHAR2(11 CHAR),
    inst_phone   VARCHAR2(12 CHAR),
    inst_dob     DATE,
    inst_salary  NUMBER(9, 2)
);


ALTER TABLE instructor ADD CONSTRAINT instructor_pk PRIMARY KEY ( inst_id );
```

MAJOR

```
CREATE TABLE major (
    major_id    VARCHAR2(8 CHAR) NOT NULL,
    dept_id     VARCHAR2(5 CHAR),
    major_name  VARCHAR2(40 CHAR)
);

ALTER TABLE major ADD CONSTRAINT major_pk PRIMARY KEY ( major_id );
```

ROOM

```
CREATE TABLE room (
    room_id      VARCHAR2(8 CHAR) NOT NULL,
    building_id  VARCHAR2(6 CHAR) NOT NULL
);

ALTER TABLE room ADD CONSTRAINT room_pk PRIMARY KEY ( room_id );
```

SEMESTER

```
CREATE TABLE semester (
    semester_id          VARCHAR2(8 CHAR) NOT NULL,
    semester_name        VARCHAR2(7 CHAR),
    semester_start_date  DATE
);

ALTER TABLE semester ADD CONSTRAINT semester_pk PRIMARY KEY ( semester_id );
```

STUDENT

```
CREATE TABLE student (
    stud_id             VARCHAR2(7 CHAR) NOT NULL,
    dept_id             VARCHAR2(5 CHAR),
    stud_fname          VARCHAR2(20 CHAR),
    stud_lname          VARCHAR2(25 CHAR),
    stud_social         VARCHAR2(11 CHAR),
    stud_dob            DATE,
    stud_street_address VARCHAR2(30 CHAR),
    stud_city           VARCHAR2(30 CHAR),
    stud_state          VARCHAR2(2 CHAR),
    stud_zip            VARCHAR2(10 CHAR),
    stud_email          VARCHAR2(40 CHAR),
    stud_phone          VARCHAR2(12 CHAR)
);

ALTER TABLE student ADD CONSTRAINT student_pk PRIMARY KEY ( stud_id );
```

STUDENT_DEG_MAJOR

```
CREATE TABLE student_deg_major (
    stud_id      VARCHAR2(7 CHAR) NOT NULL,
    degree_id    VARCHAR2(8 CHAR) NOT NULL,
    major_id     VARCHAR2(8 CHAR) NOT NULL,
    declare_date DATE
);

ALTER TABLE student_deg_major
    ADD CONSTRAINT student_deg_major_pk PRIMARY KEY ( stud_id,
                                                      degree_id,
                                                      major_id );
```

Here we see that STUDENT_DEG_MAJOR has a primary key that is a composite key and is declared in a single line. Since those three attributes are the primary key, we specify that they must not be NULL.

All the tables are now created at this point and it is time to add the foreign keys. (I was surprised to find out that there is no way of declaring the foreign keys in one *ALTER TABLE* statement!)

CLASSES

```
ALTER TABLE classes
    ADD CONSTRAINT class_course_fk FOREIGN KEY ( course_id )
        REFERENCES course ( course_id );

ALTER TABLE classes
    ADD CONSTRAINT class_instructor_fk FOREIGN KEY ( inst_id )
        REFERENCES instructor ( inst_id );

ALTER TABLE classes
    ADD CONSTRAINT class_room_fk FOREIGN KEY ( room_id )
        REFERENCES room ( room_id );

ALTER TABLE classes
    ADD CONSTRAINT class_semester_fk FOREIGN KEY ( semester_id )
        REFERENCES semester ( semester_id );
```

COURSE

```
ALTER TABLE course
    ADD CONSTRAINT course_department_fk FOREIGN KEY ( dept_id )
        REFERENCES department ( dept_id );
```

DEPARTMENT

```
ALTER TABLE department
    ADD CONSTRAINT department_instructor_fk FOREIGN KEY ( inst_id )
        REFERENCES instructor ( inst_id );

ALTER TABLE department
    ADD CONSTRAINT department_room_fk FOREIGN KEY ( room_id )
        REFERENCES room ( room_id );
```

ENROLL

```
ALTER TABLE enroll
    ADD CONSTRAINT enroll_class_fk FOREIGN KEY ( class_id )
        REFERENCES classes ( class_id );

