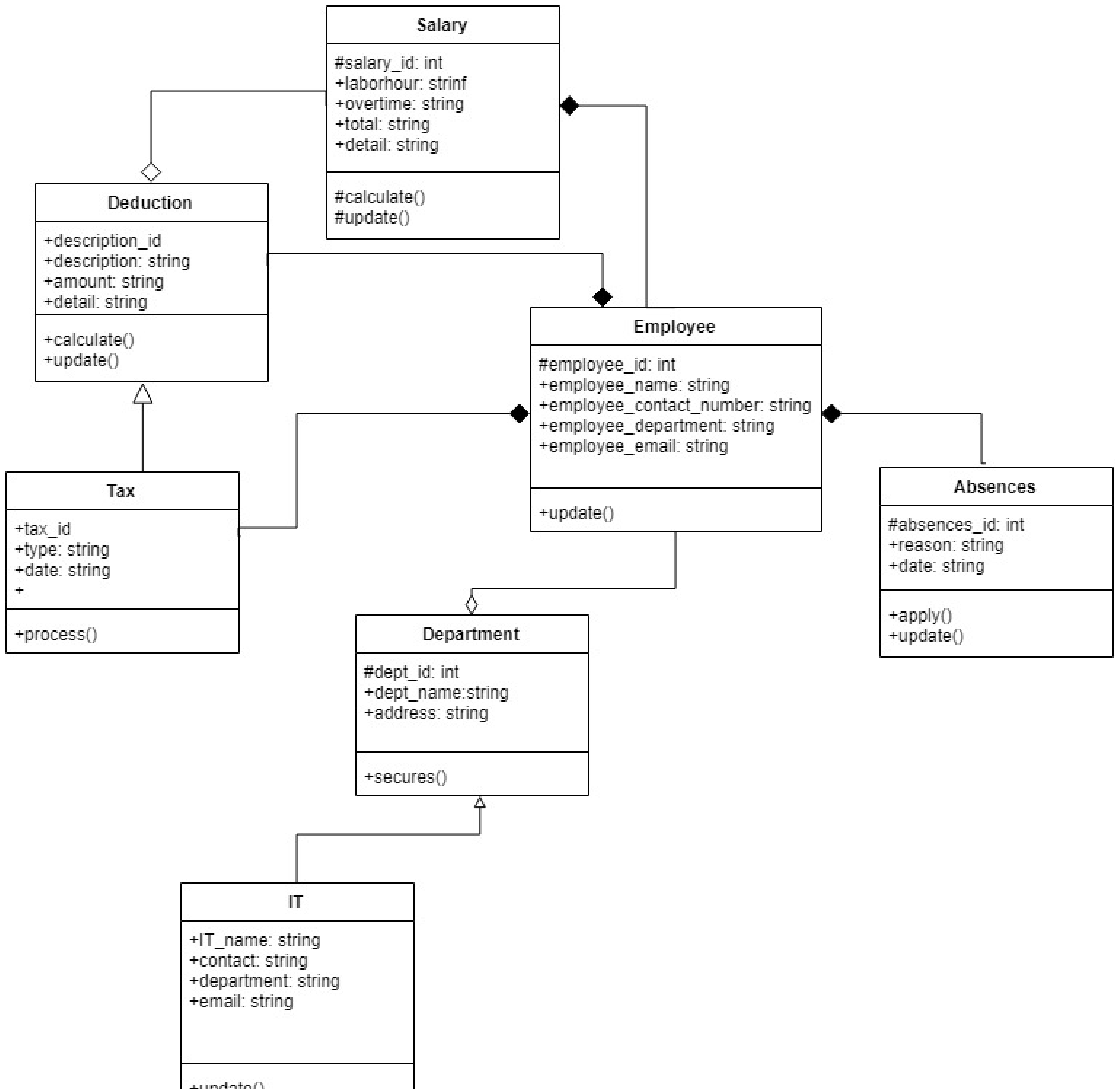
# Appendix 1 - CET341 Assignment Structure Template

