1 (16 points) Equivalences

For each of the following pairs of queries (in relational algebra or SQL), you will write the contents of two database instances. The databases have the following schemas:

```
A(a int, b int);
B(a int, b int)
```

(2 Points) The first database instance should be populated with one or more rows so that Q1 and Q2 output different results. If Q1 and Q2 are equivalent, then write "identical" next to the empty tables instead.

(2 Points) The second database instance should be populated with one or more rows so that Q1 and Q2 return the same results. If this is not possible, write "not possible" next to the empty tables instead.

1.1 (4 Points, 2 Per Database Instance)

Q1: $A \bowtie_a B$
Q2: $A \bowtie B$

> Different: Any non-empty table is acceptable. The queries return different schemas.
>
> Same: Not possible

1.2 (4 Points, 2 Per Database Instance)

Q1: $A \bowtie_a B$
Q2: $A \times \sigma_{B.a='a'}(B)$

> Different: Any non-empty table is acceptable. The queries return different schemas. Stating that Q2 is an error is also acceptable
>
> Same: Not possible

1.3 (4 Points, 2 Per Database Instance)

Q1: $\sigma_{\$1=\$3}(A \times B)$
Q2: `SELECT * FROM A, B WHERE A.a = B.a`

> Different: Note that SQL is multiset semantics, so database instances that allow for duplicate values will produce different query results. A contains (1,2), (1,2). B contains (1,4)
>
> Same: database instances that do not result in duplicates are acceptable. A trivial instance is one where the result is empty: A contains (1,2), B contains (3,4)

1.4 (4 Points, 2 Per Database Instance)

Q1:
```
SELECT * FROM A JOIN B ON A.a = B.a WHERE B.a = 1 or B.b = 2
```
Q2:
```
SELECT * FROM A JOIN B ON A.a = B.a WHERE B.a = 1
 UNION ALL
SELECT * FROM A JOIN B ON A.a = B.a WHERE B.b = 2
```

Different: UNION ALL preserves duplicates, thus any instances where the subqueries in Q2 return overlapping tables is acceptable.

Same: Any instances where the subqueries in Q2 do not overlap in records is acceptable.

2 (18 points) Entity-Relationship Models

2.1 Constraints (3 Points)

Your friend downloaded the following CSV file and shared it with you. What constraints, if any, can be inferred from this dataset? If there are none, simply write "none" and explain why in one sentence:

name	age	state
amy	18	TX
amy	18	CA
amy	18	FL
joe	20	MA
joe	20	NY

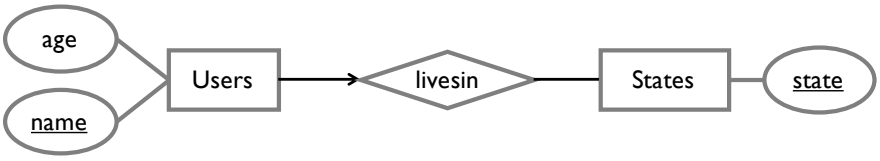
Table 1: Table for Problem 2

Constraints are a property of EVERY database instance. Constraints cannot be inferred from a single database instance.
We did not deduct points if you only statement the following constraint: primary key consists of all three attributes.

2.2 ER Constraints

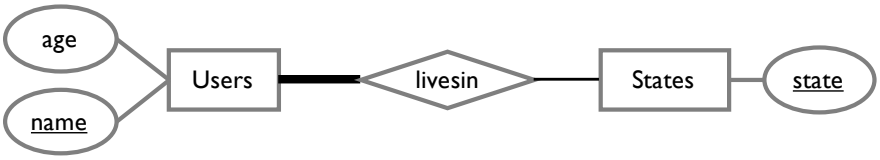
For each of the following ER diagrams, select TRUE if Table 1 satisfies the constraints depicted in the diagram, and FALSE otherwise. If FALSE, write a short sentence about why.

2.2.1 (3 Points)



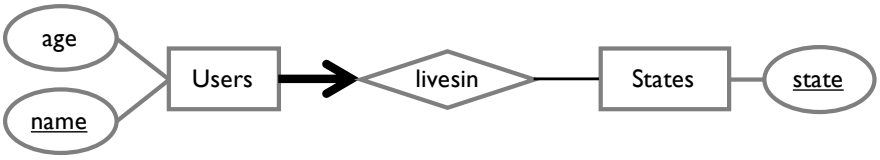
False. Violates at most one constraint.

2.2.2 (3 Points)



True

2.2.3 (3 Points)



False. Violates at most one constraint.

2.3 ER to SQL

2.3.1 (3 Points)

Translate the ER diagram from Problem 2.2.1 into SQL.

This can be translated into two tables or three. We looked for the correct primary key. Common mistakes include stating that states should be unique.

```
CREATE TABLE users_livesin(
  name text primary key,
  age int,
  state text references States
);
CREATE TABLE state(
  state text primary key
)
```

or

```
CREATE TABLE users(
  name text primary key,
  age int,
);
CREATE TABLE livesin(
  name text references users,
  state text not null references state,
  primary key (name)
)
CREATE TABLE state(
  state text primary key
)
```

2.3.2 (3 Points)

Translate the ER diagram from Problem 2.2.3 into SQL.

