Assignment A-w4 KEY   Spring 2018   Name:

For the Dorm Energy Monitoring DB design, some constraints are given and some are absent. You are asked to fill in some of these absent constraints.

1. Fill Foreign Key constraints (FKs) for selected tables.

2. Fill in in-table CHECK statements for selected tables, which will typically use nested queries. Nested queries are allowed in in-TABLE CHECK statements in the SQL standard, but they are not supported on any platform (nonetheless, you might implement them one day). If these were supported, then they would be evaluated when an insertion or update is made to the table.

3. Fill in general assertions that are (in theory) evaluated whenever any change (insertion, deletion, update) is made to any table named in the assertion. Again, part of SQL standard, though not implemented on any platform.

4. Fill in trigger definitions, using SQLite syntax.

Instructions on where you are to fill in FKs, in-table CHECKs, general assertions, and triggers, are **shown in red**. There are other constraints that you are not required to fill in, but you are welcome to and you can compare them later on the key.

The additions in blue are all that is required

Dorm Energy Monitoring Application (standard tables and assertions)

/* Observer records information about those who are observing campus enery and water usage. These may be on-campus or off-campus observers. Observer associates computer network identifiers (e.g., IP addresses) with campus dormitories. DormName can be NULL, thus allowing easy recording of off-campus (or otherwise non-dorm) network identifiers of virtual visitors to dorm web-based electricity and water usage summaries. */

2 pts for this (all or nothing), or 2 points for placing FK phrase after DormName declaration (using syntax found here by following column-def, then column-constraint, then foreign-key-clause)

```
CREATE TABLE Observer (
        NetworkID CHAR(20),
        DormName VARCHAR(35),          /* Corresponds to a DormName in Dorm */
        PRIMARY KEY (NetworkID),
        FOREIGN KEY (DormName) REFERENCES Dorm ON DELETE CASCADE ON UPDATE CASCADE
);  /* Add a FOREIGN KEY constraint that will cause a DELETE or UPDATE in Dorm (of a row with a matching
       DormName) to CASCADE to Observer. */
```

/* A record of visits to a Dorm's (often assembled, on demand) Webpage, which displays statistics on electricity and water usage */

```
CREATE TABLE DormWebPage (
        DWPageID INTEGER,
        CreateDate DATE NOT NULL, /* NOT NULL indicates field cannot be NULL in any record */
        CreateTime TIME NOT NULL,
        ObserverNetworkID CHAR(20) NOT NULL,   /* Corresponds to a NetworkID in Observer */
        DormName VARCHAR(35) NOT NULL,         /* Corresponds to a DormName in Dorm */
        PRIMARY KEY (DWPageID),
        FOREIGN KEY (ObserverNetworkID) REFERENCES Observer (NetworkID)
              ON DELETE NO ACTION ON UPDATE CASCADE,
        FOREIGN KEY (DormName) REFERENCES Dorm
              ON DELETE NO ACTION ON UPDATE CASCADE
); /* Add a FOREIGN KEY constraint that will block (i.e., prevent) a Dorm from being deleted if there is a
      tuple in DormWebPage with a matching DormName. Define the same FK to cascade an update in Dorm
      to DormWebPage. */
```

2 pts for this (all or nothing), 1 of which is for NO ACTION or RESTRICT

/* Dorms can be high res and low res, and tables for each subtype have an FK reference to Dorm, which contains the common information that is inherited for each type of dorm. Dorms are also FK referenced by a number of other tables. */

```
CREATE TABLE Dorm (
        DormName VARCHAR(35),
        MaxOccupancy SMALLINT,
        PRIMARY KEY (DormName)
        CHECK ((DormName IN (SELECT DormName FROM HighResDorm) UNION
                                        (SELECT DormName FROM LowResDorm))
);
```

/* A table of time-stamped outdoor temperatures (required) and light conditions (optional) */

```
CREATE TABLE AmbientValues (
        AmbientReadingsDate DATE,
        AmbientReadingsTime TIME,
        AmbientTemp TINYINT NOT NULL,
        AmbientLight CHAR(2),
        PRIMARY KEY (AmbientReadingsDate, AmbientReadingsTime)
);
```