## Task One:

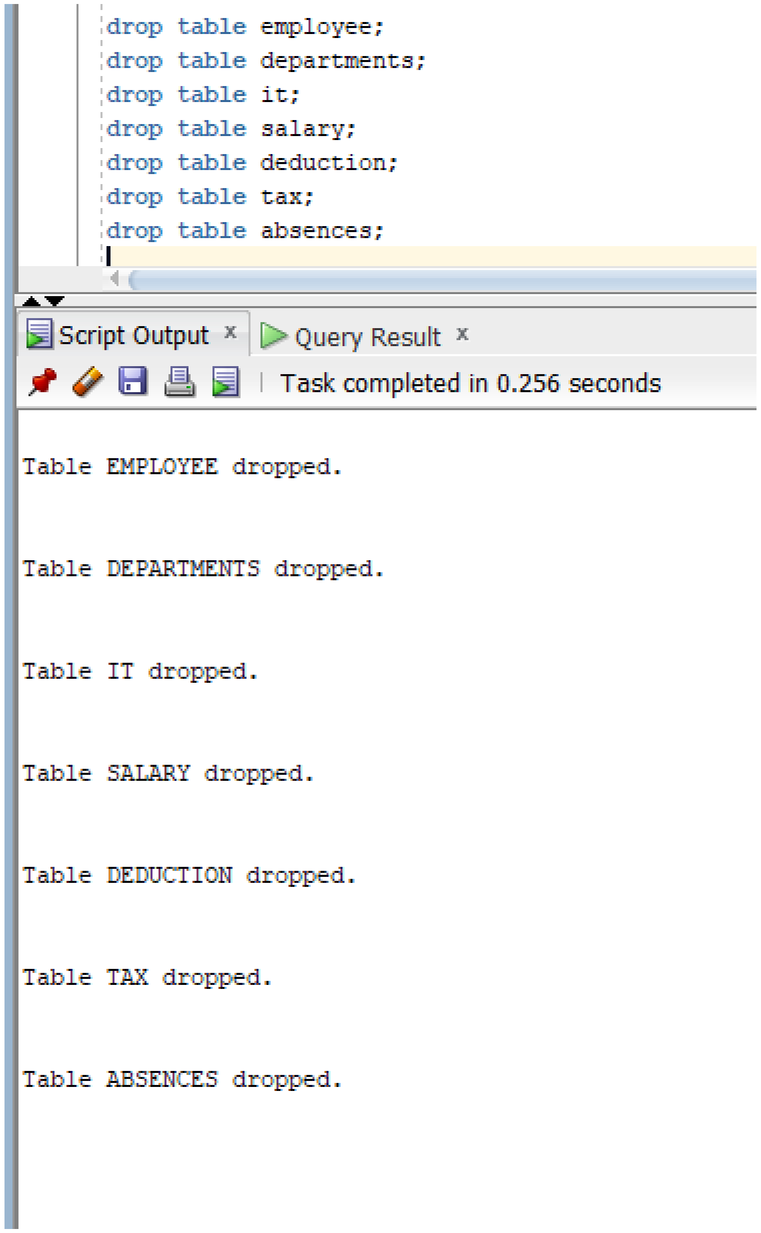
*UML class diagram*


Salary
#salary_id: int
+laborhour: strinf
+overtime: string
+total: string
+detail: string
#calculate()
#update()
Deduction
+description_id
+description: string
+amount: string
+detail: string
+calculate()
+update()
Employee
#employee_id: int
+employee_name: string
+employee_contact_number: string
+employee_department: string
+employee_email: string
+update()
Tax
+tax_id
+type: string
+date: string
+
+process()
Absences
#absences_id: int
+reason: string
+date: string
+apply()
+update()
Department
#dept_id: int
+dept_name:string
+address: string
+secures()
IT
+IT_name: string
+contact: string
+department: string
+email: string