ALTER TABLE enroll
    ADD CONSTRAINT enroll_student_fk FOREIGN KEY ( stud_id )
        REFERENCES student ( stud_id );
```

INSTRUCTOR

```
ALTER TABLE instructor
    ADD CONSTRAINT instructor_department_fk FOREIGN KEY ( dept_id )
        REFERENCES department ( dept_id );

ALTER TABLE instructor
    ADD CONSTRAINT instructor_room_fk FOREIGN KEY ( room_id )
        REFERENCES room ( room_id );
```

MAJOR

```
ALTER TABLE major
    ADD CONSTRAINT major_department_fk FOREIGN KEY ( dept_id )
        REFERENCES department ( dept_id );
```

ROOM

```
ALTER TABLE room
    ADD CONSTRAINT room_building_fk FOREIGN KEY ( building_id )
        REFERENCES building ( building_id );
```

STUDENT_DEG_MAJOR

```
ALTER TABLE student_deg_major
    ADD CONSTRAINT student_deg_major_degree_fk FOREIGN KEY ( degree_id )
        REFERENCES degrees ( degree_id );

ALTER TABLE student_deg_major
    ADD CONSTRAINT student_deg_major_major_fk FOREIGN KEY ( major_id )
        REFERENCES major ( major_id );

ALTER TABLE student_deg_major
    ADD CONSTRAINT student_deg_major_student_fk FOREIGN KEY ( stud_id )
        REFERENCES student ( stud_id );
```

STUDENT

```
ALTER TABLE student
    ADD CONSTRAINT student_department_fk FOREIGN KEY ( dept_id )
        REFERENCES department ( dept_id );
```

DML (Data Manipulation Language)

Now that all tables are created and the primary/foreign keys have been defined, it is time to add data to the tables. It is important to add rows to tables that are in the “1” side of a 1:M relationship. Therefore, I begin by adding rows to the tables that have no foreign keys.

(I had fun creating fake data for this project. I used python to write my SQL commands and to make my data. The code for this fake data is found in my github. You could find it here:

https://github.com/knolasco/Data-Science-Projects/blob/master/Database%20Management/Fake_Data_for_Database.ipynb

SEMESTER

The code that defines the first three rows of SEMESTER are shown below. The entirety of SEMESTER is shown after using (SELECT * FROM SEMESTER;).

```
INSERT INTO SEMESTER
    VALUES ('87016831','Summer','');
INSERT INTO SEMESTER
    VALUES ('20813867','Winter','');
INSERT INTO SEMESTER
    VALUES ('75100245','Fall','');
```

	SEMESTER_ID	SEMESTER_NAME	SEMESTER_START_DATE
1	87016831	Summer	26-MAY-13
2	20813867	Winter	24-DEC-16
3	75100245	Fall	18-AUG-18
4	45114241	Winter	22-DEC-11
5	14942265	Winter	07-DEC-19
6	10060628	Spring	18-JAN-21
7	29539150	Fall	26-AUG-21
8	98962001	Spring	09-JAN-20
9	88716606	Winter	26-DEC-19
10	73116457	Summer	22-MAY-11
11	60084737	Fall	28-AUG-16
12	52867218	Summer	11-MAY-11
13	12937999	Summer	12-MAY-16
14	46600917	Spring	26-JAN-14
15	75611164	Fall	15-AUG-21

BUILDING

The code that defines the first three rows of BUILDING are shown below. The entirety of BUILDING is shown after using (SELECT * FROM BUILDING;).

```
INSERT INTO BUILDING
    VALUES ('347514','Kevin Johnson Hall','North Campus');
INSERT INTO BUILDING
    VALUES ('418031','Edward Brown Hall','South Campus');
INSERT INTO BUILDING
    VALUES ('208177','Sydney Smith Hall','North Campus');
```

	BUILDING_ID	BUILDING_NAME	BUILDING_LOCATION
1	347514	Kevin Johnson Hall	North Campus
2	418031	Edward Brown Hall	South Campus
3	208177	Sydney Smith Hall	North Campus
4	856250	Mitchell Gentile Hall	West Campus
5	515297	Daniel Brown Hall	North Campus
6	602140	Sydney Williams Hall	South Campus
7	262500	Carlos Palacios Hall	North Campus
8	194476	Edward Palacios Hall	North Campus
9	169746	Ashley Nolasco Hall	West Campus

DEGREES

The code that defines the first three rows of DEGREES are shown below. The entirety of DEGREES is shown after using (SELECT * FROM DEGREES;).