Can only be expressed by merging users and livesin. We looked for the correct primary key and NOT NULL constraint.

```
CREATE TABLE users_livesin(
  name text primary key,
  age int,
  state text NOT NULL references States
);
CREATE TABLE state(
  state text primary key
)
```

3 (8 points) Triggers

Consider the following database schema, and each table is empty:

```
CREATE TABLE A(a int);
CREATE TABLE B(b int);
```

In this question, we will present two implementations of triggers. Assuming the two tables are initially empty, and you run the following `INSERT` statements, you will write the final contents of `A` and `B` in the answer sheet.

```
INSERT INTO a VALUES(1);
INSERT INTO a VALUES(2);
INSERT INTO a VALUES(3);
```

3.1 Triggers

3.1.1 (4 Points, 2/table)

```
CREATE FUNCTION UDF1() RETURNS trigger
AS $$
BEGIN
  INSERT INTO b VALUES(NEW.*);
  RETURN NEW;
END;
$$ language plpgsql;
```

```
CREATE TRIGGER T1
BEFORE INSERT ON a
FOR EACH ROW
  EXECUTE PROCEDURE UDF1();
```

Both tables contain 1,2,3. This is because the UDF returns a non-null value, and the UDF inserts each new row into b.

3.1.2 (4 Points, 2/table)

```
CREATE FUNCTION UDF2() RETURNS trigger
AS $$
BEGIN
  INSERT INTO b VALUES(NEW.*);
  RETURN null;
END;
$$ language plpgsql;
```

```
CREATE TRIGGER T2
AFTER INSERT ON a
FOR EACH ROW
  EXECUTE PROCEDURE UDF2();
```

Both tables contain 1,2,3. The UDF runs AFTER the insert, so its null return value has no effect.

4 (10 points) Misc. Questions

4.1 (2 Points)

In at most 2 short sentences, describe the significance of integrity constraints in database management systems as compared to writing code to check constraints within the application.

Constraints are stored together with the data and always enforced by the database system. It is very difficult to check every piece of data that may change the database within the application code.

4.2 (2 Points)

List 2 important properties that the relational model provides that the Network/Hierarchical model does not provide. 4 words MAX for each property.

Data independence
Declarative query interface
Avoids redundancy sometimes.

4.3 (2 Points)

In at most 2 short sentences, describe the difference between `VIEW` and `WITH` in SQL.

WITH defines a (possibly recursive) temporary table that lasts for the duration of a single query. VIEWs are tables defined as a query that persists until the view is dropped, and is evaluated by rewriting references to the VIEW with its query definition.

4.4 (2 Points)

In ONE sentence, explain multiset semantics.

Extends relational algebra's set semantics to allow duplicates.

4.5 (2 Points)

Write a creative example of joins In Real Life by filling in the sentence in the answer sheet. Most creative answer (subjectively judged by the staff) gets 2 extra credit points.

Accepted any answer that is sensible and can actually be viewed as a join. A common mistake was writing a filter as the join condition. Another was to fill the blanks with words that seem similar. Another was to join individual entities rather than actual SETS of entities.

5 (14 points) Pass the SQL

The Warriors is the dominant basketball team in the National Basketball Association (NBA). Legend states that their dominance is not due to having a team of four (now five) NBA all stars, but instead due to their focus on passing and unselfish ball handling. Is this really the case? This problem will walk through an analysis to study how long players hold the ball before they pass to their teammates. When a player holds the ball, we say that the player *possesses* the ball.

Consider the following database schema, where `Players` contains information about each basketball player, `Teams` contains information about each team, and `Possessions` contains information about each time a player held the ball and how long the player possessed the ball.

```
Players(
  pid int primary key,
  tid int not null
      references Teams,
  name text not null,
  age int not null

)
```

```
Teams(
  tid int primary key,
  name text not null,
  westcoast bool not null
)
```

```
Possessions(
  id int primary key,
  pid int not null
      references Players,
  -- when the possession started
  time timestamp not null,
  -- number of seconds the player
  -- held the ball
  held int not null
)
```

5.1 Sports Never Ages (2 Points)

Write the SQL query to find the average age across all players on west coast teams (westcoast is true). The output should be the average age.

```
SELECT avg(age)
FROM Players AS P JOIN Teams AS T ON P.tid = T.tid
WHERE westcoast
```

5.2 Hold It (4 Points)

Fill in the `CREATE TABLE TimeHeld(pid, name, held` statement by writing a query that computes the average possession time for each player. The average possession time is defined as the average number of seconds that a player possesses the ball before the ball is passed to a teammate. Return the `pid` and `name` of each player, and the player's average possession time.

```
CREATE TABLE TimeHeld(pid, name, held) AS
  SELECT pid, name, avg(held)
    FROM Possessions
GROUP BY pid, name
```

5.3 Teams That Pass (4 Points)

Fill in the `CREATE TABLE TeamPasses(passer, passee)` statement so it contains the `pids` of the players that passed the ball (`passer`) and the teammate that received the pass (`passee`).