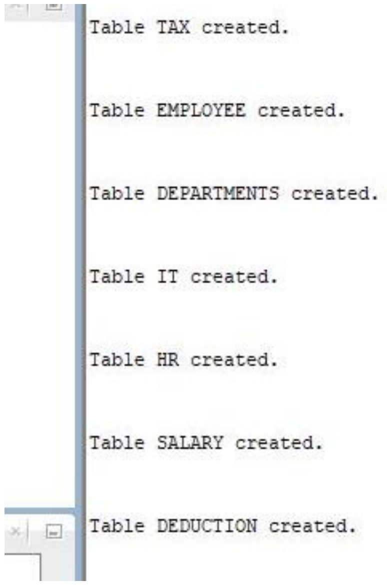
**Task Two:**

*screenshots to demonstrate that ALL of your SQL code works.*

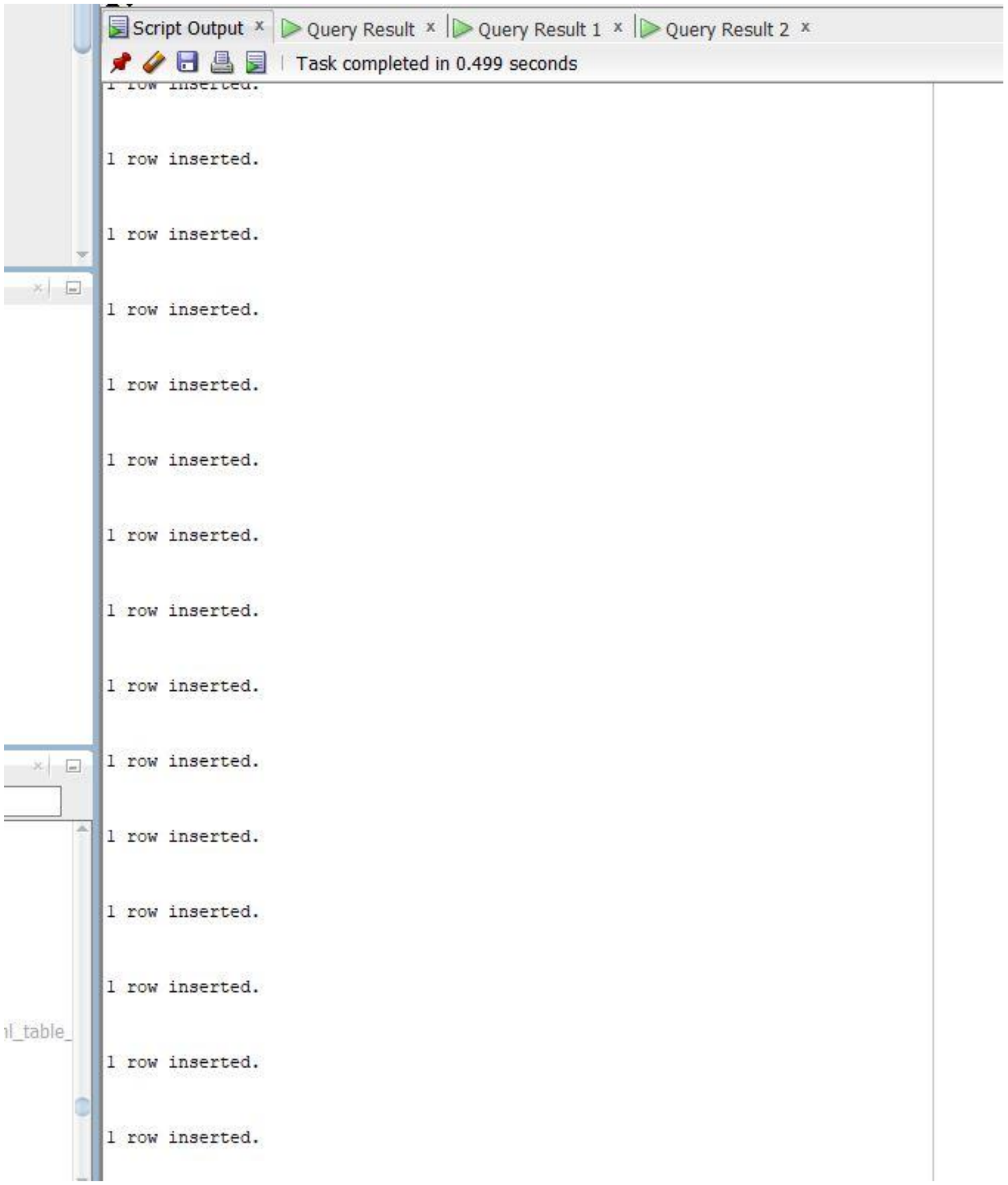
Dropping and creating all tables


```
drop table employee;
drop table departments;
drop table it;
drop table salary;
drop table deduction;
drop table tax;
drop table absences;
```

```
Table EMPLOYEE dropped.


Table DEPARTMENTS dropped.


Table IT dropped.


Table SALARY dropped.


Table DEDUCTION dropped.


Table TAX dropped.


Table ABSENCES dropped.
```



```
Table TAX created.


Table EMPLOYEE created.


Table DEPARTMENTS created.


Table IT created.


Table HR created.


Table SALARY created.


Table DEDUCTION created.
```

