**Assignment 1**

*Introduction to Databases*

**Due date:** 04 June 2021

**Course ID:** AMOD- 5450H

**Instructor:** Alaadin Addas

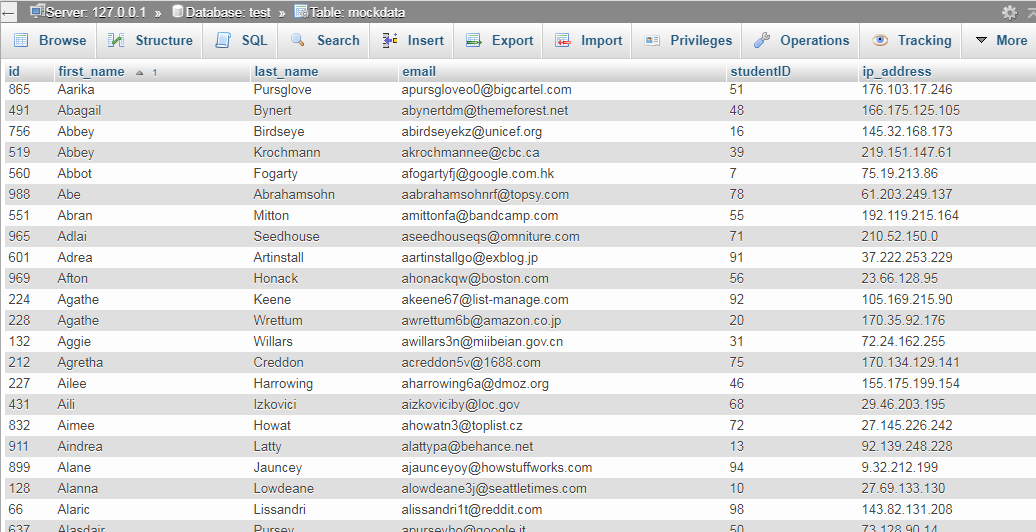
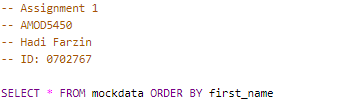
**Student name:** Hadi Farzin

**ID:** 0702767

**Part 1 - SQL**

1. **Query that alphabetizes the first names in ascending order**

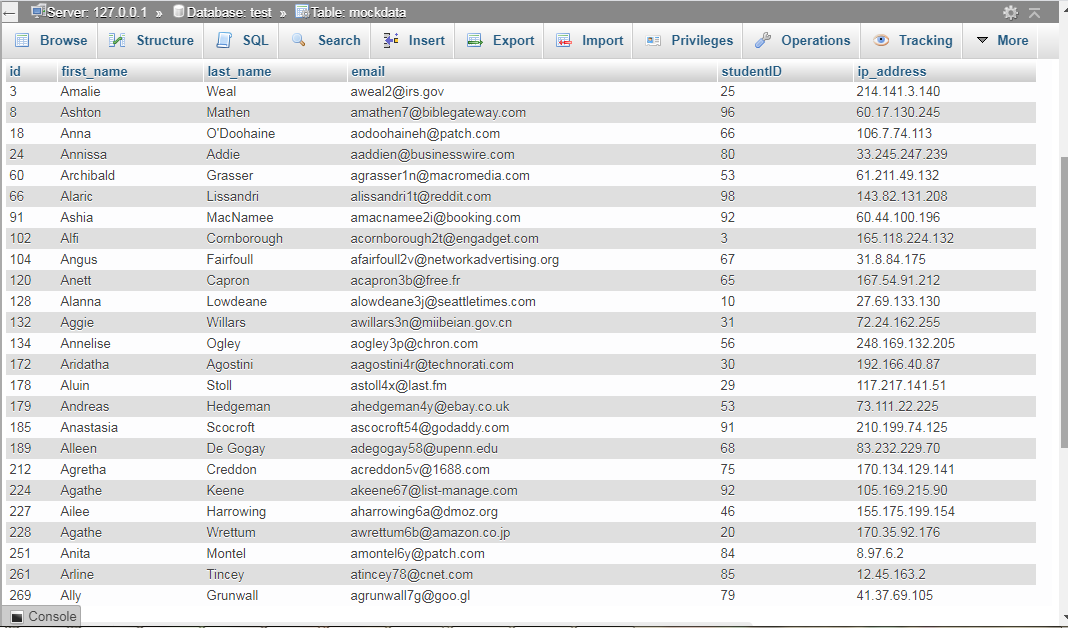
SELECT \* FROM mockdata ORDER BY first\_name



