Jai Malhi
301457742

CMPT354 - Assignment 1

1. Using CREATE TABLE, represent the statements

1) Patients attend doctors. Store attending information and date of attending. Patients (pid). Doctors (did)

- **Patient:** CREATE TABLE Patient (
  pid: CHAR(9),
  name: VARCHAR(255),
  age: INTEGER,
  address: VARCHAR(255)
  PRIMARY KEY (pid)
  );
- **Doctor:** CREATE TABLE Doctor (
  did: CHAR(9),
  PRIMARY KEY (did)
  );
- **Attends:** CREATE TABLE Attends (
  date: DATETIME,
  info: VARCHAR(255),
  pid: CHAR(9),
  did: CHAR(9),
  PRIMARY KEY (pid, did),
  FOREIGN KEY (pid) REFERENCES Patient,
  FOREIGN KEY (did) REFERENCES Doctor
  );

2) Continue 1, each patient attends doctors at most once. (merged attends due to "at most" constraint)

- **Patient:** CREATE TABLE Patient (
  pid: CHAR(9),
  did: CHAR(9),
  name: VARCHAR(255),
  address: VARCHAR(255),
  age: INTEGER,
  date: DATETIME,
  info: VARCHAR(255),
  PRIMARY KEY (pid),
  FOREIGN KEY (did) REFERENCES Doctor
  );
- **Doctor:** CREATE TABLE Doctor (
  did: CHAR(9),
  PRIMARY KEY (did)
  );

3) Continue 2, each patient attends doctors at least once (merged attends due to "at most" constraint)

- **Patient:** CREATE TABLE Patient (
  pid: CHAR(9),
  did: CHAR(9) NOT NULL,
  name: VARCHAR(255),
  address: VARCHAR(255),
  age: INTEGER,
  date: DATETIME,
  info: VARCHAR(255),
  PRIMARY KEY (pid),
  FOREIGN KEY (did) REFERENCES Doctor
  );
- **Doctor:** CREATE TABLE Doctor (

did: CHAR(9),
PRIMARY KEY (did)
);

4) Continue 1, only existing doctors can be attended by a patient

- **Patient:** CREATE TABLE Patient (
  pid: CHAR(9),
  name: VARCHAR(255),
  address: VARCHAR(255),
  age: INTEGER,
  PRIMARY KEY (pid),
  );
- **Doctor:** CREATE TABLE Doctor (
  did: CHAR(9),
  PRIMARY KEY (did)
  );
- **Attends:** CREATE TABLE Attends (
  date: DATETIME,
  info: VARCHAR(255),
  pid: CHAR(9),
  did: CHAR(9),
  PRIMARY KEY (pid, did),
  FOREIGN KEY (pid) REFERENCES Patient,
  FOREIGN KEY (did) REFERENCES Doctor
  );

5) Continue 1, every doctor must be attended by a patient.

- **Patient:** CREATE TABLE Patient (
  pid: CHAR(9),
  name: VARCHAR(255),
  address: VARCHAR(255),
  age: INTEGER,
  PRIMARY KEY (pid),
  );
- **Doctor:** CREATE TABLE Doctor (
  did: CHAR(9),
  pid: CHAR(9),
  PRIMARY KEY (did, pid),
  FOREIGN KEY (pid) REFERENCES Patient
  );
- **Attends:** CREATE TABLE Attends (
  date: DATETIME,
  info: VARCHAR(255),
  pid: CHAR(9),
  did: CHAR(9),
  PRIMARY KEY (pid, did),
  FOREIGN KEY (pid) REFERENCES Patient,
  FOREIGN KEY (did) REFERENCES Doctor
  );

6) Continue 1, each of (Name, Address) and (Name, Age) uniquely identifies a patient.