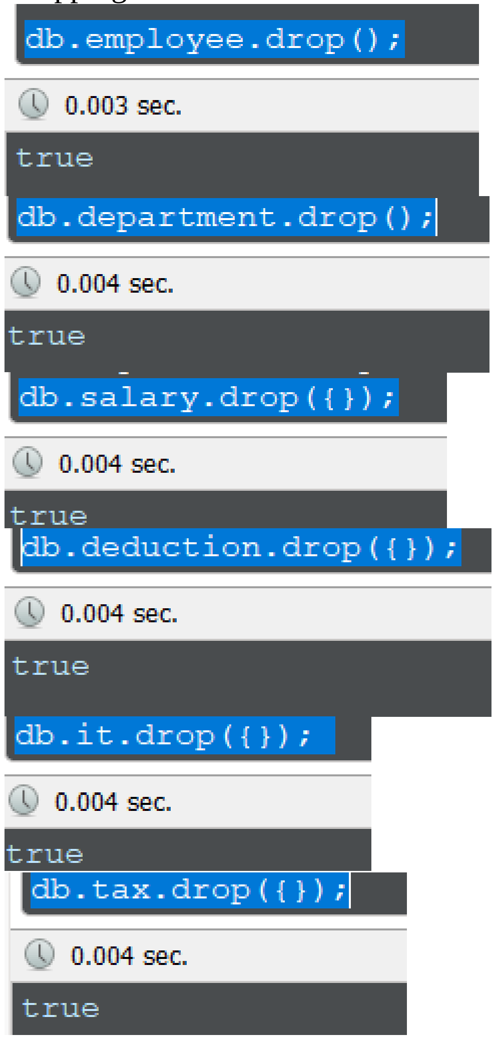


Task completed in 0.499 seconds

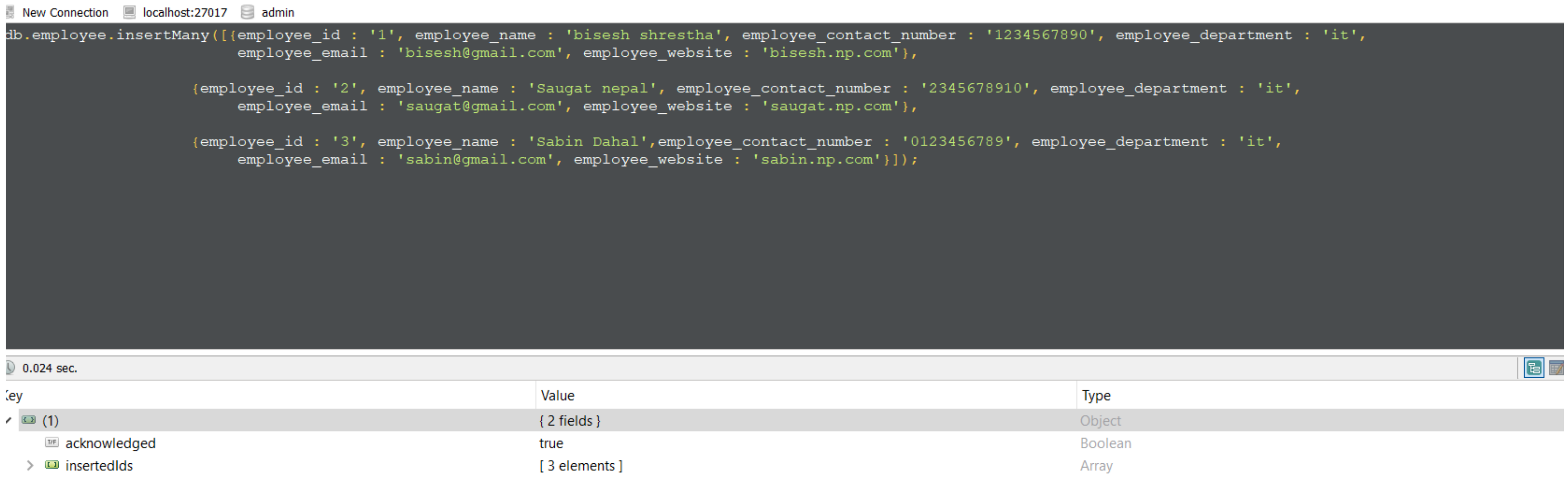
```
1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.
```

**Task Three:**

*screenshots to demonstrate that all of your MongoDB code works.*
Dropping the 7 collections then inserting data.