1. **Query that outputs every student whose first name starts with an “A”**

SELECT \* FROM `mockdata` WHERE first\_name LIKE "A%"

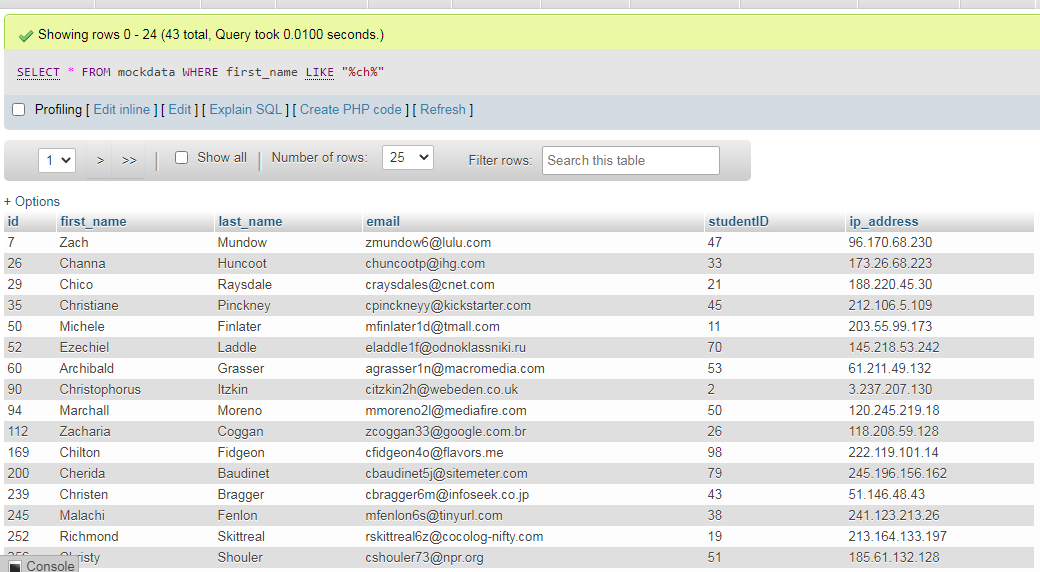




1. **Query that outputs every student who has a ‘ch’ anywhere in their name**

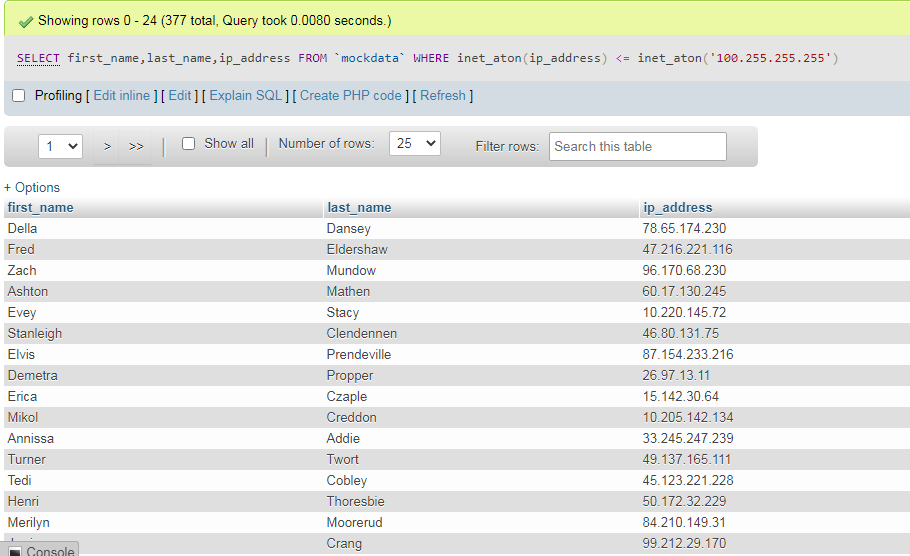
SELECT \* FROM `mockdata` WHERE first\_name LIKE "%ch%"