```
INSERT INTO DEGREES
    VALUES ('38048866','Bachelor of Arts');
INSERT INTO DEGREES
    VALUES ('11564266','Bachelor of Science');
INSERT INTO DEGREES
    VALUES ('79969969','Master of Arts');
```

	DEGREE_ID	DEGREE_NAME
1	38048866	Bachelor of Arts
2	11564266	Bachelor of Science
3	79969969	Master of Arts
4	14855520	Master of Science

All the previous tables did not have foreign keys. The tables that follow all reference one or more of the previous tables.

ROOM

The code that defines the first three rows of ROOM are shown below. Part of ROOM is shown after using (SELECT * FROM ROOM;). I added 50 entries to this table but am only showing 17 here.

```
INSERT INTO ROOM
    VALUES ('53464097','856250');
INSERT INTO ROOM
    VALUES ('30246633','602140');
INSERT INTO ROOM
    VALUES ('62992312','262500');
```

	ROOM_ID	BUILDING_ID
1	53464097	856250
2	30246633	602140
3	62992312	262500
4	97366946	208177
5	16480894	856250
6	19722233	347514
7	81924865	418031
8	22633920	208177
9	59081935	856250
10	88220482	169746
11	17784483	856250
12	78106871	262500
13	38816302	347514
14	15032582	194476
15	21535642	194476
16	68202938	194476
17	66126116	262500

Now I would like to add rows into DEPARTMENT since many of the remaining tables reference DEPARTMENT with their foreign keys. However, DEPARTMENT has the foreign key of INSTRUCTOR. To avoid any integrity/referential violations, it is necessary to add the primary key of INSTRUCTOR before adding the rows to DEPARTMENT. The code for the first three primary keys of instructor is shown below.

```
INSERT INTO INSTRUCTOR (INST_ID)
    VALUES ('319131');
INSERT INTO INSTRUCTOR (INST_ID)
    VALUES ('112220');
INSERT INTO INSTRUCTOR (INST_ID)
    VALUES ('646640');
```

Now that INSTRUCTOR has all of it's primary keys defined, we can add rows to DEPARTMENT. The code that defines the first three rows of DEPARTMENT are shown below. (I ran into some trouble here, the column for DEPT_EMAIL was too narrow for some of the department email addresses. To fix this, I used *ALTER TABLE DEPARTMENT MODIFY DEPT_EMAIL VARCHAR(40)* to make the column wider.) The entirety of DEPARTMENT can be found after the code.

```
ALTER TABLE DEPARTMENT
    MODIFY DEPT_EMAIL VARCHAR(40);