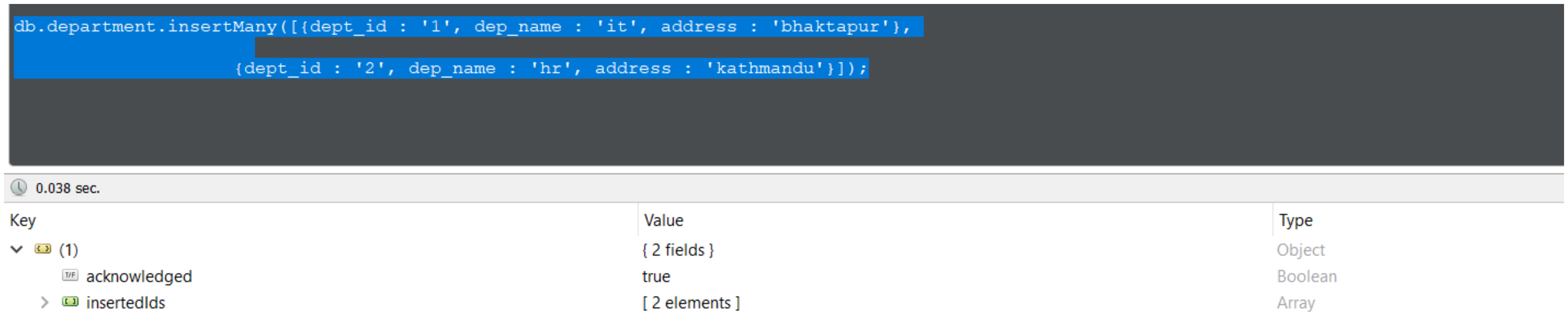


```
db.employee.drop();
0.003 sec.
true
db.department.drop();
0.004 sec.
true
db.salary.drop({});
0.004 sec.
true
db.deduction.drop({});
0.004 sec.
true
db.it.drop({});
0.004 sec.
true
db.tax.drop({});
0.004 sec.
true
```



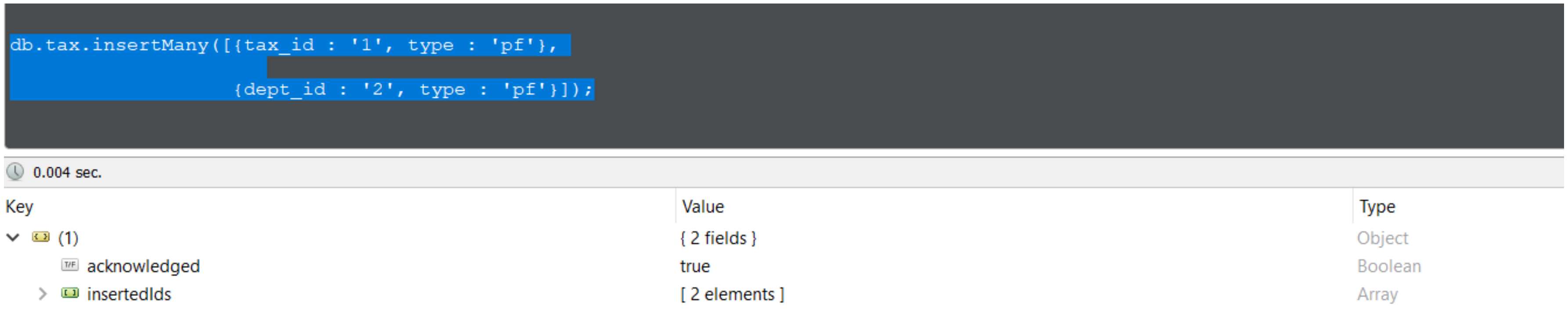
```
db.employee.insertMany([{employee_id : '1', employee_name : 'bisesh shrestha', employee_contact_number : '1234567890', employee_department : 'it',
                         employee_email : 'bisesh@gmail.com', employee_website : 'bisesh.np.com'},

                    {employee_id : '2', employee_name : 'Saugat nepal', employee_contact_number : '2345678910', employee_department : 'it',
                         employee_email : 'saugat@gmail.com', employee_website : 'saugat.np.com'},

                    {employee_id : '3', employee_name : 'Sabin Dahal',employee_contact_number : '0123456789', employee_department : 'it',
                         employee_email : 'sabin@gmail.com', employee_website : 'sabin.np.com'}]);
```

```
0.024 sec.
Key                         Value               Type
(1)                         { 2 fields }        Object
    acknowledged            true                Boolean
    insertedIds             [ 3 elements ]      Array
```



