# INFO330 Final Exam – Winter 2024

## Instructions

This exam is take-home. It will be released on Tuesday, March 12 at 8AM and must be turned in through Canvas by **Tuesday, March 12 at 6PM PT**. **No exceptions will be made for late exams.** If you experience any technical difficulties uploading through Canvas by the deadline, you should email your completed exam to your instructors prior to the deadline to avoid penalty. Please plan accordingly.

This exam is designed to take 2 hours to complete. In that time, please answer each question below to the best of your ability. You may use the classroom server to test and refine your queries, though please work in your own databases. There are 131 points available on this exam, but it will be scored out of a total of 125 points (in other words, 6 points of extra credit are available here in this exam in addition to the 25 points of extra credit available from the 4 extra credit assignments in Canvas).

Please print your name, sign, and date the following statement before uploading your complete exam:

*I attest that I did not communicate with anyone other than the instructors about the questions on this final exam, and that all work is my own.*

Name: *John Harrison*____________

Signature: *John Harrison*____________

Date: *March 12, 2024*____________

All answers will be reflected in runnable plaintext in attached .sql file.

## PART 1: SQL (50 pts total)

Please answer each of the following questions by writing a SQL query unless otherwise specified. Each query should run on a table with a matching schema; in other words, syntax matters!

Here is the premise for the problems in this part:

King County Metro is deploying a new system to manage the public transit system, specifically buses and bus routes. The system has *drivers* that drive *buses* on certain *routes* with *stations* that the route might stop at. Each station has an attribute covered that describes whether it is covered from the weather (i.e., rain). Each route consists of a series of stations, in order.

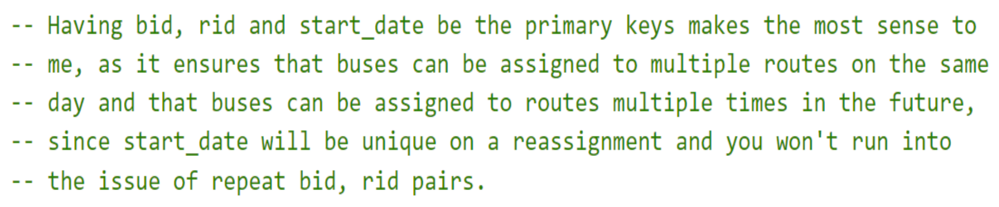
Here's a proposed schema:

```
Bus(bid, electrified, year)
Route(rid, name, type, electrified, description)
Station(sid, name, addr, lat, long, covered)
Stops(rid, sid, stop_order)
BusAssignment(bid, rid, start_date, end_date)
```

- In the *Bus* table, each bus is uniquely identified by a surrogate bus ID (bid). It includes attributes such as whether the bus is electrified, a boolean value indicating if it runs on electricity, and the year of manufacture. Both ‘electrified’ and ‘year’ always have values.
- In *Route*, each bus route is uniquely identified by a surrogate route ID (rid). The attribute ‘name’ is a string name for the route; ‘type’ is a string describing the type of route and can only take on the values “local”, “express”, “commuter”, or “night”; ‘electrified’ is a boolean value indicating whether the route is served by electrified buses; and ‘description’ is a free text description about the route, containing information such as its destination or major landmarks it passes through. The attributes ‘name’, ‘type’, and ‘electrified’ are available for all routes, but ‘description’ may not be.

