Exam 2 - Requires Respondus LockDown Browser - R...

Attempt 1 of 1

Written Mar 26, 2025 3:01 PM - Mar 26, 2025 4:49 PM

Attempt Score **85 / 100 - 85 %**

Overall Grade (Highest Attempt) **85 / 100 - 85 %**

[Data modeling, 25 points]

Consider the following tables about cars sold at dealerships.

Assume that:

- Every salesperson works for **exactly one** dealership.
- If a car is sold, there will be **only one** salesperson involved in the sale.

Note that:

- Key attributes in the tables are bolded and underlined. vin is a vehicle identification number.
- We assume that the names of both dealerships and salespeople are unique.
- Car(**vin**, make, model, year) // e.g., (100, “Toyota”, “Camry”, 2025)
- Dealership(**name**, address) // e.g., (“Downtown LA”, “100 Vermont Ave”)
- Salesperson(**name**, phone, dealership) // e.g., (“Michael”, “321-123-4567”, “Downtown LA”)
- SoldBy(**vin**, salesperson, price) // e.g., (100, “Michael”, 10000)

Question 1 21 / 25 points

Reverse engineer the above tables into the corresponding ER model. State the entity sets, relationships, and attributes in the model. Be sure to indicate keys for entity sets, and the multiplicity of relationships. For the one-side of multiplicity, indicate if it is exactly one or at most one.

entity sets:
Car(vin PRIMARY KEY, make, model, year)

Dealership(name PRIMARY KEY, address)
Salesperson(name PRIMARY KEY, phone, dealership)

relationships:
SoldBy(vin PRIMARY KEY, salesperson, price)

SoldBy has many-to-one relationship with Salesperson //at most one
Salesperson has many-to-one relationship with Dealership //at most one

▼ Hide question 1 feedback

Feedback

[-2] Missing or incorrect key for an entity
[-2] Missing a relationship or describing it incorrectly

[SQL-1, 15 points]

Consider the same tables as in Question 1, reproduced below.

- Car(**vin**, make, model, year) // e.g., (100, "Toyota", "Camry", 2025)
- Dealership(**name**, address) // e.g., ("Downtown LA", "100 Vermont Av")
- Salesperson(**name**, phone, dealership) // e.g., ("Michael", "321-123-4567", "Downtown LA")
- SoldBy(**vin**, salesperson, price) // e.g., (100, "Michael", 10000)

Question 2 8 / 8 points

Write "create table" command to create the Salesperson and Dealership tables in MySQL database. The definition of the tables should capture the data constraints stated in Question 1. Be sure to properly define the key and foreign key. Use the default "reject" action for violations of foreign key constraints.

```
CREATE TABLE Salesperson(
name VARCHAR(255) PRIMARY KEY,
phone VARCHAR(255),
dealership VARCHAR(255),
FOREIGN KEY (`dealership`) REFERENCES `Dealership `(`name`)
);

CREATE TABLE Dealership(
name VARCHAR(255) PRIMARY KEY,
address VARCHAR(255)
);
```

Question 3 7 / 7 points

Explain when (e.g., insert/delete/update) the database server should check possible violations of the foreign key constraint(s) defined in the Salesperson table. For each case, what other actions than reject (e.g., set null or cascade) are also possible when violations occur. Explain your answer.

insert: when inserting data into Salesperson table, database will check if the value of dealership exists in the name column of the Dealership table. if the value of dealership doesn't exist in the name column of the Dealership table, violations will occur.

update: when updating data into Salesperson table, database will check if the value of dealership exists in the name column of the Dealership table. if the value of dealership doesn't exist in the name column of the Dealership table, violations will occur.
when set null is applied in update, if the value of name in Dealership table is updated, the corresponding value in Salesperson table will be set to NULL.
when cascade is applied, if the value of name in Dealership table is updated, the corresponding data in Salesperson table will be deleted.

delete: when deleting data from Salesperson table, database should check if there is any reference with the Dealership table.
when set null is applied, if the value of name in Dealership table is deleted, the corresponding value in Salesperson table will be set to NULL.
when cascade is applied, if the value of name in Dealership table is deleted, the corresponding data in Salesperson table will be deleted.

```
CREATE TABLE Salesperson(
name VARCHAR(255) PRIMARY KEY,
phone VARCHAR(255),
dealership VARCHAR(255),
FOREIGN KEY (`dealership`) REFERENCES `Dealership `(`name`) ON
UPDATE SET NULL ON DELETE SET NULL
);

CREATE TABLE Salesperson(
name VARCHAR(255) PRIMARY KEY,
phone VARCHAR(255),
dealership VARCHAR(255),
FOREIGN KEY (`dealership`) REFERENCES `Dealership `(`name`) ON
UPDATE CASCADE ON DELETE CASCADE
);
```

[SQL-2, 20 points]

Consider again the same tables as previous questions, duplicated below.