This is defined as `Possessions` where the passer that is in possession of the ball is on the same team as the passee that receives the ball, *and* if the time of the passee's possession is equal to the time of the passer's possession plus the amount of time that the passer held onto the ball. You may assume that `TimeHeld` has been correctly created and may use it in your answer if it helps.

```
CREATE TABLE TeamPasses(passer, passee) AS
  SELECT P1.pid, P2.pid
    FROM Possessions AS P1, Players AS L1,
         Possessions AS P2, Players AS L2
   WHERE P1.pid = L1.pid AND
         P2.pid = L2.pid AND
         L1.tid = L2.tid AND
         P1.time + P1.held = P2.time
```

5.4 The Control Tower (6 Points)

Fill in the `CREATE TABLE Control(tid, pid)` statement to identify the players on each team that have passed the ball to every teammate at least once. Return the team `tid` and player `pid`. You may assume that `TimeHeld` and `TeamPasses` have been correctly created and may use it in your answer if it helps.

```
CREATE TABLE Control(tid, pid) AS
 SELECT tid, pid
   FROM Players AS L1
  WHERE NOT EXISTS (
   SELECT *
     FROM Players AS L2
    WHERE L1.tid = L2.tid AND
          L1.pid != L2.pid AND
          NOT EXISTS (
            SELECT *
              FROM TeamPasses AS TP
             WHERE TP.passer = L1.pid AND
                   TP.passee = L2.pid AND
       )
```

5.5 The Passing-est (4 Points)

Let the team possession time be the average of the average possession time over all players on the team. Write the SQL query that returns the name of the team with the lowest team possession time. You may assume that `TimeHeld`, `TeamPasses`, and `Control` have been correctly created and may use it in your answer if it helps.

```
SELECT T.name
  FROM TimeHeld AS TH, Players AS L, Teams T
 WHERE TH.pid = L.pid AND T.tid = L.tid
```

1 (10 points) Terms and Definitions

(2 points each) In *at most* two short sentences each, explain the meaning of the following terms as they relate to database management systems.

1. **Entity Set**

The set of all possible entities of a given type, that all have the same set of attributes.

1 point: set of objects; 1 point: same attributes

2. **Super Key**

A set of fields in a relation, where a subset of the fields uniquely identify a tuple. This means there are more fields than are strictly necessary, which separates this from a candidate key.

1 point: uniquely identifies a tuple; 1 point: has more columns than necessary.

3. **Null**

In SQL, indicates that there is no value or a missing value for an attribute.

1 point: missing / optional / nothing; 1 point: refers to values for attributes.

4. **Integrity Constraint**

Checks on the data contained in the database that must always be true. Ensures that the data is always correct.

1 point: checks or validations; 1 point: correctness for entire database

5. **Natural Join**

A join between where the values for all fields with the same name in the two tables are equal.

1 point: same name / common attributes; 1 point: equality on the values

2 (10 points) EJBank Relational Algebra

Evan's bank stores its data in two relations, with the following SQL schema:

```
CREATE TABLE Customers(
  cid int PRIMARY KEY,
  name text,
  state text
);

CREATE TABLE Accounts(
  aid int PRIMARY KEY,
  cid int NOT NULL REFERENCES cid,
  balance real NOT NULL
);
```

1. **(2 points)** Write a relational algebra expression to compute the account ids that have balances greater than $50,000.

$\pi_{aid}\sigma_{balance>50000}(Accounts)$

2. **(4 points)** Write a relational algebra expression to compute the names of customers with balances greater than $50,000 in the state of "NY".

$\pi_{name}((\sigma_{state=NY}(Customers)) \bowtie_{cid} (\sigma_{balance>50000} Accounts))$

3. **(4 points)** Given the following values for the Account relation:

aid	cid	balance
1	102	1000.00
2	102	2000.00
3	107	2000.00
4	108	1000.00

What is the result of the following relational algebra expression?

$\rho(A, Accounts)$
$\rho(B, Accounts)$
$\pi_{A.aid,B.aid}(A \bowtie_{A.balance>B.balance} B)$

(continued on next page)

Fill in your answer in this table. Do not fill in the names for the fields. *Note*: you may or may not need all the columns and rows.