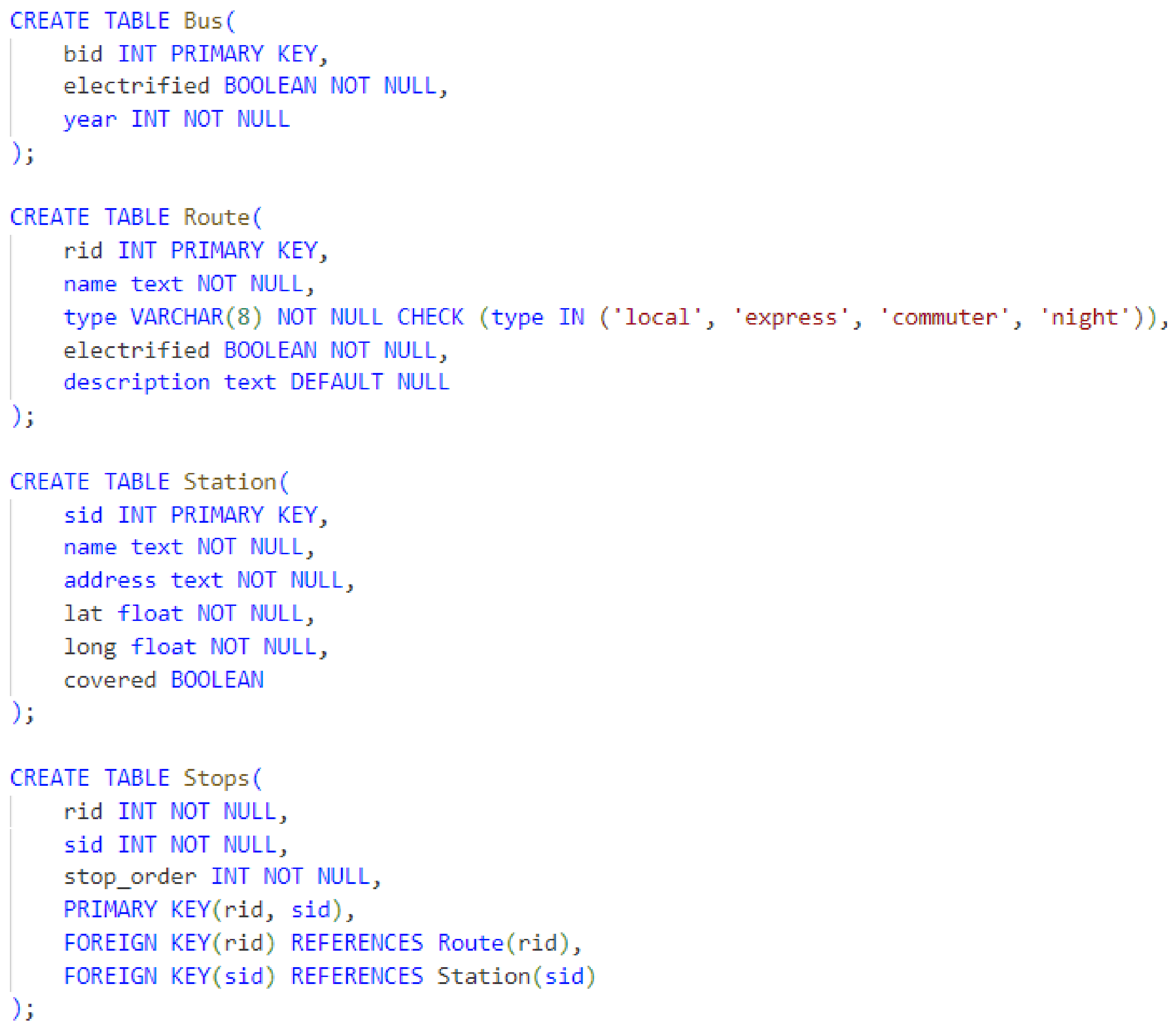
- Each station in the *Station* table is uniquely identified by a surrogate station ID (sid) and is associated with a station ‘name’ as a descriptive text string, and ‘address’, which stores its street address. Latitude (lat) and longitude (long) are decimal numbers that provide the geographic coordinates of the station. A boolean attribute called ‘covered’ indicates whether the station provides shelter from weather conditions. The attributes ‘name’, ‘address’, ‘lat’, and ‘long’ are always specified.
- The *Stops* table defines the sequence of stops along each bus route. It records the relationship between routes (identified by rid) and stations (identified by sid) and specifies the order in which buses stop at each station. The ‘stop_order’ attribute is an integer specifying which stop along the route a station is (e.g., order 1 means this is the first stop on the route, order 10 means it is the 10th stop) and must be specified.
- The *BusAssignment* table records the assignment of buses to specific routes within defined time periods. Buses (identified by bid) are assigned to operate on routes (identified by rid) during a specified timeframe. The attributes ‘bid’, ‘rid’, and ‘start_date’ are always specified. If the ‘end_date’ is NULL, this means the bus is currently in service on that route.
- ‘bid’, ‘rid’, and ‘sid’ are integers

1. (5 pts) I have not specified a primary key for the BusAssignment table. What attribute or set of attributes makes sense as primary key and why?