- **Patient:** CREATE TABLE Patient (
  pid: CHAR(9),
  name: VARCHAR(255),
  address: VARCHAR(255),

age: INTEGER,
PRIMARY KEY (pid),
UNIQUE (name, address),
UNIQUE (name, age)
);

- **Doctor:** CREATE TABLE Doctor (
  did: CHAR(9),
  PRIMARY KEY (did)
  );
- **Attends:** CREATE TABLE Attends (
  date: DATETIME,
  info: VARCHAR(255),
  pid: CHAR(9),
  did: CHAR(9),
  date: DATETIME,
  PRIMARY KEY (pid, did),
  FOREIGN KEY (pid) REFERENCES Patient,
  FOREIGN KEY (did) REFERENCES Doctor
  );

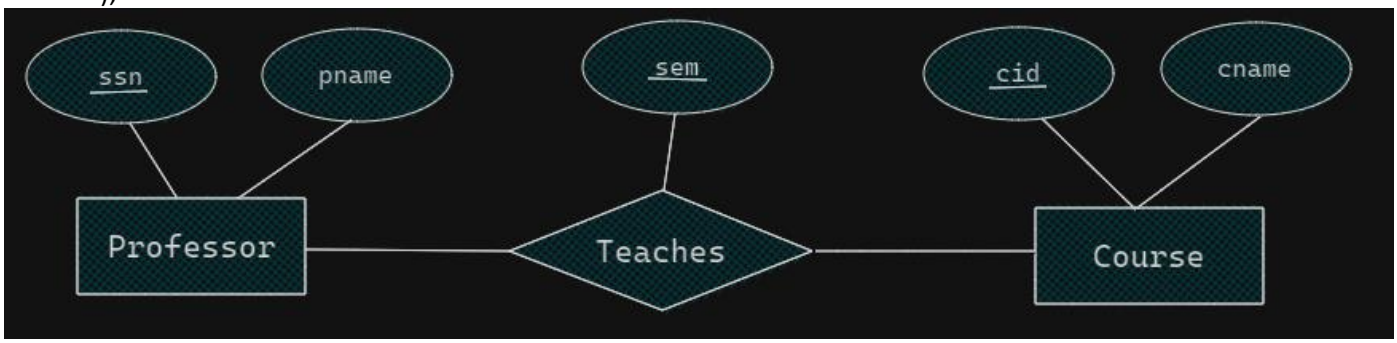
==========

2. A university database contains information about professors (identified by SSN) and courses (identified by courseid). Professors teach courses; each of the following situations concerns the Teaches relationship set. For each situation, draw an ER diagram that describes it (if no further constraints hold) and using CREATE TABLE to model the information in the ER diagram

---

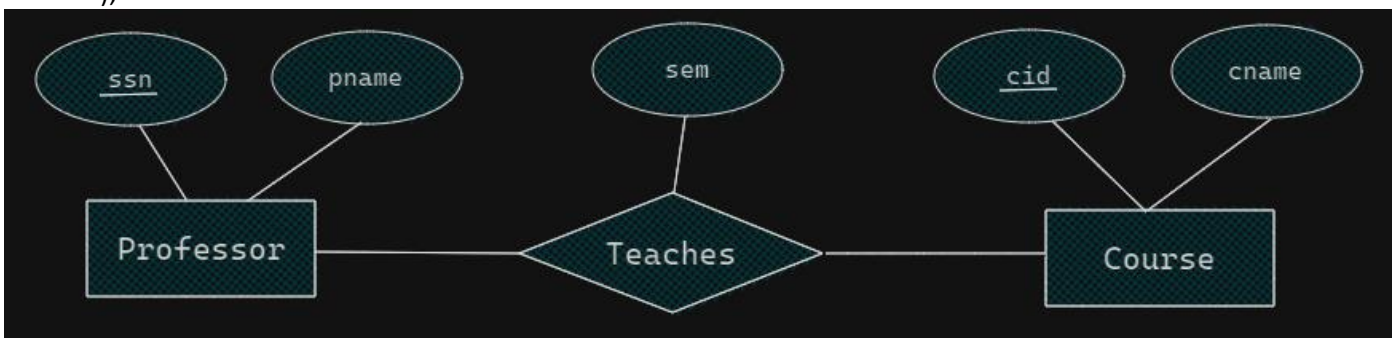
1) Professors can teach the same course in several semesters, and each offering must be recorded