Note: Order does not matter

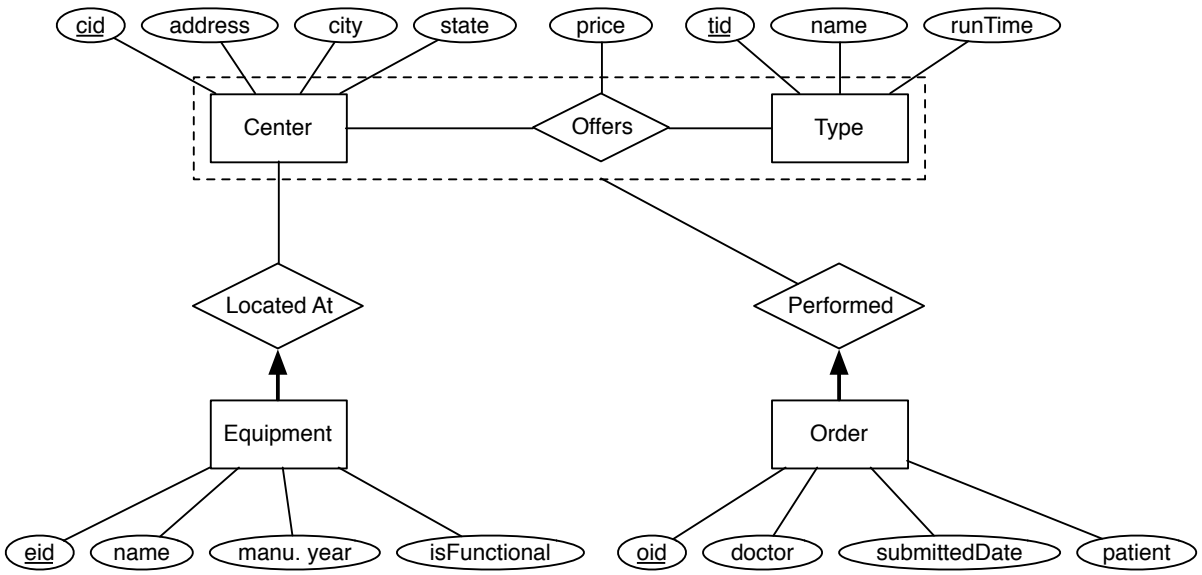
2	1				
3	1				
2	4				
3	4				

3 (20 points) Medical testing Entity-Relationship Modelling

A medical lab testing company has several testing centers all over the country. In this problem, you will design a schema to keep track of the testing centers, tests and order information. Specifically, you will need to keep track of:

1. The address, city, state and manager for each testing center.
2. The equipment at each testing center including the name of the machine, manufacturing year, and status of each machine (either functional or in repair). Note that one center may have multiple machines of the same kind.
3. The types of tests the company can perform. Tests have a name, time to run and price. The price of the test depends on the specific testing center. Some tests are only available at specific centers.
4. When an order for a test is submitted, the lab must record the doctor's name, the patient's name, and the date it was ordered. Each order is performed at a specific testing center. It must be performed at a center that offers that type of test.

Part 1: (10 points) Draw an ER diagram representing your database. Include 1-3 sentences of justification for why you drew it the way you did.



Alternatives and notes:

- We accepted basically any reasonable unique id for the entities, but it was an error to forget to specify ids. The question should have specified that it is okay to add ids if that makes things easier.
- Equipment needs either an exactly one or at most one relationship with Center. Otherwise you can have a machine at multiple testing centers, which isn't physically possible. The fact that there is a "status" attribute means that each Equipment entity represents one physical machine. The note about multiple machines of the same kind is a distraction and can effectively be ignored. Adding a quantity field isn't quite right, because then multiple machines share the same status.
- Order must have exactly one test type. A common mistake was to permit an order to be for multiple tests, or to have zero tests, which doesn't match the description.

- The price from the Test Type must be an attribute of a relationship with Center. Otherwise, you either need to duplicate the test type information for multiple centers, or the price doesn't depend on the specific Center.
- Some people chose to make things weak entities, which is acceptable. (I'd argue there are no weak entities here, but I can see the argument the other way.)

Close but incorrect alternatives:

- Performed is a ternary relationship between Type and Center. This permits tests to be ordered at centers that do not offer them.
- Order has two exactly one relationships: one with Center and one with Type. This is equivalent to the ternary relationship described above.

Part 2: (10 points) Write a SQL schema for your database. Include 1-3 sentences of justification for why you chose the tables you did.

We evaluated if you correctly translated your entity-relationship diagram into SQL. The SQL below is for the solution above. Some common mistakes:

- Equipment status needs a `CHECK` constraint or must be a boolean, to enforce that it can only take a limited set of values. We didn't care about any other constraints (except foreign keys).
- A relationship with an at least one constraint must permit the foreign key to be NULL, or must be in a separate table.
- A relationship with an exactly one constraint must be combined with the entity, and must not permit a NULL foreign key.
- When you have a relationship without constraints, it usually has a composite primary key (e.g. `Offers(cid, tid)`). When it is involved in another relationship, you must reference *that table*, and not the "base" tables (e.g. (Order `REFERENCES` Offers, not Order `REFERENCES` Center and Type). You need the database to enforce the constraint that a record exists in the relationship, not the base tables. This is the difference between a the "correct" aggregate compared to a ternary relationship between Center, Type, and Order.

```
CREATE TABLE Centers(
  cid int PRIMARY KEY,
  address text NOT NULL,
  city text NOT NULL,
  state text NOT NULL,
  manager text NOT NULL
);

CREATE TABLE Types(
  tid int PRIMARY KEY,
  name text NOT NULL,
  runTime int NOT NULL
);

CREATE TABLE Offers(
  cid int REFERENCES Centers,
  tid int REFERENCES Types,
```

```
  price real NOT NULL,
  PRIMARY KEY (cid, tid)
);

CREATE TABLE Equipment(
  eid int PRIMARY KEY,
  locationCid int NOT NULL REFERENCES Centers,
  name text NOT NULL,
  manufactureYear int NOT NULL,
  isFunctional bool NOT NULL
  -- ALTERNATIVE: status text NOT NULL,
  --              CHECK(status = 'functional' OR status = 'in repair')
);

CREATE TABLE Orders(
  oid int PRIMARY KEY,
  performedCid int NOT NULL REFERENCES Centers,
  cid int NOT NULL,
  tid int NOT NULL,
  doctor text NOT NULL,
  submittedDate date NOT NULL,
  patient text NOT NULL,
  FOREIGN KEY (cid, tid) REFERENCES Offers
);
```