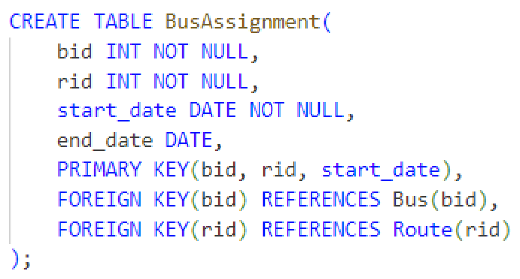


```
-- Having bid, rid and start_date be the primary keys makes the most sense to
-- me, as it ensures that buses can be assigned to multiple routes on the same
-- day and that buses can be assigned to routes multiple times in the future,
-- since start_date will be unique on a reassignment and you won't run into
-- the issue of repeat bid, rid pairs.
```

2. (10 pts) Write the sequence of SQL statements necessary to create the tables above. Include primary key, key, foreign key, NULL constraints, and/or CHECK constraints if they are needed. Please make sure to implement the primary key you've specified in your answer to question 1.

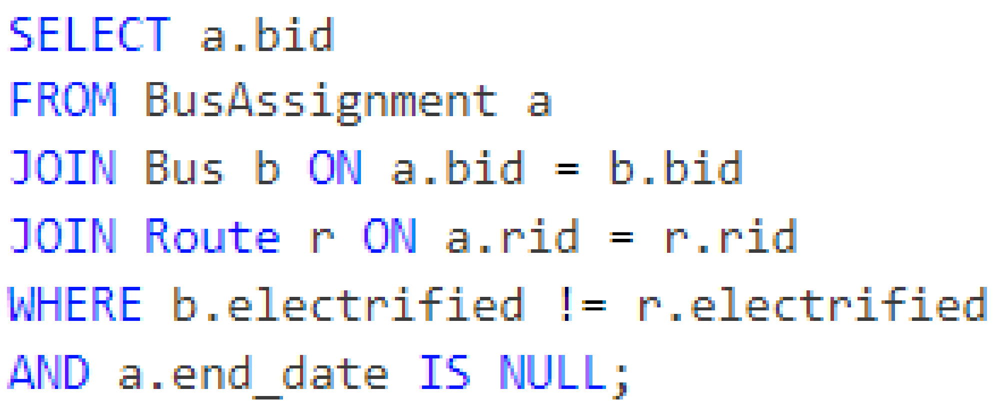


```sql
CREATE TABLE Bus(
    bid INT PRIMARY KEY,
    electrified BOOLEAN NOT NULL,
    year INT NOT NULL
);

CREATE TABLE Route(
    rid INT PRIMARY KEY,
    name text NOT NULL,
    type VARCHAR(8) NOT NULL CHECK (type IN ('local', 'express', 'commuter', 'night')),
    electrified BOOLEAN NOT NULL,
    description text DEFAULT NULL
);

CREATE TABLE Station(
    sid INT PRIMARY KEY,
    name text NOT NULL,
    address text NOT NULL,
    lat float NOT NULL,
    long float NOT NULL,
    covered BOOLEAN
);

CREATE TABLE Stops(
    rid INT NOT NULL,
    sid INT NOT NULL,
    stop_order INT NOT NULL,
    PRIMARY KEY(rid, sid),
    FOREIGN KEY(rid) REFERENCES Route(rid),
    FOREIGN KEY(sid) REFERENCES Station(sid)
);
```