1. **Selects the first name, last name, and IP address, for every IP address that starts with ‘100’ or less.**

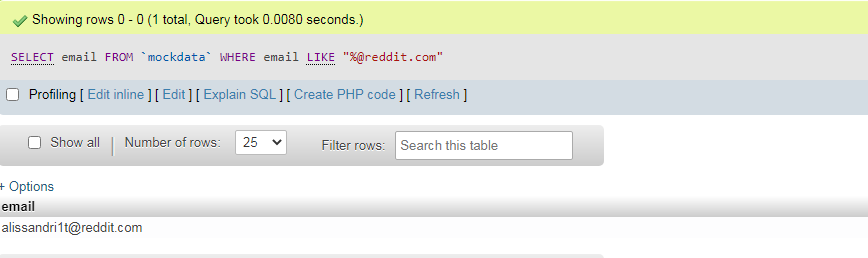
SELECT first\_name,last\_name,ip\_address FROM `mockdata` WHERE inet\_aton(ip\_address) <= inet\_aton('100.255.255.255')



1. **Query that searches for any email address registered with the reddit.com domain**

SELECT email FROM `mockdata` WHERE email LIKE [%@reddit.com](mailto:%25@reddit.com)





1. **Query that inserts four records with incorrectly formatted email addresses**

INSERT INTO `mockdata`(`id`, `first\_name`, `last\_name`, `email`, `studentID`, `ip\_address`) VALUES

(1001,'name1','famil1','zpt@d.com',223,'0.125.36.4'),

(1002,'name2','famil2','opy',450,'55.67.39.14'),

(1003,'name3','famil3','qwe.sdf','45','220.25.136.9'),

(1004,'name4','famil4','oyyj.sdfsdf.sdf@','21','200.25.200.40'),

(1005,'name5','famil5','($/td@sddf.sdsdf',223,'0.125.56.35'),

(1006,'name6','famil6','(%@df.sdfsd',223,'0.125.38.47'),

(1007,'name7','famil7','%@google.com',1000,'29.22.3.40'),

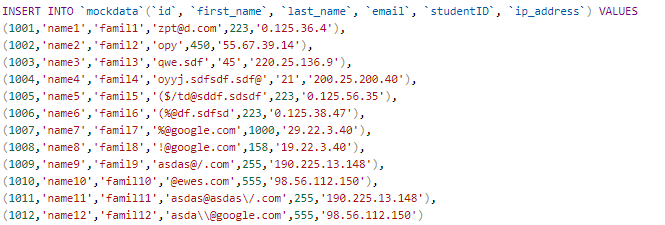
(1008,'name8','famil8','!@google.com',158,'19.22.3.40'),

(1009,'name9','famil9','asdas@/.com',255,'190.225.13.148'),

(1010,'name10','famil10','@ewes.com',555,'98.56.112.150'),

(1011,'name11','famil11','asdas@asdas\/.com',255,'190.225.13.148'),

(1012,'name12','famil12','asda\\@google.com',555,'98.56.112.150')

****

**6.a. Query that extracts all the email addresses that are not in the correct format** ([cited Link](https://www.itsupportguides.com/knowledge-base/server-stuff/sql-query-find-invalid-email-addresses/))