- **Professor:** CREATE TABLE Professor (
  ssn: CHAR(9),
  pname: VARCHAR(255),
  PRIMARY KEY (ssn)
  );
- **Course:** CREATE TABLE Course (
  cid: VARCHAR(7),
  cname: VARCHAR(255),
  PRIMARY KEY (cid)
  );
- **Teaches:** CREATE TABLE Teaches (
  sem: CHAR(4),
  ssn: CHAR(9),
  cid: VARCHAR(7),
  PRIMARY KEY (sem, ssn, cid),
  FOREIGN KEY (ssn) REFERENCES Professor,
  FOREIGN KEY (cid) REFERENCES Course,
  );



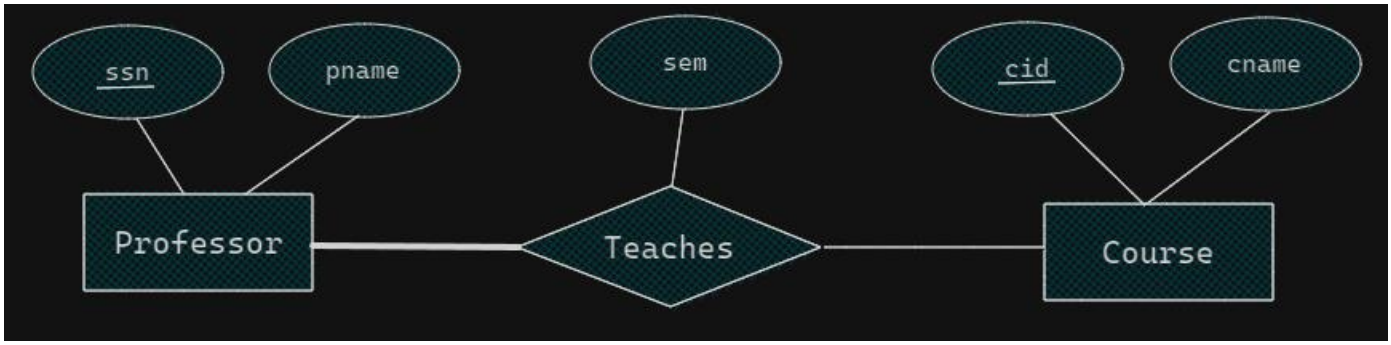
2) Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. (Assume this condition applies in all subsequent questions.)

- **Professor:** CREATE TABLE Professor (
  ssn: CHAR(9),
  pname: VARCHAR(255),
  PRIMARY KEY (ssn)
  );
- **Course:** CREATE TABLE Course (
  cid: VARCHAR(7),
  cname: VARCHAR(255),
  PRIMARY KEY (cid)
  );
- **Teaches:** CREATE TABLE Teaches (
  sem: CHAR(4),
  ssn: CHAR(9),
  cid: VARCHAR(7),
  PRIMARY KEY (ssn, cid),
  FOREIGN KEY (ssn) REFERENCES Professor,
  FOREIGN KEY (cid) REFERENCES Course,
  );



3) Every professor must teach some course.