/* Every high res dorm is also a dorm */

```
CREATE TABLE HighResDorm (
        DormName VARCHAR(35),      /* Corresponds to a DormName in Dorm */
        StartDate DATE,            /* Date at which it became a HighResDorm */
        PRIMARY KEY (DormName),
        FOREIGN KEY (DormName)  REFERENCES Dorm
           ON DELETE CASCADE  ON UPDATE CASCADE
        CHECK (DormName NOT IN (SELECT DormName FROM LowResDorm))
); /* Add an in-table CHECK that ensures a DormName found in HighResDorm is also found in Dorm */
```

2 pts for in-table CHECK as written; other answers may be possible

```
/* A LowRes dorm is assumed to have a unique (NOT NULL) electricity sensor, but the definition
allows water sensors to be shared across dorms (not unique) and none at all (allowed NULL) */

CREATE TABLE LowResDorm (
	DormName VARCHAR(35),		/* Corresponds to a DormName in Dorm */
	StartDate DATE,			/* Date at which it became a LowResDorm */
	LRElecSensorID INTEGER NOT NULL,
	UNIQUE(LRElecSensorID),  /* UNIQUE indicates a key; typically implies NOT NULL */
	LRElecSensorOnLineDate DATE,
	LRWaterSensorOnLineDate DATE,
	LRWaterSensorID INTEGER,
	PRIMARY KEY (DormName),
	FOREIGN KEY (DormName)  REFERENCES Dorm
		ON DELETE CASCADE  ON UPDATE CASCADE
	CHECK (DormName NOT IN (SELECT DormName FROM HighResDorm))
);
```

Since DormName is also the PK of this table, it must be NOT NULL, so an action of SET NULL would cause an error.

```
/* For example, FloorNum = 3 and DormName = McGill is 3rd floor of McGill */

CREATE TABLE Floor (
        DormName VARCHAR(35),  /* Corresponds to a DormName in HighResDorm */
        FloorNum TINYINT,
        MaxOccupancy SMALLINT,
        PRIMARY KEY (DormName, FloorNum),
        FOREIGN KEY (DormName)  REFERENCES HighResDorm ON DELETE CASCADE  ON UPDATE CASCADE
);
```

```
/* Similar meanings as DormWebPage, but for high res case */

CREATE TABLE FloorWebPage (
        DormName VARCHAR(35),    /* Corresponds to DormName in Floor */
        FloorNum TINYINT,        /* Corresponds to FloorNum in Floor */
        FWPageID INTEGER,
        CreateDate DATE NOT NULL,
        CreateTime TIME NOT NULL,
        ObserverNetworkID CHAR(20) NOT NULL,   /* Corresponds to NetworkID in Observer */
        PRIMARY KEY (FWPageID),
        FOREIGN KEY (ObserverNetworkID)  REFERENCES Observer (NetworkID),
              ON DELETE NO ATION ON UPDATE CASCADE, /* block deletes if Network ID has observation records */
        FOREIGN KEY (DormName, FloorNum) REFERENCES Floor
              ON DELETE NO ACTION ON UPDATE CASCADE  /* block deletes if floor has observation records */
);
```

```
/* Definition allows multiple sensors per floor (thus, DormName,Floor not required UNIQUE) */

CREATE TABLE HRElecSensor (
        DormName VARCHAR(35) NOT NULL, /* Corresponds to DormName in Floor */
        FloorNum TINYINT NOT NULL,          /* Corresponds to FloorNum in Floor */
        HRElecSensorID INTEGER,
        HRElecSensorOnLineDate DATE,
        PRIMARY KEY (HRElecSensorID),
        FOREIGN KEY (DormName, FloorNum) REFERENCES Floor
                ON DELETE CASCADE  ON UPDATE CASCADE
);



/* If you bother to record a reading, the value should be NOT NULL (perhaps coupled
    special value(e.g., -999)  indicating not read because not functional sensor */

CREATE TABLE HREReading (
        HRElecSensorID INTEGER,   /* Corresponds to HRElecSensorID in HRElecSensor */
        HREReadingDate DATE,
        HREReadingTime TIME,
        HREValue INTEGER NOT NULL,
        PRIMARY KEY (HRElecSensorID, HREReadingDate, HREReadingTime),
        FOREIGN KEY (HRElecSensorID) REFERENCES HRElecSensor
                ON DELETE NO ACTION /* if readings associated with a sensor, then block delete */
                ON UPDATE CASCADE
);
```