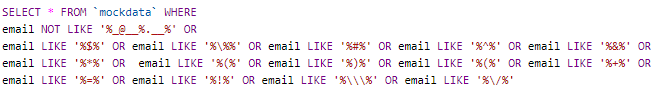
SELECT \* FROM `mockdata` WHERE

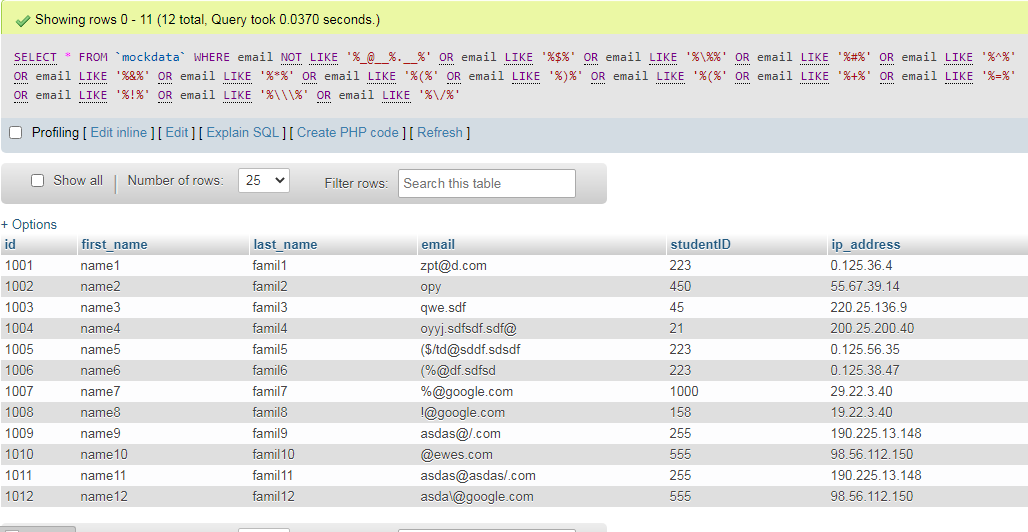
email NOT LIKE '%\_@\_\_%.\_\_%' OR

email LIKE '%$%' OR email LIKE '%\%%' OR email LIKE '%#%' OR email LIKE '%^%' OR email LIKE '%&%' OR

email LIKE '%\*%' OR email LIKE '%(%' OR email LIKE '%)%' OR email LIKE '%(%' OR email LIKE '%+%' OR

email LIKE '%=%' OR email LIKE '%!%' OR email LIKE '%\\\%' OR email LIKE '%\/%'





**6.b. Lookup what MySQL procedures are and write a MySQL procedure that deletes all the email addresses that are not in the correct format**

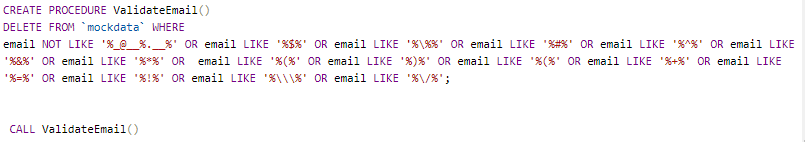
Precedures are like functions and stores bunch of codes and we can call them as a function to run the codes.

CREATE PROCEDURE ValidateEmail()

DELETE FROM `mockdata` WHERE

email NOT LIKE '%\_@\_\_%.\_\_%' OR email LIKE '%$%' OR email LIKE '%\%%' OR email LIKE '%#%' OR email LIKE '%^%' OR email LIKE '%&%' OR email LIKE '%\*%' OR email LIKE '%(%' OR email LIKE '%)%' OR email LIKE '%(%' OR email LIKE '%+%' OR email LIKE '%=%' OR email LIKE '%!%' OR email LIKE '%\\\%' OR email LIKE '%\/%';

Call ValidateEmail()



**Part 2 - Database Design**

1. **Design a database based on the given description. Your output should be a set of named tables with mock data inserted. Column names and data types must also be included.**

