1DV503 Database Technology and Modeling

Firas Moussa
School of Computer Science, Physics
and Mathematics, Linnaeus University, Sweden
fm222us@student.lnu.se

Task 1 SQL queries using MySQL Workbench DBMS

Query 1

SQL:

```
SELECT e.fname, e.lname, e.ssn, p.pname
FROM employee as e
JOIN works_on wo ON e.ssn = wo.essn
JOIN project p ON wo.pno = p.pnumber
WHERE p.pname = 'Middleware' or p.pname = 'DatabaseSystems'
;
```

Result:

#	fname	lname	ssn	pname
1	Evan	Wallis	222222200	DatabaseSystems
2	Josh	Zell	222222201	DatabaseSystems
3	Andy	Vile	222222202	DatabaseSystems
4	Tom	Brand	222222203	DatabaseSystems
5	Jenny	Vos	222222204	DatabaseSystems
6	Chris	Carter	222222205	DatabaseSystems
7	Arnold	Head	666666608	DatabaseSystems
8	Red	Bacher	666666613	DatabaseSystems
9	Kim	Grace	333333300	Middleware
10	Jeff	Chase	333333301	Middleware
11	Helga	Pataki	666666609	Middleware
12	Red	Bacher	666666613	Middleware

Query 2

SQL:

```
SELECT e.fname, e.lname, e.ssn, wo.hours
FROM employee e
JOIN works_on wo ON e.ssn = wo.essn
WHERE wo.hours > 40
;
```

Result:

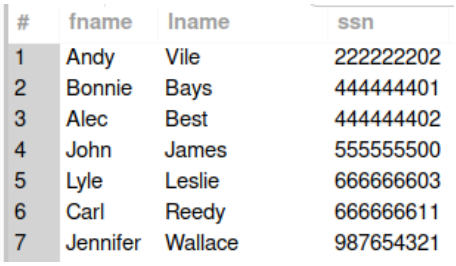
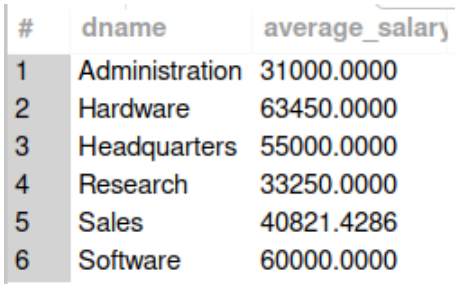
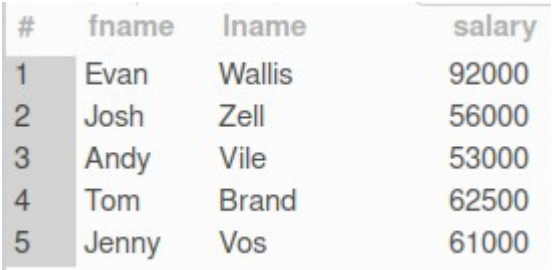
#	fname	lname	ssn	hours
1	Josh	Zell	222222201	48
2	Jeff	Chase	333333301	46
3	Nandita	Ball	555555501	44