4 (20 points) Wikipedia in SQL

For this question, we will use a simplified schema based on Wikipedia, shown below. Each page has a unique id, a human readable title, and the length of the page (in bytes). Links between pages are stored in the Link table. The source is the page that contains the link, and dest is the page that the link points to. As an example, a Link tuple with values (source=50, dest=100) means that page id 50 contains a link to page id 100.

```
CREATE TABLE Page(
  id int PRIMARY KEY,
  title text NOT NULL,
  length int NOT NULL
);

CREATE TABLE Link(
  source int REFERENCES Page,
  dest int REFERENCES Page,
  PRIMARY KEY (source, dest)
);
```

1. **(6 points)** Circle true or false for the following statements:

   (a) **True / False** There can be two pages with the title "Venus".

> **TRUE**: There is no unique constraint on `title`.

   (b) **True / False** Broken links may exist (a link where the source or the destination pages do not exist).

> **FALSE**: The foreign key constraints on Link ensures that both pages must exist.

   (c) **True / False** A page can only be the source of one link.

> **FALSE**: To express that constraint, Page and Link would need to be combined.

   (d) **True / False** If the page "Venus" contains a link to "Mars", deleting "Venus" will not be permitted.

> **TRUE**: The foreign key constraints on Link ensure that you can't delete the page while it is either the source or destination for any links.

   (e) **True / False** If the page "Venus" is not the source of any links, deleting it will be permitted.

> **FALSE**: It could still be the destination for links. Any foreign reference will prevent the entity from being deleted (unless `ON DELETE CASCADE` is specified).

   (f) **True / False** Renaming pages is not permitted.

> **FALSE**: There is no way to specify a constraint that forbids items from being edited. There are also no constraints on title, beyond `NOT NULL`, so you can even rename it to the same name as another page.

Write SQL queries to answer the following questions:

2. **(2 points)** What is the id and title of all pages that have titles that begin with the string "Database"?

```
SELECT id, title
FROM Page
WHERE title LIKE 'Database%';
```

3. **(2 points)** What are the titles of the 10 longest pages (in bytes)?

```
SELECT title
FROM Page
ORDER BY length DESC
LIMIT 10;
```

4. **(2 points)** What are the titles of all pages linked from the page with id 42?

```
SELECT title
FROM Page, Link
WHERE Link.source = 42 AND Page.id = Link.dest;
```

Alternative:

```
SELECT P.title
FROM Page p
WHERE p.id IN (
  SELECT dest
  FROM Link
  WHERE source = 42
);
```

5. **(4 points)** What are the titles of all pages linked from pages with the title "Database"?

```
SELECT destination.title
FROM Page source, Link, Page destination
WHERE source.title = 'Database'
    AND source.id = Link.source
    AND Link.dest = destination.id;
```

Alternative:

```
SELECT title
FROM Page
WHERE id IN (
  SELECT dest
  FROM Page, Link
  WHERE Page.title = 'Database' AND page.id = Link.source
);
```

Yet another version:

```
SELECT p1.title
FROM Page p1, Link l
WHERE p1.id = l.dest AND l.source IN (
  SELECT p2.id
  FROM Page p2
  WHERE p2.title = 'Database'
);
```

6. **(4 points)** Popularity of a page is defined as the number of incoming links (links leading to that page). What are the titles of the 5 most popular pages?

```
SELECT title
FROM Page, (
  SELECT dest, count(*) as count
  FROM Link
  GROUP BY dest
  ORDER BY count DESC
  LIMIT 5) AS Popular
WHERE Page.id = Popular.dest;
```

Note: `ORDER BY` and `LIMIT` can be applied to outer query instead of the inner query. Another version:

```
SELECT Page.title
FROM Link, Page
WHERE Link.dest = Page.id
GROUP BY Link.dest, Page.title
ORDER BY count(*) DESC
LIMIT 5;
```

1 (18 points) Terms and Definitions

(3 points each) In *at most* two short sentences each, explain the meaning of the following terms as they relate to database management systems. **Draw a box around your final answer**.

1. **Total Participation**

> Used to represent at-least-one relationships between two entities.

2. **Transaction**

> Ensures that a sequence of statements happens all at once or not at all in a single instant in time.