/* As with elect sensors in high res case, definition allows multiple sensors per floor (thus, DormName,Floor not required UNIQUE) */

```
CREATE TABLE HRWSensor (
        DormName VARCHAR(35) NOT NULL,  /* Corresponds to DormName in Floor */
        FloorNum TINYINT NOT NULL,           /* Corresponds to FloorNum in Floor */
        HRWaterSensorID INTEGER,
        HRWaterSensorOnLineDate DATE,
        PRIMARY KEY (HRWaterSensorID),
        FOREIGN KEY (DormName, FloorNum) REFERENCES Floor
                ON DELETE CASCADE  ON UPDATE CASCADE
); /* Write a Foreign Key constraint ensures that every (DormName, FloorNum) pair in HRWSensor is
      associated with exactly one tuple of Floor. Cascade on both deletes and updates. */
```

2 pts all or nothing; 0 points for two separate FKs for DormName and FloorNum individually

/* Time-stamped readings are associated with sensors*/

```
CREATE TABLE HRWReading (
        HRWaterSensorID INTEGER,  /* Corresponds to HRWaterSensorID in HRWaterSensor */
        HRWReadingDate DATE,
        HRWReadingTime TIME,
        HRWValue INTEGER NOT NULL,
        PRIMARY KEY (HRWaterSensorID, HRWReadingDate, HRWReadingTime),
        FOREIGN KEY (HRWaterSensorID) REFERENCES HRWaterSensor
                ON DELETE NO ACTION  /* if readings associated with a sensor, then block delete */
                ON UPDATE CASCADE
);
```

```
/*  The earlier definition of LowResDorm indicates exactly one sensor per low res dorm */

CREATE TABLE LREReading (
        DormName VARCHAR(35),   /* Corresponds to DormName in LowResDorm */
        LREReadingDate DATE,
        LREReadingTime TIME,
        LREValue INTEGER NOT NULL,
        PRIMARY KEY (DormName, LREReadingDate, LREReadingTime),
        FOREIGN KEY (DormName) REFERENCES LowResDorm
               ON DELETE NO ACTION  /* if readings associated with a low res dorm, then block delete */
               ON UPDATE CASCADE
);
```

```
/* The earlier definition of LowResDorm indicates at most one water sensor per low res dorm  */

CREATE TABLE LRWReading (
        DormName VARCHAR(35),   /* Corresponds to DormName in LowResDorm */
        LRWReadingDate DATE,
        LRWReadingTime TIME,
        LRWValue INTEGER NOT NULL,
        PRIMARY KEY (DormName, LRWReadingDate, LRWReadingTime)
        FOREIGN KEY (DormName) REFERENCES LowResDorm
               ON DELETE NO ACTION  /* if readings associated with a sensor, then block delete */
               ON UPDATE CASCADE
);
```

/* Write a general assertion that ensures that all Dorms are Low Res or High Res */

```
CREATE ASSERTION CompleteCoverOverLowAndHighRes (
      CHECK (NOT EXISTS (SELECT D.DormName FROM Dorm D
                           EXCEPT
                          SELECT LRD.DormName FROM LowResDorm LRD)
                           EXCEPT
                          SELECT HRD.DormName FROM HighResDorm HRD
                         ))); /* OR PERHAPS */

CREATE ASSERTION CompleteCoverOverLowAndHighRes (
      CHECK (NOT EXISTS (SELECT D.DormName FROM Dorm D
                           EXCEPT
                          (SELECT LRD.DormName FROM LowResDorm LRD)
                           UNION
                          SELECT HRD.DormName FROM HighResDorm HRD)
                       ))); /* OR PERHAPS */

CREATE ASSERTION CompleteCoverOverLowAndHighRes (
      CHECK (NOT EXISTS
                 (SELECT D.DormName FROM Dorm D
                  WHERE D.DormName
                          NOT IN (SELECT LRD.DormName FROM LowResDorm LRD)
                                   UNION
                                 SELECT HRD.DormName FROM HighResDorm HRD)
                       ))); /* OR PERHAPS */

CREATE ASSERTION CompleteCoverOverLowAndHighRes (
      CHECK (NOT EXISTS
                 (SELECT D.DormName FROM Dorm D
                  WHERE D.DormName
                          NOT IN (SELECT LRD.DormName FROM LowResDorm LRD)
                      AND D.DormName
                          NOT IN  (SELECT HRD.DormName FROM HighResDorm HRD)
                       )));
```