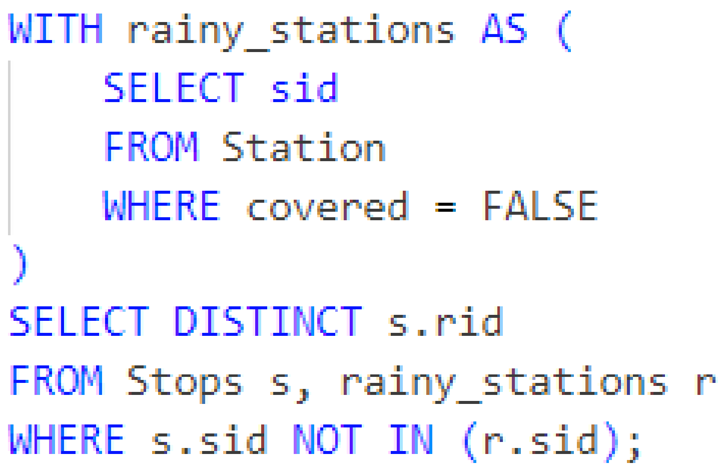
```sql
CREATE TABLE BusAssignment(
    bid INT NOT NULL,
    rid INT NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE,
    PRIMARY KEY(bid, rid, start_date),
    FOREIGN KEY(bid) REFERENCES Bus(bid),
    FOREIGN KEY(rid) REFERENCES Route(rid)
);
```

3. (5 pts) A route and bus are *incompatible* if they disagree on electrified status (e.g., an electrified bus is servicing a non-electrified route or a non-electrified bus is servicing an electrified route. Write a query to return the bus ids (bid) currently in service that are running on incompatible routes.



```
SELECT a.bid
FROM BusAssignment a
JOIN Bus b ON a.bid = b.bid
JOIN Route r ON a.rid = r.rid
WHERE b.electrified != r.electrified
AND a.end_date IS NULL;
```

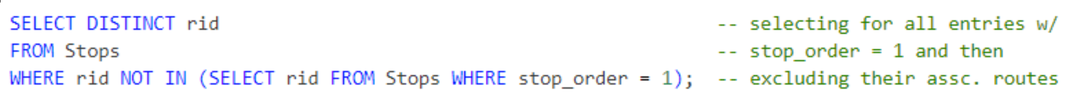
4. (10 pts) A route is *weatherproof* if *all* of the stations on its route are covered. Write a query that returns a list of route ids (rid) that are weatherproof. Your query should return the unique route ids associated with only stations that are covered.



```
WITH rainy_stations AS (
    SELECT sid
    FROM Station
    WHERE covered = FALSE
)
SELECT DISTINCT s.rid
FROM Stops s, rainy_stations r
WHERE s.sid NOT IN (r.sid);
```

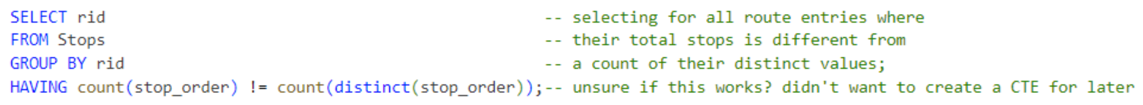
5. A proper route with N stations should have exactly N entries in the Stops table and no order value is duplicated. An *inconsistent* route is one where (i) the route doesn't start at order=1, (ii) the same order number is duplicated, or (iii) it skips an order number.

   (a) (5 pts) Write a query to return the unique route ids (rid) for all inconsistent routes that violate (i), where the route doesn't start with a station at order=1.



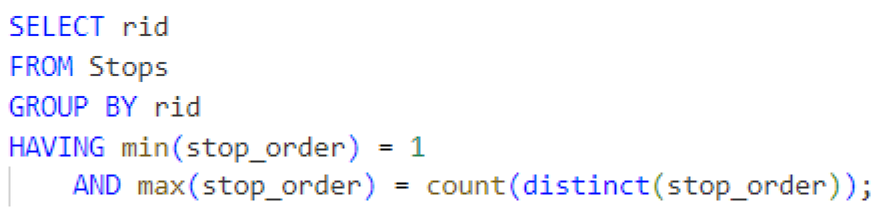
```
SELECT DISTINCT rid                                            -- selecting for all entries w/
FROM Stops                                                     -- stop_order = 1 and then
WHERE rid NOT IN (SELECT rid FROM Stops WHERE stop_order = 1); -- excluding their assc. routes
```