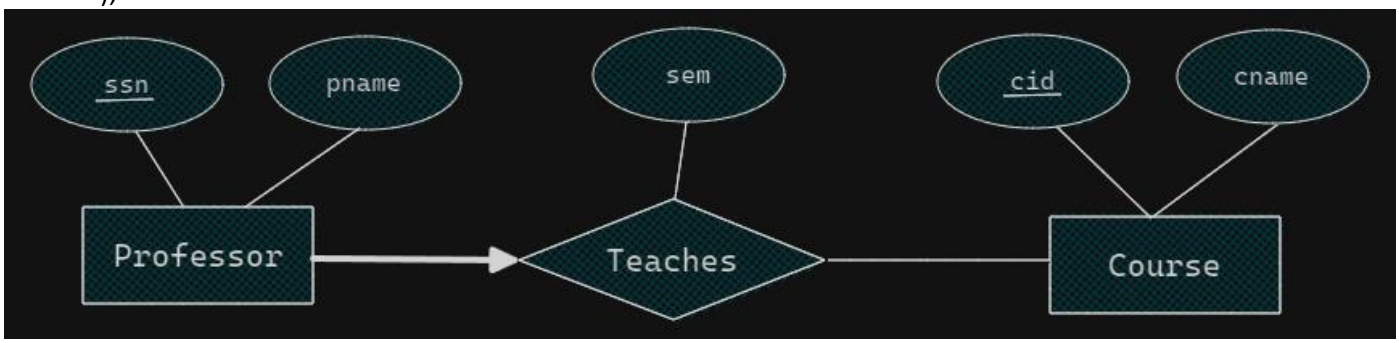
- **Professor:** CREATE TABLE Professor (
  ssn: CHAR(9),
  pname: VARCHAR(255),
  cid: VARCHAR(7) NOT NULL,
  PRIMARY KEY (ssn),
  FOREIGN KEY (cid) REFERENCES Courses
  );
- **Course:** CREATE TABLE Course (
  cid: VARCHAR(7),
  cname: VARCHAR(255),
  PRIMARY KEY (cid)
  );
- **Teaches:** CREATE TABLE Teaches (
  sem: CHAR(4),
  ssn: CHAR(9),
  cid: VARCHAR(7),
  PRIMARY KEY (ssn, cid),
  FOREIGN KEY (ssn) REFERENCES Professor,
  FOREIGN KEY (cid) REFERENCES Course,
  );



---

4) Every professor teaches exactly one course (no more, no less) (merged teaches due to "exactly one" constraint)

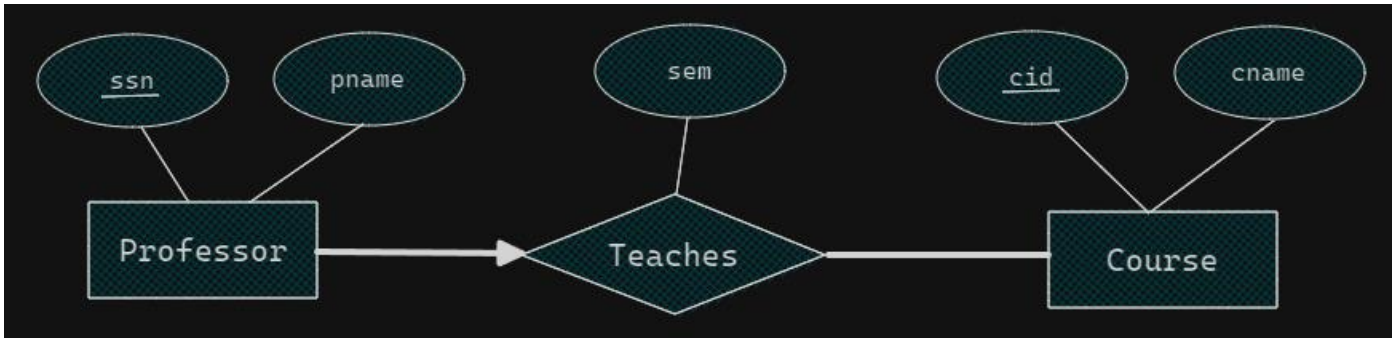
- **Course:** CREATE TABLE Course (
  cid: VARCHAR(7),
  cname: VARCHAR(255),
  PRIMARY KEY (cid)
  );
- **Professor Teaches:** CREATE TABLE Prof_Teaches (
  sem: CHAR(4),
  ssn: CHAR(9),
  pname: VARCHAR(255),
  cid: VARCHAR(7) NOT NULL,
  PRIMARY KEY (ssn),
  FOREIGN KEY (cid) REFERENCES Course,
  );



---

5) Every professor teaches exactly one course (no more, no less), and every course must be taught by some professor (merged teaches due to "exactly one" constraint)

- **Course:** CREATE TABLE Course (
  cid: VARCHAR(7),
  cname: VARCHAR(255),
  ssn: CHAR(9) NOT NULL,
  PRIMARY KEY (cid),
  FOREIGN KEY (ssn) REFERENCES Prof_Teaches
  );
- **Professor Teaches:** CREATE TABLE Prof_Teaches (
  sem: CHAR(4),
  ssn: CHAR(9),
  pname: VARCHAR(255),
  cid: VARCHAR(7) NOT NULL,
  PRIMARY KEY (ssn),
  FOREIGN KEY (cid) REFERENCES Course,
  );



---

6) Now suppose that certain courses can be taught by a team of professors jointly, but it is possible that no one professor in a team can teach the course. Model this situation.