INSERT INTO DEPARTMENT
    VALUES ('22302','319131','53464097','Mathematics','563-986-8629','mathematics@cabrini.edu');
INSERT INTO DEPARTMENT
    VALUES ('73944','112220','30246633','Biology','562-620-4111','biology@cabrini.edu');
INSERT INTO DEPARTMENT
    VALUES ('13715','646640','62992312','Chemistry','289-923-9387','chemistry@cabrini.edu');
```

	DEPT_ID	INST_ID	ROOM_ID	DEPT_NAME	DEPT_PHONE	DEPT_EMAIL
1	22302	319131	53464097	Mathematics	563-986-8629	mathematics@cabrini.edu
2	73944	112220	30246633	Biology	562-620-4111	biology@cabrini.edu
3	13715	646640	62992312	Chemistry	289-923-9387	chemistry@cabrini.edu
4	61093	871743	97366946	Physics	587-744-4050	physics@cabrini.edu
5	66723	137933	16480894	Humanities	196-557-5970	humanities@cabrini.edu
6	89618	265688	19722233	Arts	245-192-9825	arts@cabrini.edu
7	10276	350557	81924865	Philosophy	929-810-1686	philosophy@cabrini.edu
8	68377	117700	22633920	Astronomy	709-505-8421	astronomy@cabrini.edu
9	44908	157617	59081935	Sociology	769-856-3580	sociology@cabrini.edu

Since INSTRUCTOR already has it's primary keys, it is necessary to update each row and add the remaining columns. (The DML for this procedure is too long to make a screen shot of it. So I will write the code in general)

UPDATE INSTRUCTOR
SET DEPT_ID = '<dept_id>', ROOM_ID = '<room_id>', INST_FNAME = '<fname>', INST_LNAME = '<lname>', INST_EMAIL = <email >', INST_SOCIAL = '<social>', INST_PHONE = '<phone>',INST_DOB = '<DOB>', INST_SALARY = '<salary>'
WHERE INST_ID = '<id>';

The result of applying this DML to all rows of INSTRUCTOR are below.

	INST_ID	DEPT_ID	ROOM_ID	INST_F...	INST_LNAME	INST_E...	INST_S...	INST_P...	INST_DOB	INST_S...
1	319131	22302	88220482	Harold	Garcia	hgarcia...	585-44-...	365-397...	04-OCT-83	76793
2	112220	73944	17784483	Maria	Jones	mjones@...	782-54-...	767-336...	08-MAY-66	100053
3	646640	13715	78106871	Rose	Miller	rMiller...	490-11-...	990-330...	10-DEC-54	102910
4	871743	61093	38816302	Steven	Davis	sDavis@...	594-45-...	233-172...	22-AUG-69	126452
5	137933	66723	15032582	Danielle	Rodriguez	dRodrig...	807-86-...	966-319...	15-NOV-75	107494
6	265688	89618	21535642	Marcus	Martinez	mMartin...	671-10-...	130-541...	13-FEB-66	129630
7	350557	10276	68202938	John	Palacios	jpalaci...	550-57-...	957-800...	08-JUN-72	62372
8	117700	68377	66126116	Ester	Nolasco	enolasc...	447-36-...	382-954...	26-MAY-73	133999
9	157617	44908	19375836	Barry	Johnson	bjohnso...	690-35-...	186-992...	26-NOV-83	140031
10	943971	89618	42301241	Julien	Williams	jwillia...	625-97-...	946-561...	05-MAR-85	64618
11	814447	13715	22175294	Rod	Brown	rbrown@...	797-61-...	242-912...	22-NOV-67	125861
12	254401	73944	83960310	Laura	Wilson	lwilson...	119-17-...	464-970...	06-JAN-54	127212
13	998988	68377	66978001	Christina	Bryans	cbryans...	328-21-...	598-873...	04-OCT-71	133547
14	828526	44908	17933677	Jacqueline	Atzberger	jatzber...	554-24-...	786-472...	01-FEB-67	133064

STUDENT

The code that defines the first three rows of STUDENT are shown below. The entirety of STUDENT is shown after using (SELECT * FROM STUDENT;).

```
INSERT INTO STUDENT
    VALUES ('3254257','66723','Edward','Smith','838-39-2674','1-july-98','249 Adams Blvd','Huntington','CO','39745-1201','esmith@yahoo.com','616-523-8945');
INSERT INTO STUDENT
    VALUES ('2058756','10276','Carlos','Gentile','425-13-1365','29-dec-98','425 Ingelwood Blvd','Goleta','CA','90584-7448','cgentile@yahoo.com','569-714-1458');
INSERT INTO STUDENT
    VALUES ('5279348','66723','Mitchell','Mora','126-93-9870','19-jan-97','376 Hawthorne Blvd','Hermosa Beach','TN','81871-4803','mmora@outlook.com','663-698-3961');
INSERT INTO STUDENT
```

	STUD_ID	DEPT_ID	STUD_FNAME	STUD_LNAME	STUD_SOCIAL	STUD_DOB	STUD_STREET_ADDRESS	STUD_CITY	STUD_STATE	STUD_ZIP	STUD_EMAIL	STUD_PHONE
1	3254257	66723	Edward	Smith	838-39-2674	01-JUL-98	249 Adams Blvd	Huntington	CO	39745-1201	esmith@yahoo.com	616-523-8945
2	2058756	10276	Carlos	Gentile	425-13-1365	29-DEC-98	425 Ingelwood Blvd	Goleta	CA	90584-7448	cgentile@yahoo.com	569-714-1458
3	5279348	66723	Mitchell	Mora	126-93-9870	19-JAN-97	376 Hawthorne Blvd	Hermosa Beach	TN	81871-4803	mmora@outlook.com	663-698-3961
4	2978347	10276	Sydney	Palacios	109-58-4548	01-JUL-98	904 Alabama Blvd	Hawthorne	AR	81871-4803	spalacios@yahoo.com	653-738-6425
5	9312021	61093	Kevin	Nolasco	532-13-9644	10-MAY-97	89 Beach Blvd	Santa Monica	MA	63012-9417	knolasco@cabrini.edu	616-523-8945
6	8541208	13715	Daniel	Johnson	327-66-9123	04-DEC-95	680 Hawthorne Blvd	Hawthorne	MA	39745-1201	djohnson@outlook.com	616-523-8945
7	8922960	13715	Ashley	Williams	666-39-6663	19-JAN-97	111 Ingelwood Blvd	Santa Monica	AR	63012-9417	awilliams@google.com	663-698-3961
8	7368886	89618	Richard	Brown	336-96-4584	28-SEP-96	534 Adams Blvd	Compton	TN	39745-1201	rbrown@google.com	653-738-6425

COURSE

Before adding the rows, I needed to widen the COURSE_NAME column. The code that defines the first three rows of COURSE are shown below. The entirety of COURSE is shown after using (SELECT * FROM COURSE;).