3. **Bag/Multi-Set Semantics**

> Duplicates are allowed in relational tables. SQL uses these semantics, thus necessitating the DISTINCT operator.
>
> Note: has nothing to do with "multiple-sets". The multi term describes the multiplicity of member elements in a set.

4. **Positional-field Notation**

> Used to unambiguously refer to attributes by their position. Useful when there is ambiguity in column names (common in intermediate query results).

5. **User Defined Functions**

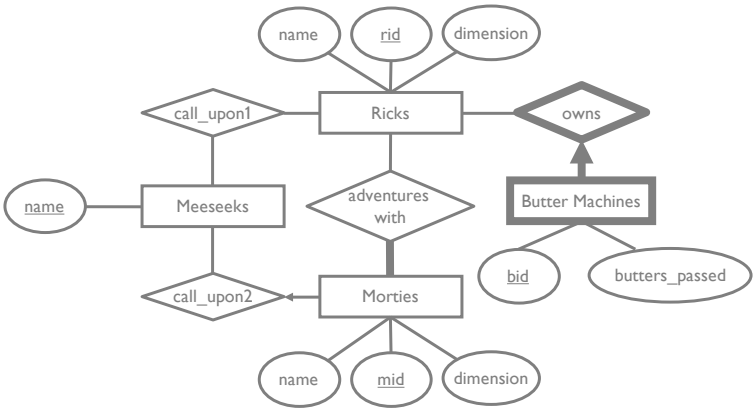
> Allows users to write custom programmatic functions that can be used in SQL queries.

6. **Physical-data Independence**

> Protects the application from changes in the physical schema of the database. A core property of declarativity.

2 (28 points) Buliting a Multi-verse using ER diagrams

Consider the following Entity-Relationship diagram that describes relationships in Rick and Morty's world. There are multiple Ricks and Morties, each from a particular dimension. Ricks go on multi-dimensional adventures with Morties. Ricks also own butter machines whose job is to pass the butter. In addition, Meeseeks are creatures that can be called upon by Ricks and/or Morties to accomplish tasks – in fact, that is their sole purpose. The description above does not describe the multiplicity of the relationships in the diagram, which is your job to understand!



1. **(12 points)** For the next 4 items, answer them based on the ER diagram above. You will get 3 points for each correct answer, -0.5 points for each incorrect answer, and 0 points for each answer left blank.

   (a) **True / False** There are Morties that grow up and live their lives without going on any adventures with a Rick.

   FALSE. There is a total-participation relationship.

   (b) **True / False** A single Meeseek can been called upon by both a Rick from dimension `C-137` and a Morty from dimension `C-137`.

   TRUE. The relationship between a Meeseek and a Rick is separate from the relationships between that Meeseek and a Morty.

   (c) **True / False** Ricks have a lot of enemies (including other Ricks) and might be exterminated, thus removing the dead Rick from the Ricks table. In that situation, does this allow the deceased Rick's Butter Machine to run away and live a long, prosperous, non-butter-passing life?

   FALSE Sadly, the Butter Machine is a weak entity and cannot be represented without its owner Rick.

   (d) Consider Morty from dimension `C-137`, let's call him $Morty_{c137}$. $Morty_{c137}$ is feeling overwhelmed about missing math class to go on adventures and has called upon as many Meeseeks as he can (according to the ER diagram). Circle the most general correct answer.

      i. $Morty_{c137}$ must have called more Meeseeks than some Ricks.

      ii. $Morty_{c137}$ must have called more Meeseeks than all Ricks.

iii. $Morty_{c137}$ must have called less Meeseeks than some Ricks.

iv. $Morty_{c137}$ must have called less Meeseeks than all Ricks.

v. None of the above.

> **iii: less Meeseeks than some Ricks.** The arrow means $Morty_{c137}$ can call at most 1 Meeseek, whereas Ricks can call as many as they want. Thus $Morty_{c137}$ must call less than *some* Ricks.

2. **(16 points)** Consider the following SQL schema, which translates of the ER diagram above into tables in the relational model.

```
CREATE TABLE Ricks(
  rid int UNIQUE,
  name text,
  dimension text
);

CREATE TABLE MortiesAdventuresWith(
  mid int NOT NULL,
  name text,
  dimension text,
  rid int REFERENCES Ricks(rid),
  mname_call_upon2 text REFERENCES Meeseeks2(name),
  PRIMARY KEY (mid)
)

CREATE TABLE ButterMachines(
  bid int,
  butters_passed int,
  check (bid is not null),
  rid int REFERENCES Ricks(rid) NOT NULL ON CASCADE delete,
  PRIMARY KEY (bid, rid)
)

CREATE TABLE Meeseeks1(
  name text UNIQUE NOT NULL,
  PRIMARY KEY (name)
);

CREATE TABLE call_upon1(
  rick_id int,
  mname text REFERENCES Meeseeks1(name),
  UNIQUE (rick_id, mname)
);

CREATE TABLE Meeseeks2(
  name text UNIQUE NOT NULL,
  PRIMARY KEY (name)
);
```

(4 points each) List and briefly explain the four most important problems that you find with this schema, in terms of how well it models the E/R diagram above. For each problem, provide a *very* brief example of data that illustrates the problem. Do not base your answer on comparing this schema with other possible schemas. Instead, just compare the schema against the E/R diagram to identify what is not captured from the diagram, etc. in the schema.