Query 3

SQL:

```
SELECT p.pnumber, p.dnum, e.lname, e.address, e.bdate
```

	FROM project p JOIN department d ON p.dnum = d.dnumber JOIN employee e ON d.mgrssn = e.ssn WHERE p.plocation = 'Houston' ; **Result:** | # | pnumber | dnum | lname | address | bdate | |---|---|---|---|---|---| | 1 | 3 | 5 | Wong | 638 Voss, Houston, TX | 1945-12-08 | | 2 | 20 | 1 | Borg | 450 Stone, Houston, TX | 1927-11-10 |
Query 4	**SQL:** SELECT e.fname, e.lname, s.fname, s.lname FROM employee e JOIN employee s ON e.superssn = s.ssn WHERE e.superssn is not null ; **Result:** | # | fname | lname | fname | lname | |---|---|---|---|---| | 1 | Jon | Jones | Jared | James | | 2 | Justin | Mark | Jared | James | | 3 | Brad | Knight | Jared | James | | 4 | John | Smith | Franklin | Wong | | 5 | Josh | Zell | Evan | Wallis | | 6 | Andy | Vile | Evan | Wallis | | 7 | Tom | Brand | Evan | Wallis | | 8 | Jenny | Vos | Josh | Zell | | 9 | Chris | Carter | Josh | Zell | | 10 | Jeff | Chase | Kim | Grace | | 11 | Franklin | Wong | James | Borg | | 12 | Bonnie | Bays | Alex | Freed | | 13 | Alec | Best | Alex | Freed | | 14 | Sam | Snedden | Alex | Freed | | 15 | Joyce | English | Franklin | Wong | | 16 | Nandita | Ball | John | James | | 17 | Jill | Jarvis | Bob | Bender | | 18 | Kate | King | Bob | Bender | | 19 | Lyle | Leslie | Jill | Jarvis | | 20 | Billie | King | Lyle | Leslie | | 21 | Jon | Kramer | Lyle | Leslie | | 22 | Ray | King | Billie | King | | 23 | Gerald | Small | Kate | King | | 24 | Arnold | Head | Kate | King | | 25 | Helga | Pataki | Kate | King | | 26 | Naveen | Drew | Gerald | Small | | 27 | Carl | Reedy | Naveen | Drew | | 28 | Sammy | Hall | Carl | Reedy | | 29 | Red | Bacher | Sammy | Hall | | 30 | Ramesh | Narayan | Franklin | Wong | | 31 | Jennifer | Wallace | James | Borg | | 32 | Ahmad | Jabbar | Jennifer | Wallace | | 33 | Alicia | Zelaya | Jennifer | Wallace |
Query 5	**SQL:** SELECT fname, lname, ssn FROM employee WHERE sex = 'F' AND address LIKE '%Houston%'; **Result:** | # | fname | lname | ssn | |---|---|---|---| | 1 | Joyce | English | 453453453 |
Query 6	**SQL:** SELECT fname, lname, ssn FROM employee WHERE MONTH(bdate) = 6;

	Result: 
Query 7	**SQL:** SELECT d.dname, AVG(e.salary) AS average_salary FROM department d JOIN employee e ON d.dnumber = e.dno GROUP BY d.dname; **Result:** 
Query 8	**SQL:** SELECT e.fname, e.lname FROM employee e JOIN works_on wo ON e.ssn = wo.essn WHERE wo.pno IS NULL; **Result:** **0 Matches – Indicating all employees are on a project**
Query 9	**SQL:** SELECT e.fname, e.lname, e.salary FROM employee e JOIN works_on wo ON e.ssn = wo.essn JOIN project p ON wo.pno = p.pnumber WHERE e.dno = 7 AND e.salary > 50000 AND p.pname = 'DatabaseSystems'; **Result:** 

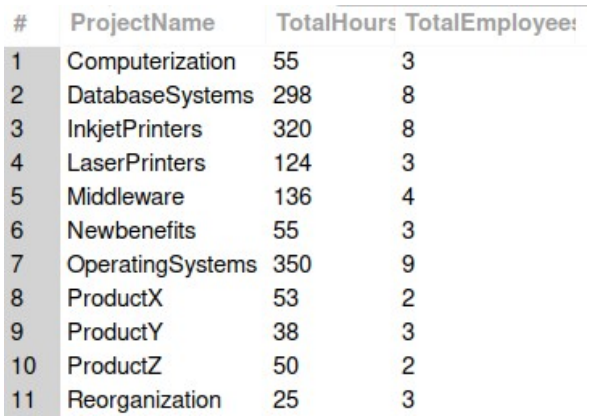
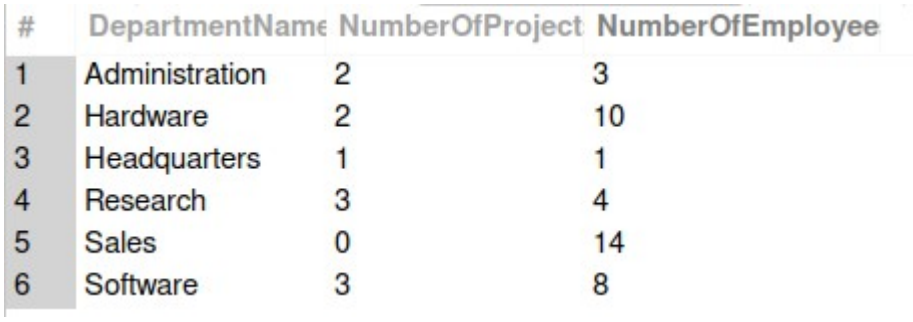
#	fname	lname	ssn
1	Andy	Vile	222222202
2	Bonnie	Bays	444444401
3	Alec	Best	444444402
4	John	James	555555500
5	Lyle	Leslie	666666603
6	Carl	Reedy	666666611
7	Jennifer	Wallace	987654321