```
ALTER TABLE COURSE
    MODIFY COURSE_NAME VARCHAR(50);

INSERT INTO COURSE
    VALUES ('1725678','13715','Advanced Linear Algebra','This course is Advanced Linear Algebra.','3');
INSERT INTO COURSE
    VALUES ('9731973','73944','Intro To Differential Equations','This course is Intro To Differential Equations.','3');
INSERT INTO COURSE
    VALUES ('3024816','13715','Advanced Database Management','This course is Advanced Database Management.','2');
```

	COURSE_ID	DEPT_ID	COURSE_NAME	COURSE_DESCRIPTION	COURSE_CREDITS
1	9581406	10276	Advanced Statistics	This course is Advanced Statistics.	3
2	4347009	73944	Advanced Probability	This course is Advanced Probability.	2
3	6824526	68377	Intro To Python	This course is Intro To Python.	3
4	9878484	10276	Advanced R	This course is Advanced R.	2
5	5854738	61093	Advanced Sociology	This course is Advanced Sociology.	3
6	3487386	89618	Intro To Philosophy	This course is Intro To Philosophy.	3
7	1725678	13715	Advanced Linear Algebra	This course is Advanced Linear Algebra.	3
8	9731973	73944	Intro To Differential Equations	This course is Intro To Differential Equations.	3
9	3024816	13715	Advanced Database Management	This course is Advanced Database Management.	2
10	7602416	13715	Advanced Machine Learning	This course is Advanced Machine Learning.	3

CLASSES

The code that defines the first three rows of CLASSES are shown below. The entirety of CLASSES is shown after using (SELECT * FROM CLASSES;).

```
INSERT INTO CLASSES
    VALUES('32139144','1725678','319131','85893910','88716606','advancedlinearalgebra.cabrini.edu');
INSERT INTO CLASSES
    VALUES('66102158','9731973','112220','26616417','88716606','introtodifferentialequations.cabrini.edu');
INSERT INTO CLASSES
    VALUES('66107047','3024816','646640','39962626','45114241','advanceddatabasemanagement.cabrini.edu');
```

	CLASS_ID	COURSE_ID	INST_ID	ROOM_ID	SEMESTER_ID	CLASS_WEBSITE
1	32139144	1725678	319131	85893910	88716606	advancedlinearalgebra.cabrini.edu
2	66102158	9731973	112220	26616417	88716606	introtodifferentialequations.cabrini.edu
3	66107047	3024816	646640	39962626	45114241	advanceddatabasemanagement.cabrini.edu
4	95228249	9581406	871743	94641177	12937999	advancedstatistics.cabrini.edu
5	47749676	4347009	137933	94212661	87016831	advancedprobability.cabrini.edu
6	74303776	7602416	265688	88248519	87016831	advancedmachinelearning.cabrini.edu
7	38992954	6824526	350557	18302983	10060628	introtopython.cabrini.edu
8	73706075	9878484	117700	87457446	75611164	advancedr.cabrini.edu
9	78780712	5854738	157617	88590039	73116457	advancedsociology.cabrini.edu
10	34639864	3487386	943971	63241552	29539150	introtophilosophy.cabrini.edu

MAJOR

The code that defines the first three rows of MAJOR are shown below. The entirety of MAJOR is shown after using (SELECT * FROM MAJOR;).