- Car(**vin**, make, model, year) // e.g., (100, “Toyota”, “Camry”, 2025)
- Dealership(**name**, address) // e.g., (“Downtown LA”, “100 Vermont Ave”)
- Salesperson(**name**, phone, dealership) // e.g., (“Michael”, “321-123-4567”, “Downtown LA”)
- SoldBy(**vin**, salesperson, price) // e.g., (100, “Michael”, 10000)

For each of the following questions, write an SQL query to find the answer to the question. 4 points each question.

Question 4 4 / 4 points

Find salesperson who sold a car whose vin is 100 and a car whose vin is 102. Return name of salesperson only.

```
select sp.name
from Salesperson sp
where sp.name in (
select sb. salesperson
from SoldBy sb
where sb.vin = 100 or sb.vin = 102
);
```

Question 5 4 / 4 points

Find names of salespersons who only sold Toyota’s.

```
select sp.name
from Salesperson sp
where sp.name not in(
select sb.salesperson from Car c inner join SoldBy sb on c.vin = sb.vin
where c. make != 'Toyota'
);
```

Question 6 4 / 4 points

Find names of salespersons who made the most revenue (i.e., with the largest total sales amount).

```
select sb1.name
from SoldBy sb1
group by sb1.salesperson
having sum(sb1.price) >= all(select sum(sb.price) from SoldBy sb group by
sb.salesperson );
```

Question 7 4 / 4 points

Find names of salespersons who have sold more than 5 Toyota’s.

```
select A.salesperson
from (
select sb.salesperson from SoldBy sb inner join Car c on sb.vin = c.vin
where c.make = 'Toyota'
) A
group by A.salesperson having count(*) > 5;
```

Question 8 4 / 4 points

Find the top 3 best-selling brands (i.e., makes) of cars, ranked by the number of cars sold.

```
select c.make, count(*) num_of_sold
from SoldBy sb inner join Car c
on sb.vin = c.vin
group by c.make
```

```
order by num_of_sold desc
limit 3
```

[MongoDB, 25 points]

Now consider storing the same tables in MongoDB, one collection per table. Recall the tables are:

- Car(**vin**, make, model, year) // e.g., (100, “Toyota”, “Camry”, 2025)
- Dealership(**name**, address) // e.g., (“Downtown LA”, “100 Vermont Ave”)
- Salesperson(**name**, phone, dealership) // e.g., (“Michael”, “321-123-4567”, “Downtown LA”)
- SoldBy(**vin**, salesperson, price) // e.g., (100, “Michael”, 10000)

Suppose the corresponding collection uses the same key attribute(s) for its primary key (but stores the values in the JSON object format).

For example, here is an example document in the Car collection:
{“_id”: {“vin”: 100}, “make”: “Toyota”, “model”: “Camry”, “year”: 2025}.

Question 9 2 / 3 points

Write MongoDB functions to insert example documents for Dealership, Salesperson, and SoldBy into the database.

```
db.Dealership.insert({
'_id': {'name': 'Downtown LA'},
'address': '100 Vermont Ave'
});

db.Salesperson.insert({
'_id': {'name': 'Michael'},
'phone': '321-123-4567',
'dealership': 'Downtown LA'
});

db.SoldBy.insert({
'_id': {'vin': 100},
'salesperson': 'Michael',
'price': 10000
});
```

▼ Hide question 9 feedback

Feedback

insertOne

Question 10 — 2 / 2 points

Update the price of car 100 (i.e., its vin = 100) in SoldBy to 20000.

```
db.SoldBy.update({'_id.vin': 100}, {$set: {'price': 20000}});
```

Question 11 — 4 / 4 points

Write a MongoDB function to find out how many cars Michael has sold at a price of at least 20000.

```
db.SoldBy.find({
'price': {$gte: 20000},
'salesperson': 'Michael'
}).count()
```

Question 12 — 4 / 4 points

Write a MongoDB function to find salespersons whose phone contains 123 and who work for a dealership whose address contains “Vermont” (case insensitive). Return names of salespersons only.

```
db.Salesperson.find({
'phone': '/123/',
'dealership': '/Vermont/i'
}, {'_id.name': 1, '_id': 0})
```

▼ Hide question 12 feedback

Feedback

Incorrect use of regex — string literals instead of regex objects; also incorrect field for dealership address filtering.