   (b) (5 pts) Write a query to return the unique route ids (rids) for all inconsistent routes that violate (ii), where the same order number is duplicated. You could think of this as checking whether the number of order numbers is the same as the unique number of order numbers associated with each route.



```
SELECT rid                                              -- selecting for all route entries where
FROM Stops                                              -- their total stops is different from
GROUP BY rid                                            -- a count of their distinct values;
HAVING count(stop_order) != count(distinct(stop_order));-- unsure if this works? didn't want to create a CTE for later
```

   (c) (5 pts) Write a query to return the unique route ids (rid) for all inconsistent routes that violate (iii), where the route skips an order number. You could think of this as checking whether the smallest order number associated with the route is 1 and the largest order number associated with the route is the same as the number of unique order numbers on the route.



```
SELECT rid
FROM Stops
GROUP BY rid
HAVING min(stop_order) = 1
    AND max(stop_order) = count(distinct(stop_order));
```

(d) (5 pts) Write a query that combines results from your queries from (a), (b), and (c) to return all unique route ids (rids) that are inconsistent for any of the three reasons, along with three additional boolean columns called 'incorrect_start', 'duplicate_order', and 'skips_number' that correspond to whether each of (a), (b), and/or (c) are violated. Include your queries from (a), (b), and (c) as CTEs, followed by the main query to construct the output table. Note: some routes may be inconsistent in multiple ways.

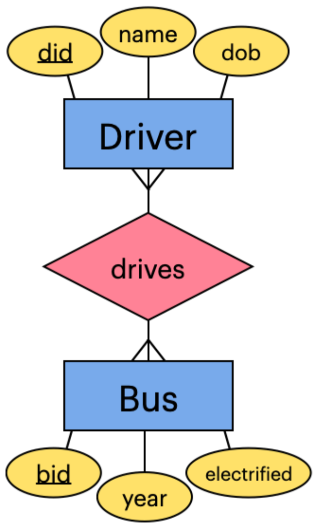
Here are some example rows that could appear in the resulting table:

| rid | incorrect_start | duplicate_order | skips_number |
|---|---|---|---|
| 2 | FALSE | TRUE | TRUE |
| 19 | TRUE | FALSE | |

```
WITH incor_start AS (
    SELECT DISTINCT rid
    FROM Stops
    WHERE rid NOT IN (SELECT rid FROM Stops WHERE stop_order = 1)
),
dup_order AS (
    SELECT rid
    FROM Stops
    GROUP BY rid
    HAVING count(stop_order) != count(distinct(stop_order))
),
skip_num AS (
    SELECT rid
    FROM Stops
    GROUP BY rid
    HAVING min(stop_order) = 1
        AND max(stop_order) = count(distinct(stop_order))
)
SELECT s.rid,
        (IF s.rid IN i.rid THEN TRUE ELSE FALSE) as incorrect_start,
        (IF s.rid IN d.rid THEN TRUE ELSE FALSE) as duplicate_order,
        (IF s.rid IN n.rid THEN TRUE ELSE FALSE) as skips_number
FROM Stops s, incor_start i, dup_order d, skip_num n
GROUP BY s.rid;
-- i don't know how to do this! so i'll try my best and hopefully get partial credit.
```

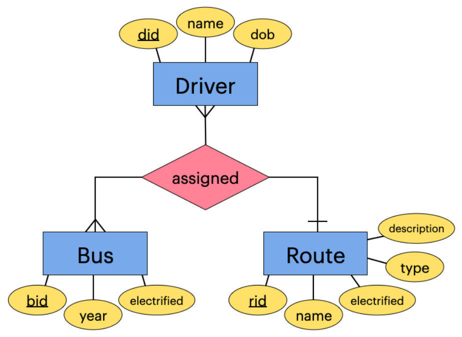
## PART 2: ERDs and Relational Schema (41 pts total)

6. (8 pts) King County Metro has proposed some changes to the schema, and provided the following ER diagram of a new relationship. `Driver` is a new entity set representing employees who drive buses, and is connected to the `Bus` entity set by the `drives` relationship. <u>did</u> is an integer primary key, name is the driver's name represented as a string, and dob is their birthdate (none of these can be NULL). Write additional CREATE TABLE statements to create new table(s) based on the content in this ER diagram. You can choose to modify the CREATE TABLE statement for the existing `Bus` table if you believe it needs to be modified; otherwise, it will be assumed to be unchanged from what you provided in problem 2. The tables Route, Station, Stops, and BusAssignment can be assumed unchanged from what you provided in problem 2.