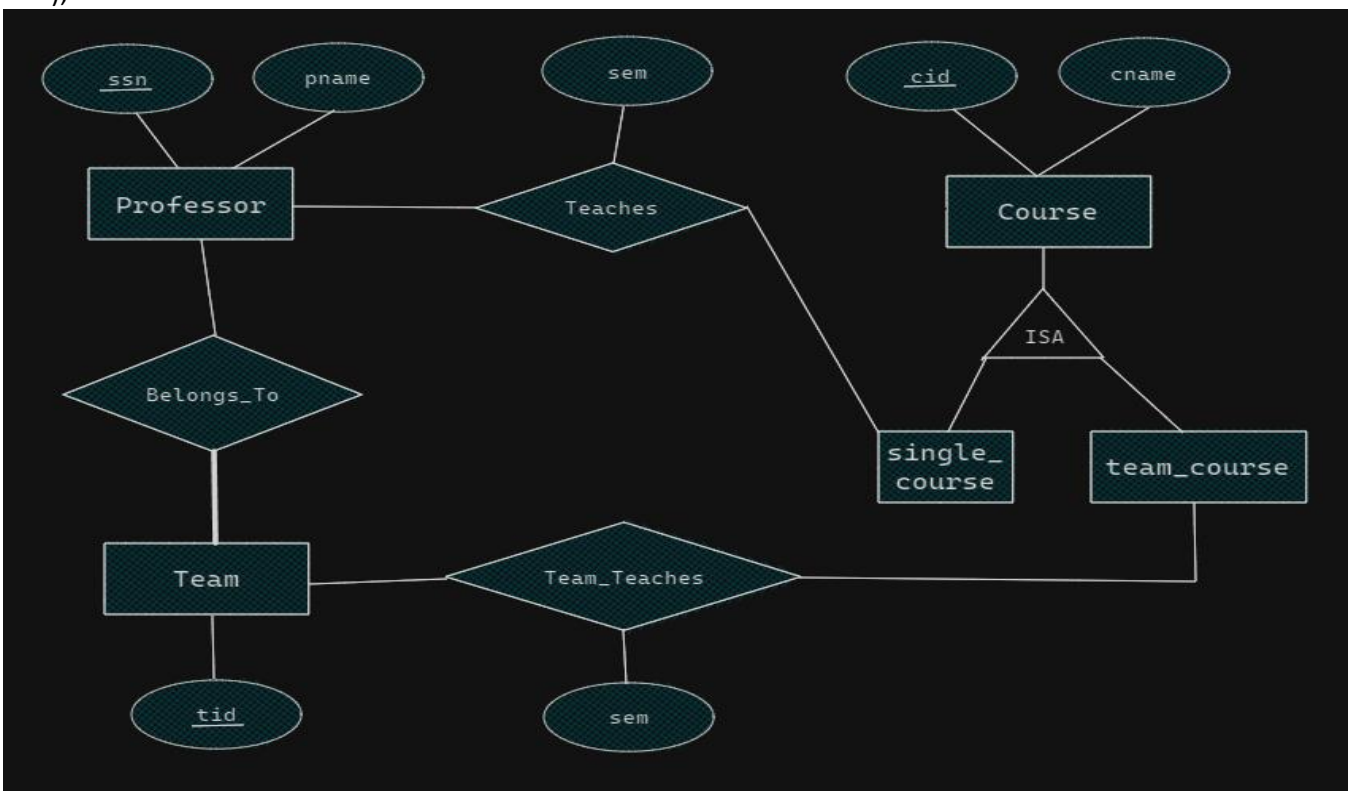
- **Professor:** CREATE TABLE Professor (
  ssn: CHAR(9),
  pname: VARCHAR(255),
  PRIMARY KEY (ssn)
  );
- **Course:** CREATE TABLE Course (
  cid: VARCHAR(7),
  cname: VARCHAR(255),
  PRIMARY KEY (cid)
  );
- **Prof. Teaches:** CREATE TABLE Teaches (
  sem: CHAR(4),
  ssn: CHAR(9),
  cid: VARCHAR(7),
  PRIMARY KEY (ssn, cid),
  FOREIGN KEY (ssn) REFERENCES Professor,
  FOREIGN KEY (cid) REFERENCES Course,
  );
- **Belongs To:** CREATE TABLE Belongs_To (
  ssn: CHAR(9),
  tid: CHAR(9),
  PRIMARY KEY (ssn, tid),
  FOREIGN KEY (ssn) REFERENCES Professor,
  FOREIGN KEY (tid) REFERENCES Team,