#	dname	average_salary
1	Administration	31000.0000
2	Hardware	63450.0000
3	Headquarters	55000.0000
4	Research	33250.0000
5	Sales	40821.4286
6	Software	60000.0000

#	fname	lname	salary
1	Evan	Wallis	92000
2	Josh	Zell	56000
3	Andy	Vile	53000
4	Tom	Brand	62500
5	Jenny	Vos	61000

Query 10	**SQL:** SELECT fname, lname FROM employee WHERE superssn = 333445555 AND address LIKE '%Houston, TX' ; **Result:** | #	fname	lname | |---|---|---| | 1 | John | Smith | | 2 | Joyce | English |
Query 11	**SQL:** SELECT e.fname, e.lname FROM employee e JOIN department d ON e.dno = d.dnumber WHERE d.dnumber IN (SELECT d.dnumber FROM employee e JOIN department d ON e.dno = d.dnumber GROUP BY d.dnumber HAVING AVG(e.salary) = (SELECT MAX(avg_salary) FROM (SELECT AVG(e.salary) AS avg_salary FROM employee e JOIN department d ON e.dno = d.dnumber GROUP BY d.dnumber) AS department_avg_salary)); **Result:** | #	fname	lname | |---|---|---| | 1 | Evan | Wallis | | 2 | Josh | Zell | | 3 | Andy | Vile | | 4 | Tom | Brand | | 5 | Jenny | Vos | | 6 | Chris | Carter | | 7 | Alex | Freed | | 8 | Bonnie | Bays | | 9 | Alec | Best | | 10 | Sam | Snedden |
Query 12	**SQL:** SELECT d.dnumber,

	d.dname, COUNT(e.ssn) AS num_employees, AVG(e.salary) AS avg_salary FROM department d JOIN employee e ON d.dnumber = e.dno GROUP BY d.dnumber, d.dname HAVING AVG(e.salary) > 35000; **Result:** | # | dnumber | dname | num_employees | avg_salary | | --- | --- | --- | --- | --- | | 1 | 7 | Hardware | 10 | 63450.0000 | | 2 | 1 | Headquarters | 1 | 55000.0000 | | 3 | 8 | Sales | 14 | 40821.4286 | | 4 | 6 | Software | 8 | 60000.0000 |
Query 13	**SQL:** SELECT d.dependent_name, d.relationship FROM dependent d JOIN employee e ON d.essn = e.ssn WHERE e.superssn = '333445555' ORDER BY d.dependent_name ASC; **Result:** | # | dependent_name | relationship | | --- | --- | --- | | 1 | Alice | Daughter | | 2 | Elizabeth | Spouse | | 3 | Michael | Son |
Query 14	**SQL:** SELECT p.pname AS ProjectName, SUM(wo.hours) AS TotalHours, COUNT(DISTINCT wo.essn) AS TotalEmployees FROM project p JOIN works_on wo ON p.pnumber = wo.pno GROUP BY p.pname ORDER BY

	p.pname; **Result:** 
Query 15	**SQL:** SELECT d.dname AS DepartmentName, COUNT(DISTINCT p.pnumber) AS NumberOfProjects, COUNT(DISTINCT e.ssn) AS NumberOfEmployees FROM department d LEFT JOIN project p ON d.dnumber = p.dnum LEFT JOIN employee e ON d.dnumber = e.dno GROUP BY d.dname ORDER BY d.dname; **Result:** 

#	ProjectName	TotalHours	TotalEmployees
1	Computerization	55	3
2	DatabaseSystems	298	8
3	InkjetPrinters	320	8
4	LaserPrinters	124	3
5	Middleware	136	4
6	Newbenefits	55	3
7	OperatingSystems	350	9
8	ProductX	53	2
9	ProductY	38	3
10	ProductZ	50	2
11	Reorganization	25	3

#	DepartmentName	NumberOfProjects	NumberOfEmployees
1	Administration	2	3
2	Hardware	2	10
3	Headquarters	1	1
4	Research	3	4
5	Sales	0	14
6	Software	3	8

Task 2 Functional Dependencies

Task **2.1 Solution**

Primary key: Composite Key of **EMPLOYEE_ID** and **JOB_ID.**
Since it seems that an employee can have multiuple job roles, we need a composite primary key to uniquely identify each row in the table.

Functional Dependency	Explanation/Example
{employee_ID, job_ID} → name, job_title, post_code, city:	The combination of **EMPLOYEE_ID** and **JOB_ID** as the primary key, uniquely determines the employee's name, job title, postal code, and city.
{employee_ID} → name, post_code, city:	The **EMPLOYEE_ID** uniquely determines the employee's name, postal code, and city. JOB_ID is needed to identify the unique JOB_TITLE as seen in the dependency above.
{post_code} → city:	The **POST_CODE** uniquely determines the CITY, as per real-world application.