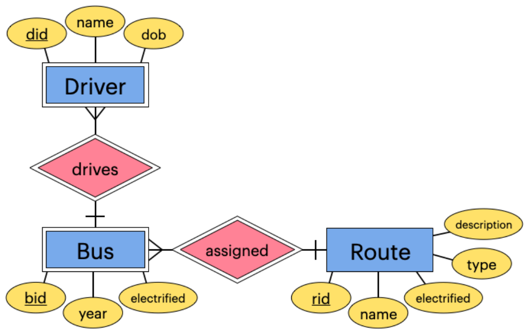


```
CREATE TABLE Driver(
    did INT PRIMARY KEY,
    name TEXT NOT NULL,
    dob DATE NOT NULL
);

CREATE TABLE Drives(
    did INT NOT NULL,
    bid INT NOT NULL,
    PRIMARY KEY(did, bid),
    FOREIGN KEY(did) REFERENCES Driver(did),
    FOREIGN KEY(bid) REFERENCES Bus(bid)
);
```

7. Two ERD snippets are shown below.



(i)



(ii)

(a) (8 pts) Describe what (i) and (ii) each mean and how the two diagrams differ, both in turns of what is represented in the ERD and in plain language (e.g., what are the implications of the cardinalities?). Focus on articulating the differences.

```
-- (i) is a many-to-many-to-one relationship with no weak entity sets or
-- supporting relationships. In this case, each driver, bus and route are
-- uniquely identifiable, independent of their relationship with each other.
-- Many drivers can be assigned to many buses, and many buses can be assigned
-- to many drivers, but each bus/driver pair can only be assigned to one route,
-- since both of them pass through the assigned relation.

-- (ii) is saying that many drivers can drive one bus, and many buses can be
-- assigned to one route. The outline around driver and bus indicates they're
-- weak entity sets, which means that they're uniquely identified by their
-- supporting relationship. In this case, multiple drivers can have the same
-- name and multiple buses from the same year can be electrified, but the
-- pairing of did and bid serves as a unique identifier. Conversely, a bus'
-- assignment to route also serves as a unique identifier based on its
-- supporting relationship, where many buses can be assigned to one route.
```

(b) (5 pts) Which one do you think is a better reflection of the real world scenario and why?

```
-- I think that (ii) is a more accurate reflection of the real-world scenario,
-- as it feels more flexible and applicable to on-the-job situations. With the
-- separation of driver from route, it doesn't necessarily mean each driver is
-- limited to one route/bus pair at a time; drivers can drive multiple buses,
-- where each of those go on multiple routes, but drivers themselves are never
-- stuck on one route. In real world cases, I imagine drivers would likely
-- stay consistent to their routes, and that buses would stay consistent on
-- the lines they run, but (ii) redefines that relationship and makes it so
-- both can have more flexibility.
```

8. (20 pts) Imagine you are given data in a spreadsheet called `equipment_reservation` with the following columns (there is no actual spreadsheet; you don't need it to complete this problem):

   `(name, status, gym, address, equipment, type, trainer, date)`

   You are also given the following functional dependencies for these attributes:

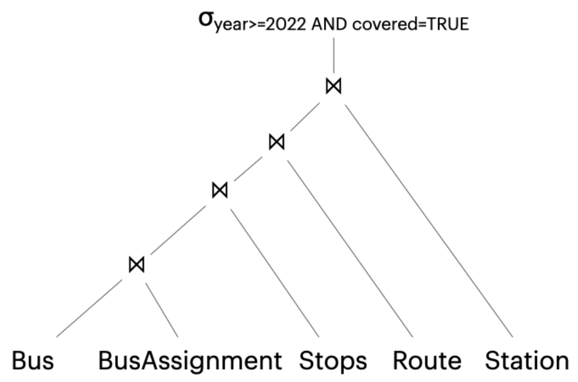
   **equipment→type** (each piece of equipment has a specific type)
   **gym, type→trainer** (each gym has one specialist trainer for each type of equipment)
   **name→status** (each member has a specific status)
   **gym→address** (each gym has a specific address)

   Decompose `equipment_reservation` into BCNF. In your final answer, make sure to indicate the key(s) of each relation. If you show your work, you are more likely to earn partial credit in case of mistakes!