**Table1**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **●**Item ID  **(VARCHAR)** | Item Name  **(VARCHAR)** | Item Category  **(VARCHAR)** | Room code  **(VARCHAR)** | Building ID  **(VARCHAR)** | Room Number  **(VARCHAR)** | Building Name  **(VARCHAR)** | Building manager | | Manager ID  **(VARCHAR)** |
| Name  **(VARCHAR)** | Family  **(VARCHAR)** |
| 100 | Table | Furniture | A1 | 1 | 12 | Center | Jack | Beiber | 1100 |
| 101 | Table | Furniture | B1 | 1 | 15 | Center | Jack | Beiber | 1100 |
| 102 | Table | Furniture | C2 | 2 | 12 | Production | Michael | Gainey | 1200 |
| 103 | Table | Furniture | D2 | 2 | 13 | Production | Michael | Gainey | 1200 |
| 104 | Table | Furniture | B2 | 2 | 10 | Design | Sara | Ford | 1300 |
| 105 | Chair | Furniture | B3 | 3 | 12 | Center | Jack | Beiber | 1100 |
| 106 | Chair | Furniture | C1 | 1 | 15 | Center | Jack | Beiber | 1100 |
| 107 | Chair | Furniture | C2 | 2 | 15 | Center | Jack | Beiber | 1100 |
| 108 | Chair | Furniture | B1 | 1 | 12 | Production | Michael | Gainey | 1200 |
| 109 | Chair | Furniture | B3 | 3 | 13 | Production | Michael | Gainey | 1200 |
| 110 | Chair | Furniture | A2 | 2 | 10 | Design | Sara | Ford | 1300 |
| 111 | Chair | Furniture | A3 | 3 | 10 | Design | Sara | Ford | 1300 |
| 112 | Monitor | Computer | A1 | 1 | 12 | Center | Jack | Beiber | 1100 |
| 113 | Monitor | Computer | C3 | 3 | 15 | Production | Michael | Gainey | 1200 |
| 114 | Monitor | Computer | C2 | 2 | 11 | Design | Sara | Ford | 1300 |
| 115 | Monitor | Computer | C1 | 1 | 12 | Design | Sara | Ford | 1300 |
| 116 | Keyboard | Computer | A3 | 3 | 15 | Center | Jack | Beiber | 1100 |
| 117 | Keyboard | Computer | B2 | 2 | 10 | Production | Michael | Gainey | 1200 |
| 118 | Keyboard | Computer | B3 | 3 | 12 | Design | Sara | Ford | 1300 |
| 119 | Keyboard | Computer | A1 | 1 | 15 | Design | Sara | Ford | 1300 |
| 120 | Mouse | Computer | A2 | 2 | 11 | Center | Jack | Beiber | 1100 |
| 122 | Mouse | Computer | C3 | 3 | 12 | Production | Michael | Gainey | 1200 |
| 123 | Mouse | Computer | B3 | 3 | 15 | Design | Sara | Ford | 1300 |
| 124 | Mouse | Computer | B2 | 2 | 10 | Design | Sara | Ford | 1300 |
| 125 | Printer HP | Printer | A1 | 1 | 15 | Center | Jack | Beiber | 1100 |
| 126 | Printer HP | Printer | B2 | 2 | 12 | Production | Michael | Gainey | 1200 |
| 127 | Printer Canon | Printer | C1 | 1 | 11 | Design | Sara | Ford | 1300 |

**First step: What is the primary key?**

In Table 1, “Item ID” is the primary key. Because by having the Item ID we can retrieve all other attributes and it has unique values for each row.

**Second step: Check for 1stNF**

The table satisfies the 1stNF as each column contains atomic values and there are no repeating groups of columns.

**Third step: Check for 2ndNF**

For 2ndNF satisfaction, the table should be in 1st normal form and all the columns depend on the table’s primary key. It means we shouldn’t have **partial dependence**. As we don’t have **composite keys** in this table so we are not facing partial dependency hence the table satisfies the 2ndNF.