);

- **Team:** CREATE TABLE Team (
  tid: CHAR(9),
  PRIMARY KEY (tid),
  );
- **Team Teaches:** CREATE TABLE Team_Teaches (
  sem: CHAR(4),
  team_id: CHAR(9),
  cid: CHAR(9),
  PRIMARY KEY (tid, cid),
  FOREIGN KEY (cid) REFERENCES Team_Course,
  FOREIGN KEY (tid) REFERENCES Team
  );
- **Team Course:** CREATE TABLE Team_Course (
  cid: CHAR(9),
  PRIMARY KEY (cid)

);

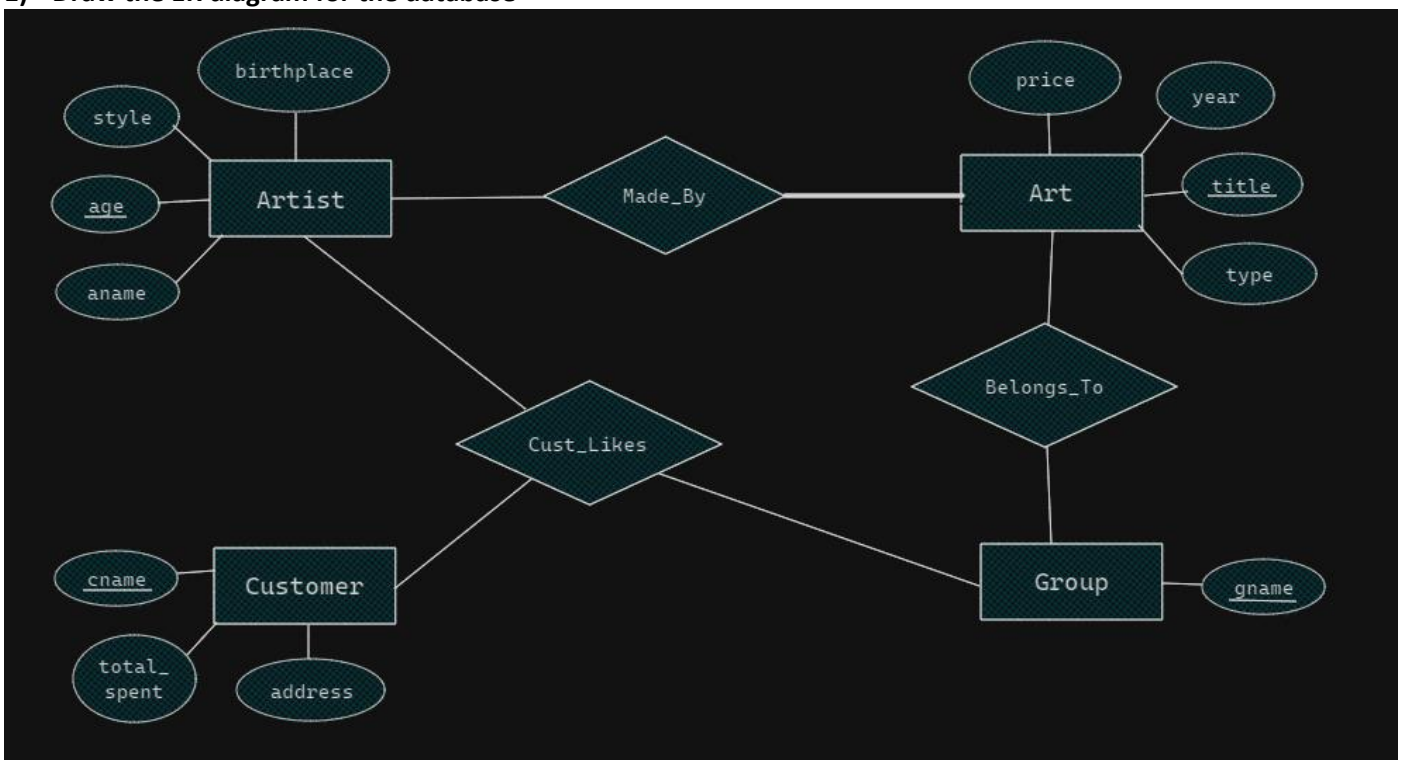
- **Single Course:** CREATE TABLE Single_Course (
  cid: CHAR(9),
  PRIMARY KEY (cid)
  );