You will be graded on the importance of the problems that you identify and the extent that the problems you describe overlap. For example, if you say "table Foo is wrong" as one problem, and "relationship with Foo is a problem because Foo is wrong", then we will grade the second "problem" lower.

Braw a box around your final answer.

(a) **Problem 1:**

(b) **Problem 2:**

(c) **Problem 3:**

(d) **Problem 4:**

Any of the following were given points. Superfluous or contradictory text resulted in point reductions. Overlapping problems were given 2 points instead of 4. `CASCADE DELETE` was a typo addressed during the exam and is not a problem.

(a) Ricks doesn't have primary key

(b) Meeseeks should be a single table

(c) `call_upon1` should have primary key rickid, mname

(d) MortiesAdventuresWith.rid should be NOT NULL

(e) Primary key on MortiesAdventuresWith.mid restricts Morties to only one adventure

(f) In general, total participation cannot be expressed adequately in the relational model

(g) `call_upon1`.rick_id should reference Ricks

(h) ButterMachines may be owned by ¿1 Ricks

(i) Should have CHECK on butterspassed to ensure no negative values

3 (26 points) Save the Multi-verse with SQL

Consider the following intergalactic friendship database, consisting of two relations for `users` and `friends`. The direction of the friendship is explicitly stored, such that the friender `A` initiated the friendship and the friendee `B` accepts the friendship. This means `B` can also initiate a friendship with `A`, so that `B` is in two records – one as a friender and one as a friendee. Despite this, we say that users `X` and `Y` are friends regardless of which one is the friender and friendee. Users cannot be friends with themselves.

```
CREATE TABLE users(
  uid int PRIMARY KEY,
  username text UNIQUE NOT NULL,
  species text
);

CREATE TABLE friends(
  friender int REFERENCES users(uid),
  friendee int REFERENCES users(uid),
  when_added date,
  PRIMARY KEY (friender, friendee)
);
```

In the following 3 questions, write **a single** SQL statement that satisfies the english statement. You may use SQL features such as the `WITH` clause. Draw a box around your final answer. In the last question, you will fill in a `TRIGGER` template to express the english statement.

1. **(6 points)** `Warren-G` and `Ice-T` are `Alphabetrians` (users of the species = `Alphabetrian`). Find the usernames of all users that are friendees of both of these `Alphabetrians`.

(a) +2 points: if correct query for friends of warren-g

(b) +2 points: if correct query for friends of ice-t

(c) full points: if correct query

```
SELECT u2.username
FROM users u1, users u2, friends
WHERE u1.uid = friends.friender and
      u2.uid = friends.friendee and
      u1.username = 'Warren-G'
INTERSECT
SELECT u2.username
FROM users u1, users u2, friends
WHERE u1.uid = friends.friender and
      u2.uid = friends.friendee and
      u1.username = 'Ice-T'
```

2. **(6 points)** `Alphabetrians` are at war with `Numbericons` (users of the species = `Numbericon`) and it's time to *crunch the numbers*. We need to list our `Alphabetrian` brethren and note down the traitors (users that are friends with a `Numbericon`). Create a two column table containing all `Alphabetrians`. The first column is the username and the second column is `TRUE` if the user is a traitor, and `NULL` or `FALSE` otherwise. The names of the columns don't matter.

(a) +1 point: if correct query for alphabetrian users

(b) +2 points: if correct query for numbericon friends

(c) +1 points: if used outer join or equivalent query structure (see above)

(d) full points: if correct

(e) no deductions if spelled numbericon incorrectly

```
WITH numbericonFriends(uid) AS (
  SELECT bidirfriends.dst
  FROM (
    SELECT friender as src, friendee as dst FROM friends
    UNION
    SELECT friendee as src, friender as dst FROM friends
  ) as bidirfriends,
  users
  WHERE bidirfriends.src = users.uid and
        users.species = 'Numbericon'
)
(
  SELECT users.username, numbericonFriends.uid is not null
  FROM users LEFT OUTER JOIN numbericonFriends
  ON users.uid = numbericonFriends.uid
  WHERE users.species = 'Alphabetrian'
)
```

OR

```
WITH numbericonFriends(uid) AS (
  SELECT bidirfriends.dst
  FROM (
    SELECT friender as src, friendee as dst FROM friends
    UNION
    SELECT friendee as src, friender as dst FROM friends
  ) as bidirfriends,
  users
  WHERE bidirfriends.src = users.uid and
        users.species = 'Numbericon'
)
(
  SELECT users.username, users.uid IN (numbericonFriends)
  FROM users
  WHERE users.species = 'Alphabetrian'
)
```

3. **(8 points)** Rick is wanted by the Inter-dimensional Federation and we're going to use social media to crack down on his rebel group. Find the usernames of all users that are frienders of all of Rick's friends (Rick has username = `Rick Sanchez`)

(a) +2 point: if correct query for finding rick's friends (both as friender and friendee)

(b) +4 points: if correct double negation structure for the "frienders of ALL of Rick's friends".