3 pts for any of these; others may be possible

/* Write a general assertion that ensures that there is no Dorm in LowResDorm that is also in HighResDorm, and vice versa.*/

3 pts for any of these; others may be possible

```
CREATE ASSERTION NoOverlapBetweenHighAndLowRes (
CHECK (NOT EXISTS (SELECT * FROM LowResDorm L, HighResDorm H
                    WHERE L.DormName=H.DormName)
); /* OR */
CREATE ASSERTION NoOverlapBetweenHighAndLowRes
CHECK (NOT EXISTS (SELECT DormName FROM LowResDorm
                   INTERSECT
                   SELECT DormName FROM HighResDorm)
);
```

/* Write a general assertion that ensures that each Floor of a high res dorm is associated with at least one high res electricity sensor */

3 pts for any of these; others may be possible

```
CREATE ASSERTION FloorParticipatesHRElecSensor (
CHECK  (NOT EXISTS
           (SELECT *
            FROM Floor F
            WHERE (F.DormName, F.FloorNum)
                   NOT IN (SELECT HRES.DormName, HRES.FloorNum
                           FROM HRElecSensor HRES)))
);

CREATE ASSERTION FloorParticipatesHRElecSensor
CHECK (NOT EXISTS
            (SELECT F.DormName, F.FloorNum FROM Floor F
               EXCEPT
             SELECT HRES.DormName, HRES.FloorNum FROM HRElecSensor HRES)
);

CREATE ASSERTION FloorParticipatesHRElecSensor ( /* A complicated version of first
solution */
CHECK  (NOT EXISTS
            (SELECT * FROM Floor F
             WHERE F.DormName NOT IN (SELECT HRES.DormName
                                      FROM HRElecSensor HRES
                                      WHERE HRES.FloorNum = F.FloorNum)
                 OR  F.FloorNum NOT IN (SELECT HRES.FloorNum
                                      FROM HRElecSensor HRES
                                      WHERE HRES.DormName = F.DormName)
);
```

/* Ensure that each Floor **of a high res dorm** is associated with at least one electricity sensor */

Why don't these work?

```
CREATE ASSERTION FloorParticipatesHRElecSensor (
CHECK  (EXISTS (SELECT F.DormName, F.FloorNum
                        FROM HRElecSensor HRES, Floor F
                        WHERE HRES.DormName = F.DormName AND
                                   HRES.FloorNum = F.FloorNum)
);
```

```
CREATE ASSERTION FloorParticipatesHRElecSensor (
CHECK  (EXISTS
            (SELECT *
             FROM Floor F
             WHERE (F.DormName, F.FloorNum)
                       IN (SELECT HRES.DormName, HRES.FloorNum
                              FROM HRElecSensor HRES)))
```

/* Write a trigger in SQLite that mimics the DELETE CASCADE action of a Foreign Key constraint in HRElecSensor that references Floor. That is, when a DELETE is made to Floor, all HRElecSensors associated with that floor are deleted */

```
CREATE TRIGGER DeleteHRElecSensorsWhenFloorDeleted
AFTER DELETE ON Floor
FOR EACH ROW    /* "FOR EACH ROW" optional */
BEGIN
DELETE FROM HRElecSensors
        WHERE FloorNum = OLD.FloorNum AND DormName = OLD.DormName;
END;
```

3 pts for this; forgive minor syntactic errors (this time) like parens, missing ';', etc

/* Write a trigger in SQLite that implements part of the constraint that each Floor participate in HRElecSensor. In particular, when the only representative of a floor is deleted from HRElecSensors, then that floor is also deleted from Floor */

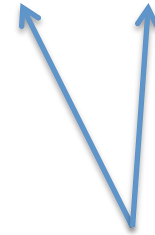
```
CREATE TRIGGER DeleteFloorWhenOnlyHRElecSensorDeleted
AFTER DELETE ON HRElecSensor
FOR EACH ROW    /* "FOR EACH ROW" optional */
WHEN NOT EXISTS (SELECT * FROM HRElecSensor
                 WHERE FloorNum = OLD.FloorNum AND DormName = OLD.DormName)
BEGIN
DELETE FROM Floor
        WHERE FloorNum = OLD.FloorNum AND DormName = OLD.DormName;
END;
```