Question 13 2 / 4 points

Write a MongoDB function to find names of top-3 salespersons ranked by the total sales amount generated by the salesperson. Return names only.

```
db.SoldBy.aggregate(
{$group: {'_id': salespersons, 'total_sales_amount': sum(price)}},
{$sort: {'total_sales_amount': -1}},
{$project: {'_id':1}},
{$limit: 3}
);
```

Hide question 13 feedback

Feedback

Incorrect use of variables (salespersons and sum(price) should be "$salesperson" and {$sum: "$price"}); projection should rename _id to name.

Question 14 4 / 4 points

Write a MongoDB function to find names of salespersons who have sold more than 5 Toyota’s.

```
db.SoldBy.aggregate(
{$lookup: {from: 'Car', localField: '_id.vin', foreignField: '_id.vin', as: 'res'}},
{$unwind: '$res'},
{$match: {'res.make':  'Toyota'}},
{$group: {'_id': 'salespersons', 'num_of_cars': {$sum: 1}},
{$match: {'num_of_cars': {$gt: 5}},
{$project: {'salesperson': 1, '_id': 0}}
);
```

Question 15 4 / 4 points

Write a MongoDB function to find the average sales price of cars made in 2025. Return only the average sales price.

```
db.SoldBy.aggregate(
{$lookup: {from: 'Car', localField: '_id.vin', foreignField: '_id.vin', as: 'res'}},
{$unwind: '$res'},
{$group: {'_id': 'res.year', 'avg_price': {$avg: '$price'}},
{$match: {'_id': 2025}},
{$project: {'avg_price': 1, '_id': 0}}
);
```

[Miscellaneous, 15 points]

Question 16 3 / 3 points

Consider a table: person(name varchar(20), age tinyint unsigned);

a. What is the range of values that can be input for the age?
b. How can you modify the table definition so that it only accepts age values between 0 and 150 inclusive.

a. 0 to 9
b. age INT NOT NULL UNSIGNED CHECK(<=150) ;

Hide question 16 feedback

Feedback

Incorrect range and syntax error in CHECK constraint.

Question 17 1 / 3 points

Consider a directed graph described by an edge table:

Create table edge(src int, tgt int);

Note that src and tgt are the id’s of source and target nodes of a directed edge.

For example, the graph may have the following edges:
(1, 2) (2, 3) (2, 4) (3, 4) (4, 5)

Assume that the graph has no cycles.

Using the edge table, write a view called “length2” that lists node x and node y such that there exists a length-2 path from x to y.

```
create view length2 as
select * from edge
where abs(src - tgt) = 2;
```

Hide question 17 feedback

Feedback

Incorrect logic — checks numeric difference instead of joining paths of length 2.

Question 18 1 / 5 points

Using the above view and the edge table, write an SQL query to find nodes x and y such that there exists either a length-3 path or a length-4 path going from x to y. Return the nodes x, y, and the length of paths. Return the same pair of nodes only once. Hint: you can also select a constant value in SQL. (e.g., select src, tgt, 1 as length from edges).

For example, your query may output:

```
src tgt length
1, 4, 3
1, 5, 4
…
```

This means that there is a length-3 path from node 1 to 4, a length-4 path from node 1 to 5, and so on.

```
select distinct l2.src, e.tgt, abs(l2.src - e.tgt) as length
from length2 l2
inner join edge e
on l2.tgt = e.src
where abs(l2.src - e.tgt) = 3 or abs(l2.src - e.tgt) = 4;
```

▼ Hide question 18 feedback

Feedback

Incorrect logic — uses absolute difference instead of proper joins to construct length-3 and length-4 paths.

Question 19 2 / 4 points

Derive the UTF-8 encoding for the character whose code point is U+00a5. Show the encoding in both binary and hexadecimal format.

U+0080 < U+00a5 < U+07FF => 2 byte units, 11 points
0000 0000 1010 0101

binary: 1100 0010 0010 0101
hexadecimal: C2 25

▼ Hide question 19 feedback

Feedback

1100 0010 1010 0101 → C2 A5

Done