(c) full points: if correct

(d) no deductions if spelled numbericon incorrectly

```
WITH rickfriends(uid) AS (
  -- rick is friendee
  SELECT friends.friender
  FROM friends JOIN users ON users.uid = friends.friendee
  WHERE users.username = 'Rick Sanchez'
```

```
  UNION
  -- rick is friender
  SELECT friends2.friendee
  FROM friends as friends2 JOIN users users2 ON users2.uid = friends2.friender
  WHERE users2.username = 'Rick Sanchez'
)

SELECT users.username
FROM users
WHERE NOT EXISTS (
  SELECT *
  FROM rickfriends
  WHERE NOT EXISTS (
    SELECT *
    FROM friends
    WHERE users.uid = friends.friender AND
          rickfriends.uid = friends.friendee
  )
)
```

4. **(6 points)** Fill in the following logical `TRIGGER` definition (not postgresql) using `PL/SQL` to prevent `Cromulons` from adding themselves (users of the species = `Cromulon`) to the `users` table.

```
CREATE TRIGGER nocromulons

_________________ INSERT ON  ____________________


FOR EACH ___________________________

  BEGIN









  END;
```

(a) +2 point: if BEFORE INSERT ON USERS

(b) +2 points: if FOR EACH ROW

(c) +2: if RETURN NULL; We accepted minor syntax errors (forget `END IF;`) and statements without the `RETURN NEW`.

```
CREATE TRIGGER nocromulons
BEFORE INSERT ON users
FOR EACH ROW
  BEGIN
    IF NEW.species = 'Cromulon'
      RETURN NULL;
    END IF;
    RETURN NEW;

  END;
```

OR

```
CREATE TRIGGER nocromulons
AFTER INSERT ON users
FOR EACH ROW [or STATEMENT]
  BEGIN
    DELETE FROM users WHERE species = 'Cromulon';
    RETURN NULL;
  END;
```

4 (10 points) Inter-dimensional Relations with Relational Algebra

Consider the database in Problem 3. Write the relational algebra query to find the usernames of users that are frienders with ALL users of species = `Alphabetrian`, but NOT frienders with species = `Numbericon`.

Draw a BOX around your final and intermediate answers – partial credit will be given to *clearly labeled* **intermediate expressions**

1. **2 points** got alphabetrians
   p(alphasfriendees(uid $\rightarrow$ friendee), $\pi_{uid}(\sigma_{species=Alphabetrian}(\text{users}))$
2. **2 points** Division for "frienders with ALL alphabetrians" correct
   p(alphas, $\pi_{friender,friendee}$(friends) / alphafriendees)
3. **2 points** Frienders of numericons
   p(numbers, $\pi_{friender}(\sigma_{species=Numbericon}(\text{friends} \bowtie_{uid=friendee} \text{users}))$
4. **1 points** Exclude numericons
   p(tmp, alphas - numbers)
5. **3 points** Get the final user name aka correct answer
   $\pi_{username}(\text{tmp} \bowtie_{uid=friender} \text{users})$

NOTE: The $\bowtie_{uid=friender}$ *users* in the final query (5) may be instead run in the definition of *alphas* and *numbers*.

5 (16 points) A =? B

Consider the relational model and two relations with the same schema, `R` and `S`. You may assume that no primary keys have been explicitly defined by the database administrator, but otherwise do not make any assumptions about the database.

```
R(a int, b int)
S(a int, b int)
```

Each row of the following table shows two queries. In the blank third column of the table write YES if the two queries are equivalent, and NO if they are not equivalent. Two queries are equivalent if they always return exactly the same result relation on all databases with the schema above.

You will receive **4 points** for each correct answer, **-1 points** for each incorrect answer, **0 points** for each answer left blank.

Q1	Q2	Equivalent?
$\pi_{R.a}((R \cup S) - S)$	$\pi_{R.a}(R) - \pi_{R.a}(R \cap S)$	No. Consider the following example: R = {(1, 2), (1, 3)}, S={(1, 2), (1,4)} Full credit given to Yes or No answers due to staff error.

$\pi_{R.b}(\sigma_{R.a=1}(R))$	`SELECT R.b`
	`FROM R`
	`WHERE R.a = 1`
		No. Q1 returns a set, Q2 returns a multiset (has duplicates)

$R \bowtie S$	$\sigma_{R.a=S.a}(R \times S)$	No. Q1 is a natural join (on a and b). Q2 is a join only on a

`SELECT * FROM R`
`WHERE R.a < ALL (`
`  SELECT a FROM S`
`)`
	`SELECT R.a, R.b`
	`FROM R JOIN S`
	`WHERE R.a < S.a`
		No. Q1 returns R rows less than ALL S.a values.
		Q2 returns R rows less than ANY S.a value.