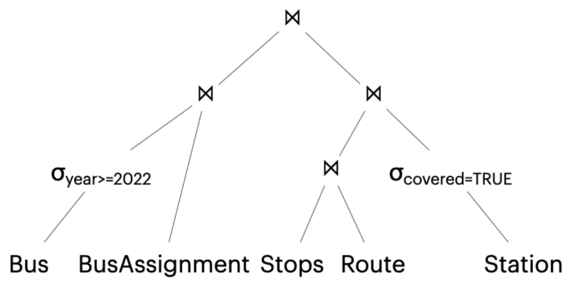
```
-- A0: (name, status, gym, address, equipment, type, trainer, date)
-- only key is {name, gym, type, equipment}, so
-- A0: (*name*, status, *gym*, address, *equipment*, *type*, trainer, date)

-- equipment→type (each piece of equipment has a specific type)
-- A1: (*name*, status, *gym*, address, *equipment*, trainer, date)
-- A2: (*equipment*, type)

-- name→status (each member has a specific status)
-- A1: (*name*, *gym*, address, *equipment*, trainer, date)
-- A2: (*equipment*, type),
-- A3: (*name*, status)

-- gym→address (each gym has a specific address)
-- A1: (*name*, *gym*, *equipment*, trainer, date)
-- A2: (*equipment*, type)
-- A3: (*name*, status)
-- A4: (*gym*, address)

-- gym, type→trainer (each gym has one specialist trainer for each type of equipment)
-- A1: (*name*, *gym*, *equipment*, date)
-- A2: (*equipment*, type)
-- A3: (*name*, status)
-- A4: (*gym*, address)
-- A5: (*gym*, *type*, trainer)

-- further decomposition (i think?)
-- A1: (*name*, *gym*, *equipment*, date)
-- A6: (*gym*, *equipment*, trainer)
-- A2: (*equipment*, type)
-- A3: (*name*, status)
-- A4: (*gym*, address)
```

## PART 3: Query Evaluation & Optimization (28 pts total)

9. You are given the following RA query plan (P0) for the Metro scenario:

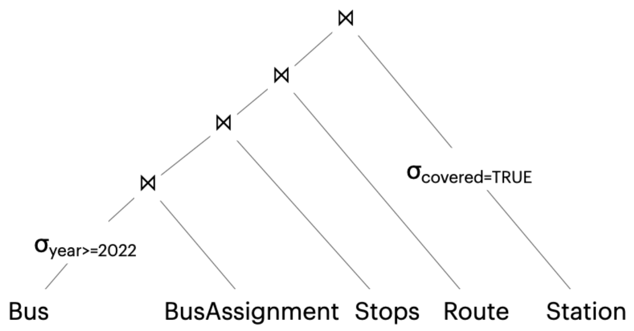


(a) (2pt) Does the following plan produce the same result as P0? If not, why not?



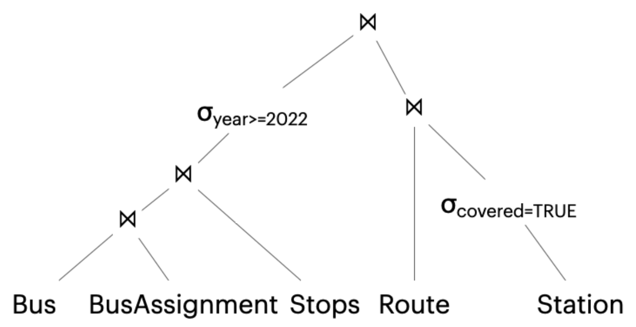
```
-- Yes, it does. It selects for buses from 2022, finds their assignments, and
-- joins them with only the routes with covered stations.
```

(b) (2pt) Does the following plan produce the same result as P0? If not, why not?



```
-- No, it doesn't. It's selecting for buses in 2022, but then joining on their
-- assingments, joining on stops, and joining on routes so you have all routes
-- being selected. Then it's pulling all covered stops from those routes, rather
-- than just pulling the covered routes with only buses from 2022.
```