==========

3. You are asked to set up a database, ArtBase, for art galleries. This database will capture all the information that galleries need.

1) Draw the ER diagram for the database



2) Represent the data using CREATE TABLE statements

- **Artist:** CREATE TABLE Artist (
  birthplace: VARCHAR(255),
  aname: VARCHAR(255),
  style: VARCHAR(255),
  age: INTEGER,
  PRIMARY KEY(aname)
  );
- **Made By:** CREATE TABLE Made_By (
  aname: VARCHAR(255),
  title: VARCHAR(255),
  PRIMARY KEY(aname, title),
  FOREIGN KEY (aname) REFERENCES Artists,
  FOREIGN KEY (title) REFERENCES Art
  );
- **Art:** CREATE TABLE Art(
  title: VARCHAR(255) NOT NULL,
  type: VARCHAR(255) NOT NULL,
  year: INTEGER NOT NULL,
  price: FLOAT NOT NULL,
  PRIMARY KEY(title)
  );
- **Group:** CREATE TABLE Group (
  gname: VARCHAR(255),
  PRIMARY KEY(gname)
  );
- **Belongs to:** CREATE TABLE Belongs_to (
  gname: VARCHAR(255),
  title: VARCHAR(255),
  PRIMARY KEY(gname, title),
  FOREIGN KEY(gname) REFERENCE Group,
  FOREIGN KEY(title) REFERENCE Art
  );
- **Customer:** CREATE TABLE Customer (
  total_spent: FLOAT,
  address: VARCHAR(255),
  cname: VARCHAR(255),
  PRIMARY KEY(cname)
  );
- **Customer Likes:** CREATE TABLE Cust_Likes (
  gname: VARCHAR(255),
  aname: VARCHAR(255),
  cname: VARCHAR(255),
  PRIMARY KEY(gname, aname, cname),
  FOREIGN KEY(gname) REFERENCE Group,
  FOREIGN KEY(cname) REFERENCE Customers,
  FOREIGN KEY(aname) REFERENCE Artists
  );

3) If Artwork as a weak entity set with the partial key title and the owner entity set Artist, describe the changes needed in (1) and (2)

- The ER diagram would be changed as we are using a weak entity, the primary key of *Art* would change to include *title* and *aname* (getting *aname* from *Artist*). Further, *title* would now become a partial key which means it would have a dotted underline (opposed to solid). Lastly, we need to bold *Made_By, Art,* and *Belongs_To* to signify the weak entity
- The CREATE TABLE would be altered following the changes made to the ER diagram as mentioned:
  - **Art:** CREATE TABLE Art(
    title: VARCHAR(255),
    aname: VARCHAR(255),
    type: VARCHAR(255),
    year: INTEGER,
    price: FLOAT,
    PRIMARY KEY(aname, title),
    FOREIGN KEY (aname) REFERENCES Artists
    );
  - **Belong to:** CREATE TABLE Belong_to(
    title: CHAR(20),
    aname: CHAR(20),
    gname: CHAR(20),
    PRIMARY KEY (title, aname, gname),
    FOREIGN KEY (title, aname) REFERENCES Art,
    FOREIGN KEY (gname) REFERENCES Group
    );