```
INSERT INTO MAJOR
    VALUES ('59452071','22302','Mechanical Engineering');
INSERT INTO MAJOR
    VALUES ('13117093','68377','Applied Mathematics');
INSERT INTO MAJOR
    VALUES ('30778656','61093','Data Science');
```

	MAJOR_ID	DEPT_ID	MAJOR_NAME
1	59452071	22302	Mechanical Engineering
2	13117093	68377	Applied Mathematics
3	30778656	61093	Data Science
4	21873423	73944	Computer Science
5	26760720	61093	Psycology
6	12623891	61093	Business
7	70817756	73944	Political Science
8	31622288	73944	Biology
9	84088571	44908	Nursing
10	56026134	68377	Communications
11	63867378	44908	English
12	75055821	10276	Economics
13	30279975	61093	History
14	35086469	89618	Art
15	99269298	13715	Chemistry

ENROLL

The code that defines the first three rows of ENROLL are shown below. Part of ENROLL is shown after using (SELECT * FROM ENROLL;).

```
INSERT INTO ENROLL
    VALUES ('4522457','38992954','C');
INSERT INTO ENROLL
    VALUES ('8541208','73706075','F');
INSERT INTO ENROLL
    VALUES ('8922960','74303776','D');
```

	STUD_ID	CLASS_ID	STUD_GRADE
1	8541208	73706075	F
2	8922960	74303776	D
3	8922960	47749676	A
4	2058756	66107047	D
5	3254257	73706075	A
6	2978347	34639864	F
7	8541208	78780712	F
8	8541208	74303776	C
9	9312021	95228249	A
10	5279348	66102158	B
11	3254257	78780712	B
12	8922960	66107047	F

STUDENT_DEG_MAJOR

The code that defines the first three rows of STUDENT_DEG_MAJOR are shown below. The entirety of STUDENT_DEG_MAJOR is shown after using (SELECT * FROM STUDENT_DEG_MAJOR;).

```
INSERT INTO STUDENT_DEG_MAJOR
    VALUES('2978347','11564266','30279975','1-oct-19');
INSERT INTO STUDENT_DEG_MAJOR
    VALUES('2058756','14855520','31622288','4-sep-10');
INSERT INTO STUDENT_DEG_MAJOR
    VALUES('8922960','11564266','63867378','5-sep-18');
```

	STUD_ID	DEGREE_ID	MAJOR_ID	DECLARE_DATE
1	2978347	11564266	30279975	01-OCT-19
2	2058756	14855520	31622288	04-SEP-10
3	8922960	11564266	63867378	05-SEP-18
4	5279348	11564266	56026134	07-OCT-18
5	8541208	14855520	30279975	09-FEB-17
6	8922960	38048866	56026134	11-FEB-19
7	5279348	11564266	30279975	11-FEB-19
8	8922960	79969969	84088571	03-NOV-16
9	7368886	11564266	26760720	19-MAY-15
10	7368886	79969969	84088571	12-JAN-14
11	8541208	38048866	63867378	17-AUG-19
12	8541208	14855520	13117093	17-NOV-17
13	2058756	14855520	21873423	19-JAN-16
14	2058756	79969969	13117093	09-FEB-17
15	8541208	14855520	12623891	19-JAN-15

At this point, all the tables are filled with rows and are ready to be queried.

Querying the Database

To test my knowledge of queries, I will try to answer a few questions.

1) Which students are using their Cabrini email?

```
SELECT STUD_FNAME, STUD_LNAME
FROM STUDENT
WHERE STUD_EMAIL LIKE '%cabrini.edu';
```

	STUD_F...	STUD_LNAME
1	Kevin	Nolasco

For this query, I use the LIKE function that checks whether an email address has the specified pattern. I also used the “%” wildcard which represents any number of characters.

2) Which majors are offered by the departments?