2.2 Solution

Primary key: Composite primary key can be the **Product** and **Part** to uniquely identify each row.

Functional dependencies:

- **{Product, Part} →{Material, Type, Quantity, Product_Price, Part_Price, Part_Supplier}**
- **{Part} →{Type, Part_Price, Part_Supplier}** (Assuming that each part is unique).
- **{Product, Quantity, Part_Supplier} →{Product_Price}** (Assuming that every instance of a product has the same price).

Anomaly	*Justification/Explanation*
Redundancy	Product_Price might be repeated for the same product across different rows, which is redundant if the price does not change based on parts.

Update	If a part's price changes, it would require updates in multiple rows where the part is listed, increasing the risk of inconsistent data.
Deletion	Deleting a product that uses a unique part would also remove the details of that part from the database.
Insertion	It's not possible to insert a new part into the database without associating it with a product, which might not be desired if the part is not yet assigned to any product.

Task 3 Normalization

3.1 Solution

Original Table Structure:

- Product Part
- Material
- Type
- Quantity
- Product_Price
- Part_Price
- Part_Supplier

Decomposition:

1. **Product Table** (Stores unique product information)
    - **Primary Key**: Product_ID (new attribute to uniquely identify each product)
    - Material
    - Product_Price
2. **Part Table** (Stores unique parts information)
    - **Primary Key**: Part_ID (new attribute to uniquely identify each part)
    - Part
    - Type
    - Part_Price
    - Part_Supplier
3. **ProductPart Table** (Associates products with their parts and stores information about this relationship)
    - **Primary Key**: Product_ID, Part_ID (Composite Key)
    - Quantity

Normalization Justification:

- Moving to **1NF**: Eliminated repeating groups by creating a separate table for ProductPart, ensuring atomicity.
- Moving to **2NF**: Removed partial dependency by separating product and part attributes into their respective tables, ensuring that all non-key attributes are fully functionally dependent on the primary key.
- Moving to **3NF**: Removed transitive dependency by ensuring that non-key attributes (e.g., Part_Price, Part_Supplier in the Part table) depend only on the primary key and not on other non-key attributes.

Anomalies and Redundancies Resolution:

- **Update Anomaly**: Resolved as changing a part's price or supplier now only requires updating a single record in the Part table.
- **Deletion Anomaly**: Resolved as deleting a product-part relationship doesn't affect the existence of the part or product information.
- **Insertion Anomaly**: Resolved as parts can be added to the Part table without being part of a product, and products can be added to the Product table without immediately specifying parts.

3.2 Solution

Explanation	**Result**
Based on the given primary key, is this relation in 1NF, 2ND, or 3NF? *Your answer here:* *This relation is in **1NF** because it has a primary key, and all its attributes contain atomic values. It's not in **2NF** or **3NF** because:* • It has partial dependency: attributes like Discount depend on part of the primary key (Date_sold), not the entire composite key. • It has transitive dependency: Commission depends on Salesperson, which is not part of the primary key.	Diagram before normalization: Given: • **Primary Key**: {Car, Salesperson} • **Dependencies**: Date_sold → Discount, Salesperson → Commission

Step 1: Remove Partial Dependencies (Move to 2NF) *After the first step, there are no transitive dependencies within CarSale, as all attributes in each table directly depend on the primary key, not on other non-key attributes. The decomposition into Discount and SalespersonCommission tables during the move to 2NF also addressed transitive dependencies.*	**CarSale Table**: • **Primary Key**: {Car, Salesperson} • Date_sold • **Discount Table**: • **Primary Key**: Date_sold • Discount • **SalespersonCommission Table**: • **Primary Key**: Salesperson • Commission
Final Results in 3NF ***Explanation:*** • **CarSale** table now focuses on the relationship between cars and salespeople, and the date a car was sold. • **Discount** table eliminates partial dependencies by associating the discount directly with the sale date. • **SalespersonCommission** table removes transitive dependencies, associating salespeople directly with their commission, independent of the specific car sales.	**Final Tables in 3NF:** 1. **CarSale**: • **Primary Key**: {Car, Salesperson} • Date_sold 2. **Discount**: • **Primary Key**: Date_sold • Discount 3. **SalespersonCommission**: • **Primary Key**: Salesperson • Commission