(c) (2pt) Does the following plan produce the same result as P0? If not, why not?



```
-- Yes, it does. It's selecting for only routes with stops with covered bus
-- stations, then joining that on the routes from bus assignments only with
-- buses from the year of 2022.
```

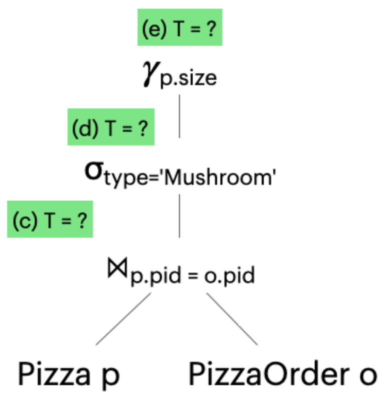
(d) (2 pt) Of the plans that produce the same results, which one or ones are most efficient and why?

```
-- A will be the most efficient, as it pushes its select statements furthest
-- down and therefore each join has less data to work with than C or the
-- original query.
```

10. Cardinality estimation: consider the following schema:

```
CREATE TABLE Pizza (
  pid int PRIMARY KEY,
  type VARCHAR(50),
  size int,
  price MONEY
);

CREATE TABLE PizzaOrder (
  order_id int,
  pid int REFERENCES Pizza(pid),
  quantity int,
  PRIMARY KEY (order_id, pid)
);
```

You are given the following RA diagram and statistics about the data:



```
T(Pizza) = 100
T(PizzaOrder) = 10000
V(Pizza, type) = 25
V(Pizza, size) = 4
V(PizzaOrder, order_id) = 8000
V(PizzaOrder, pid) = 100
```

Please answer the following (show your work to get partial credit):

(a) (4 pt) Describe in plain language what the RA query does:

```
-- The query joins all orders based on their given pid, so that each order
-- has an assigned Pizza. It then selects for all pizza orders with the
-- type Mushroom, and orders them based on their size. Basically, it just
-- groups all the mushroom pizzas together and sorts them by size.
```

(b) (4 pt) What is `V(Pizza, pid)`?

```
-- Since V(PizzaOrder, pid) = 100, V(Pizza, pid) = 100
```

(c) (4 pt) What is the expected number of tuples at (c) in the RA query?

```
-- Since you're joining each order to a specific pizza, and there are
-- 10,000 orders, T at (c) is 10,000.
```

(d) (4 pt) What is the expected number of tuples at (d) in the RA query?

```
-- Since we don't know the distribution of pizza types in these orders,
-- we can assume they're normally distributed, meaning each pizza type has
-- a roughly even number of orders. Since Mushroom is 1 of 25 types, T = 400.
```

(e) (4 pt) What is the expected number of tuples at (e) in the RA query?

```
-- (e) is a grouping/aggregation query at the very top based off of size.
-- Since we don't know how sizes of mushroom pizzas are distributed, we again
-- can assume they're normally distributed, meaning each size is represented.
-- Since V(Pizza, size) = 4, we can asssume T to be 4 as well.
```

## PART 4: Concepts (12 pts total)

For each statement below, indicate whether it is true or false (2 pt each)

11. (T/F) If R(A, B, C, D) satisfies the functional dependencies AB $\rightarrow$ C and C $\rightarrow$ A then C is a key. **True.**

12. (T/F) If R(A, B, C, D) satisfies the functional dependencies A $\rightarrow$ B and B $\rightarrow$ C and C $\rightarrow$ D, then D is a key. **False.**

13. (T/F) If R(A, B, C, D) satisfies the functional dependencies A $\rightarrow$ B and B $\rightarrow$ C and C $\rightarrow$ D and D $\rightarrow$ A, then D is a key. **True.**

14. (T/F) The query optimizer may choose a different algorithm for a join depending on statistics about the data, such as number of tuples in a table or number of unique values in an attribute. **True**.

15. (T/F) A table can have more than one clustered index, but only one unclustered index. **False**.

16. (T/F) An index is clustered when the physical table is sorted by the same attributes as are included in the index. **True**