**Fourth step: Check for 3rdNF and BCNF**

For 3rdNF satisfaction, we shouldn’t see any **transition dependency**. The indirect dependency that one column has to the primary key column. Here we see that the column Building name has a dependency on Building ID and Manager name has a dependency on Manager ID. Also item type is dependent to item name. So we can get rid of thees columns and make a separate table as below.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Table 2. Items**   |  |  |  | | --- | --- | --- | | **●** Item ID | Item Name | ○ Room code | | | 100 | Table | A1 | | 101 | Table | B1 | | 102 | Table | C2 | | 103 | Table | D2 | | 104 | Table | B2 | | 105 | Chair | B3 | | 106 | Chair | C1 | | 107 | Chair | C2 | | 108 | Chair | B1 | | 109 | Chair | B3 | | 110 | Chair | A2 | | 111 | Chair | A3 | | 112 | Monitor | A1 | | 113 | Monitor | C3 | | 114 | Monitor | C2 | | 115 | Monitor | C1 | | 116 | Keyboard | A3 | | 117 | Keyboard | B2 | | 118 | Keyboard | B3 | | 119 | Keyboard | A1 | | 120 | Mouse | A2 | | 122 | Mouse | C3 | | 123 | Mouse | B3 | | 124 | Mouse | B2 | | 125 | Printer HP | A1 | | 126 | Printer HP | B2 | | 127 | Printer Canon | C1 | | **Table 3 Building**   |  |  |  | | --- | --- | --- | | **●**Build. ID | Build. Nam | Manager ID | | | 1 | Center | 12300 | | 2 | Production | 12400 | | 3 | Design | 12500 |   **Table 4. Employee**   |  |  |  | | --- | --- | --- | | **●**Employee ID | Name | Family | | 1100 | Jack | Beiber | | 1200 | Michael | Gainey | | 1300 | Sara | Ford | | 1000 | Hana | Biuk | | 1400 | Steve | Johnson |   **Table 5. Room**   |  |  | | --- | --- | | **●**Room ID | Duild. ID | | A1 | 1 | | A2 | 2 | | A3 | 3 | | B1 | 1 | | B2 | 2 | | B3 | 3 | | C1 | 1 | | C2 | 2 | | C3 | 3 | | **Table 6. Item category**   |  |  | | --- | --- | | Item name | category | | | Table | Computer | | Chair | Production | | Monitor | Computer | | Keyboard | Computer | | Mouse | Computer | | Printer HP | Printer | | Printer Canon | Printer | |

1. **Research insertion, deletion, and update anomalies as they relate to database design. Normalized databases should not suffer from any of the aforementioned anomalies. Based on the database design you constructed in Part 2 – Q1, provide examples of how your normalized database design does not suffer from insertion, deletion, and update anomalies (one example for each anomaly is sufficient).**

Our normalized databased in question 1 is completely robust to the anomalies.

*Update anomaly*: for example if the manager of one building changes it is very easy to update the table 3 (Buildings) whitout applying changes in other tables.

*Insertion anomaly*: suppose a new building is added to the company. So clearly it is needed to add the information about the new building inserted in table 3.

*Deletion Anamaly*: If one item for example a table in room C1 was broken and the manager wants ot throw it away so it is very easy to remove this item in Table 2 without data loss about other columns such as roome ID, building ID or manager of the building.

1. **Consider the following relational instance:**

*CarSales(car#, date\_sold, salesperson#, commission%, discount\_amount)*

**Assume that a car may be sold by multiple salespeople thus the primary key is [car#, salesperson#]. Some additional dependencies to consider are:**

*date\_sold → discount\_amount*

*salesperson# → commission%*

**Based on the given primary key, is this relational instance in 1NF, 2NF, or 3NF? Why or why not? How would you successively normalize it completely (up to 3NF)?.**

**1stNF:** The 1NF is satisfied as all cells seem to have atomic values and also there is no more than one value in each attribute per row.

**2ndNF**: It is not satisfied as there is partial dependencies and actually we have composite keys: Car and SalesPerson. To satisfy the 2ndNF we should separate salesman and commission in a separate instance:

*CAR\_SALE1(Car#, Salesman#, DateSold, DiscountAmount)*

*CAR\_SALE2(Salesman#, Commission%)*

**3rdNF**: In 3rd normal form we shouldn’t have transitive dependency. But here there is a transitive dependency between It is date\_sold and discount that we should make a separate instance as fallows:

*Car\_sales1(Car#, Salesman#, DateSold)*

*Car\_sales2(Salesman#, Commission%)*

*Car\_sales3(DateSold, DiscountAmount)*