```
-- majors offered by department
SELECT DEPT_NAME, MAJOR_NAME
FROM DEPARTMENT D INNER JOIN MAJOR M
ON D.DEPT_ID = M.DEPT_ID
ORDER BY D.DEPT_NAME, M.MAJOR_NAME ASC;
```

	DEPT_NAME	MAJOR_NAME
1	Arts	Art
2	Astronomy	Applied Mathematics
3	Astronomy	Communications
4	Biology	Biology
5	Biology	Computer Science
6	Biology	Political Science
7	Chemistry	Chemistry
8	Mathematics	Mechanical Engineering
9	Philosophy	Economics
10	Physics	Business
11	Physics	Data Science
12	Physics	History
13	Physics	Psycology
14	Sociology	English
15	Sociology	Nursing

This query uses an inner join so that the department name and the major name can be accessed. This query also uses aliases to reference the different tables by a single letter. Finally, we use ORDER BY to have our results presented in alphabetical order by department name and then major name. (As you can see, some of the department/major combinations don't really make sense in the real world. For example, we have the philosophy department offering a major in economics. This is because my python script randomly assigned a major to a department. The purpose of this project is to familiarize myself with SQL syntax, so I am not too worried about this.)

3) Who is the youngest student and what is their major?

```
-- the major of the youngest student
SELECT STUD_FNAME, STUD_LNAME, ROUND((SYSDATE - STUD_DOB)/365) AS AGE, MAJOR_NAME
FROM STUDENT S INNER JOIN STUDENT_DEG_MAJOR SDM USING(STUD_ID)
INNER JOIN MAJOR M USING (MAJOR_ID)
WHERE S.STUD_DOB = (SELECT MAX(STUD_DOB)
FROM STUDENT)
ORDER BY M.MAJOR_NAME ASC;
```

	STUD_F...	STUD_LNAME	AGE	MAJOR_NAME
1	Carlos	Gentile	22	Applied Mathematics
2	Carlos	Gentile	22	Biology
3	Carlos	Gentile	22	Computer Science

This query was a bit challenging. First, finding the minimum age means finding the maximum date of birth. Oracle views dates as number of days, therefore the max number of days is the youngest person. To find the youngest person, I used a nested query to return the youngest person's ID and a WHERE to match that student's ID with the outer query. I needed to use two inner joins. The first was between STUDENT and STUDENT_DEG_MAJOR so that I can have access to the majors of the students. The next inner join was between STUDENT_DEG_MAJOR and MAJOR so that I can have access to the major name. Finally, Carlos Gentile is triple majoring! So, I ordered the results by the major name.

4) How many classes is each instructor teaching?

```
-- number of classes each instructor is teaching
SELECT INST_FNAME "FIRST NAME", INST_LNAME "LAST NAME", COUNT(CLASS_ID) "NUMBER OF CLASSES"
FROM INSTRUCTOR I INNER JOIN CLASSES C USING (INST_ID)
GROUP BY C.CLASS_ID, I.INST_FNAME, I.INST_LNAME, INST_FNAME, INST_LNAME
ORDER BY I.INST_FNAME;
```

	FIRST NAME	LAST NAME	NUMBER OF CLASSES
1	Barry	Johnson	1
2	Danielle	Rodriguez	1
3	Ester	Nolasco	1
4	Harold	Garcia	1
5	John	Palacios	1
6	Julien	Williams	1
7	Marcus	Martinez	1
8	Maria	Jones	1
9	Rose	Miller	1
10	Steven	Davis	1

This query uses the INNER JOIN between INSTRUCTOR and CLASSES. Here we use the aggregate function COUNT to count the groups of classes and instructors. In this database, each instructor only teaches one class.

5) Who is the highest paid instructor and what department are they in?

```
-- highest paid instructor and the department they are in
SELECT INST_FNAME, INST_LNAME, INST_SALARY, DEPT_NAME
FROM INSTRUCTOR I INNER JOIN DEPARTMENT USING (DEPT_ID)
WHERE I.INST_SALARY = (SELECT MAX(INST_SALARY) FROM INSTRUCTOR);
```

	INST_FNAME	INST_LNAME	INST_SALARY	DEPT_NAME
1	Barry	Johnson	140031	Sociology

This query uses an INNER JOIN between INSTRUCTOR and DEPARTMENT using the common attribute DEPT_ID. This query also uses a nested query to find the highest salary.

6) List all departments and the department heads