3 pts for this; forgive minor syntactic errors (this time) like parens, missing ';', etc

The last representative was just deleted

/* Extra Credit: Write a trigger in SQLite that implements part of the constraint that each Floor participate in HRElecSensor. In particular, when a floor is inserted into Floor, an initial entry into HRElecSensors for that inserted floor */

```
CREATE TRIGGER InsertHRElecSensorWhenFloorInserted
AFTER INSERT INTO Floor
FOR EACH ROW    /* "FOR EACH ROW" optional */
WHEN NOT EXISTS (SELECT * FROM HRElecSensor          /*  WHEN clause optional */
                 WHERE FloorNum = NEW.FloorNum AND
                       DormName = NEW.DormName)
BEGIN
INSERT INTO HRElecSensor VALUES (NEW.DormName, NEW.FloorNum, NULL, NULL);
END;
```



2 pts

HRElecSensorID and HRElecSensorOnlineDate are allowed to be NULL in HRElecSensor (look at the table declaration). If they had been declared as NOT NULL, then we could not put NULL here, though we could use a special default (dummy, initial) value.

If your answer made reference to an autoincrement Variable for HRElecSensorID and/or a reference to a CurrentDate function for HRElecSensorOnlineDate Then you should receive credit.

Reflect on these questions. You do NOT need to answer and submit them in writing, but they will be topics of discussion.

a) If you try to delete a row in Observer that has one or more 'associated' entries in DormWebPage, what will happen?

b) If you try to delete a Dorm, for which no one has ever looked at (Observed) a DormWebPage for it, what will happen?

c) If you try to delete a Dorm, for which there have been one or more recorded views of DormWebPages for it, what will happen?

d) How many electricity sensors are associated with a low res dorm (so far as the DB encodes)? And vice versa?

e) How many electricity sensors are associated with a high res dorm (so far as the DB encodes)?

f) Could the current database design (tables and assertions) support the situation of a low res dorm becoming a high res dorm WITHOUT losing past data from its low res days? If so, explain, and if not, explain what changes to the DB design you might make to support this (high likelihood eventuality) PRIOR to DB implementation.

g) How could the DB design be changed to support records of room and plug level measurements of electricity (and perhaps faucet level water readings)?

Think about these queries for exam

Write SQL queries specified below in a manner that is consistent with the table definitions.

a) Write a SQL query that lists all of the individual water readings recorded for each floor of McGill, a high res dorm, on Jan 28, 2012 (HRWReadingDate = '2012-01-28'). Each row of the output should list dorm name (all McGill), floor number, the date (yes, the dates in each row should be the same), the time of reading, and the reading value.

b) Write a SQL query that lists all of the individual water readings recorded for each floor of each high res dorm on Jan 28, 2012 (HRWReadingDate = '2012-01-28'). Each row of the output should list dorm name, floor number, the date (yes, the dates in each row should be the same), the time of reading, and the reading value.

c) Much like (b), but rather than listing all the readings for a given day, list the AVERAGE water reading values for each day, together with the date, the dorm name, and the floor number.

d) Just like (c), but list only those daily averages that are computed over more than 5 values.

e) Write an SQL query that lists, for each dorm (low or high res), the AVERAGE ambient (outside) temperature and the AVERAGE electricity readings on each day in which the MINIMUM ambient temperature exceeds 70 degrees. Thus, the output should list dorm name, average electrical reading (listed as AveElec), and average ambient temperature (as AveTemp), and the reading date.

f) Write an SQL query that lists the average electrical readings for each floor that were viewed by an observer with 'FloorWebPage.ObserverNetworkID = X', for each day in which the MAX ambient temperature was greater than 80 degrees (AmbientTemp > 80). Include 'FloorWebPage.ObserverNetworkID = X' as is. Note that 'X' is a variable; when an unbound variable appears in a query, it is a parameter that is requested at interpretation time). Each row of the output should list network id (which should be the same across all rows), the AVERAGE ambient temperature for the day (as AveTemp), the dorm name, floor, sensor id, electrical reading date, and average value (as AveElec).