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**

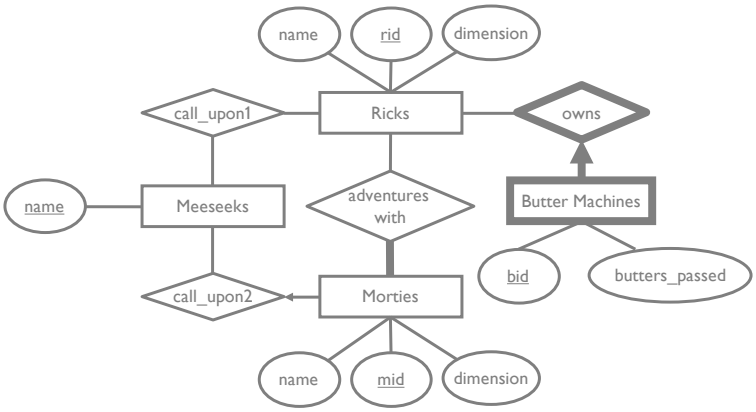
> Allows users to write custom programmatic functions that can be used in SQL queries.

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.