```
-- departments and their heads

SELECT DEPT_NAME, (INST_FNAME || ' ' || INST_LNAME) AS "DEPARTMENT HEAD"
FROM DEPARTMENT D INNER JOIN INSTRUCTOR USING(INST_ID)
ORDER BY D.DEPT_NAME;
```

	DEPT_NAME	DEPARTMENT HEAD
1	Arts	Marcus Martinez
2	Astronomy	Ester Nolasco
3	Biology	Maria Jones
4	Chemistry	Rose Miller
5	Humanities	Danielle Rodriguez
6	Mathematics	Harold Garcia
7	Philosophy	John Palacios
8	Physics	Steven Davis
9	Sociology	Barry Johnson

This query uses an INNER JOIN between INSTRUCTOR and DEPARTMENT on their common attribute DEPT_ID. The department head's name is concatenated using the CONCAT symbol "||". The || symbol is used twice to add a space between the first and last names. Finally, the output is ordered by department name.

7) List the course names and the semester/year they are offered.

```
-- course names and the semester/year they are offered.
SELECT SEMESTER_NAME, EXTRACT(YEAR FROM SEMESTER_START_DATE) AS YEAR, COURSE_NAME
FROM COURSE CO INNER JOIN CLASSES CL USING (COURSE_ID)
INNER JOIN SEMESTER S USING (SEMESTER_ID)
ORDER BY 2, S.SEMESTER_NAME, CO.COURSE_NAME ASC;
```

	SEMESTER_NAME	YEAR	COURSE_NAME
1	Summer	2011	Advanced Sociology
2	Winter	2011	Advanced Database Management
3	Summer	2013	Advanced Machine Learning
4	Summer	2013	Advanced Probability
5	Summer	2016	Advanced Statistics
6	Winter	2019	Advanced Linear Algebra
7	Winter	2019	Intro To Differential Equations
8	Fall	2021	Advanced R
9	Fall	2021	Intro To Philosophy
10	Spring	2021	Intro To Python

This query uses two INNER JOIN. The first is between COURSE and CLASSES by their common attribute COURSE_ID. The second is between the first and SEMESTER by their common attribute SEMESTER_ID. I used the EXTRACT function to get the year from the SEMESTER_START_DATE. Finally, I ordered the result by the year (which I referenced by it's index), then by semester name, then by course name.

8) Find all students that are failing, the instructor teaching the class, and the class they are enrolled in.

```
-- students failing, the instructor that is teaching, and the classes they are enrolled in

SELECT (STUD_FNAME || ' ' || STUD_LNAME) AS "STUDENT NAME", (INST_FNAME || ' ' || INST_LNAME) AS "INSTRUCTOR NAME", COURSE_NAME, STUD_GRADE
FROM STUDENT S INNER JOIN ENROLL E USING (STUD_ID)
INNER JOIN CLASSES CL USING (CLASS_ID)
INNER JOIN COURSE CO USING (COURSE_ID)
INNER JOIN INSTRUCTOR INST USING(INST_ID)
WHERE E.STUD_GRADE = 'F'
ORDER BY 1,2,3;
```

	STUDENT NAME	INSTRUCTOR NAME	COURSE_NAME	STUD_GRADE
1	Ashley Williams	Ester Nolasco	Advanced R	F
2	Ashley Williams	Rose Miller	Advanced Database Management	F
3	Ashley Williams	Steven Davis	Advanced Statistics	F
4	Daniel Johnson	Barry Johnson	Advanced Sociology	F
5	Daniel Johnson	Ester Nolasco	Advanced R	F
6	Daniel Johnson	Steven Davis	Advanced Statistics	F
7	Edward Smith	Julien Williams	Intro To Philosophy	F
8	Mitchell Mora	Marcus Martinez	Advanced Machine Learning	F
9	Richard Brown	Steven Davis	Advanced Statistics	F
10	Sydney Palacios	Julien Williams	Intro To Philosophy	F
11	Sydney Palacios	Rose Miller	Advanced Database Management	F

This query used a total of 4 INNER JOINS to join 5 tables! First, STUDENT and ENROLL were joined. Second, the previous join and CLASSES were joined. Third, the previous join and COURSES were joined. Finally, the previous and INSTRUCTOR were joined. A WHERE statement was included to narrow the search to only students that are failing a class. Also, the results were order by the student name, instructor name, then course name and all were referenced by their index.