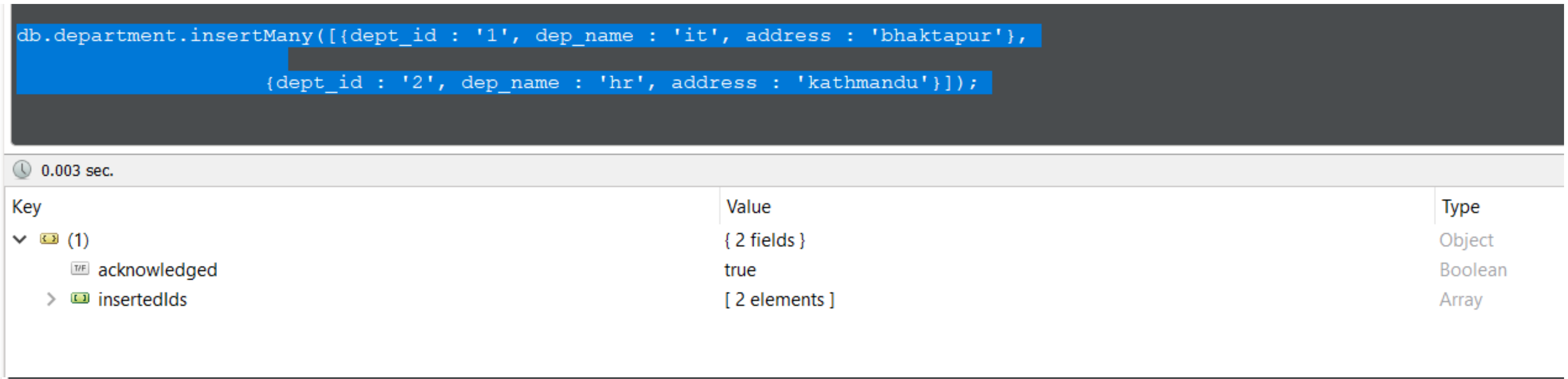
```
db.department.insertMany([{dept_id : '1', dep_name : 'it', address : 'bhaktapur'},

                         {dept_id : '2', dep_name : 'hr', address : 'kathmandu'}]);
```

```
0.038 sec.
Key                         Value               Type
(1)                         { 2 fields }        Object
    acknowledged            true                Boolean
    insertedIds             [ 2 elements ]      Array
```



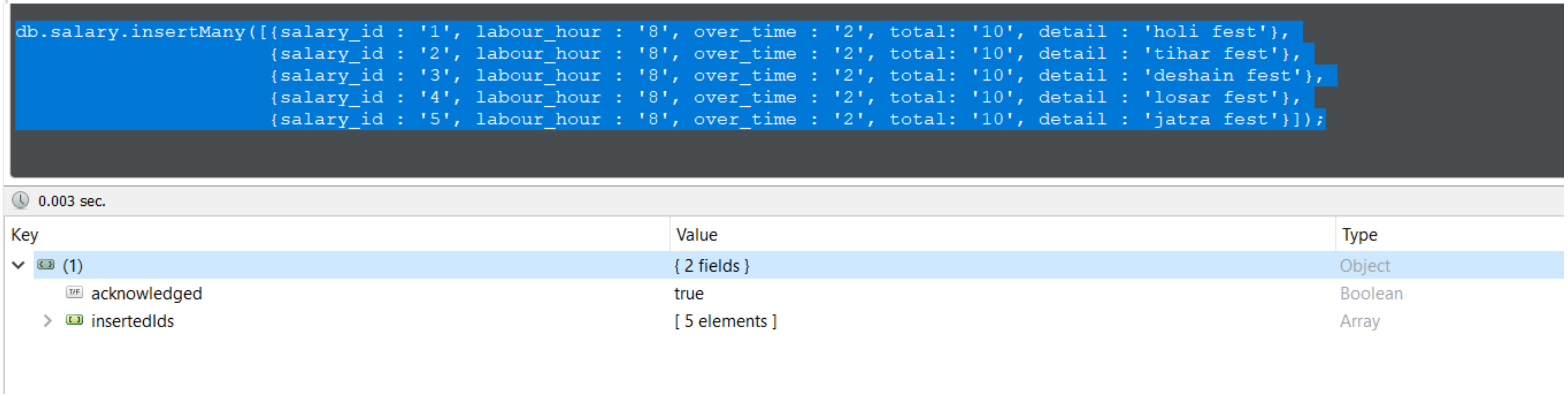
```
db.tax.insertMany([{tax_id : '1', type : 'pf'},
                   {dept_id : '2', type : 'pf'}]);
```

```
0.004 sec.
Key                         Value            Type
⌄ (1)                       { 2 fields }     Object
    acknowledged            true             Boolean
  › insertedIds             [ 2 elements ]   Array
```



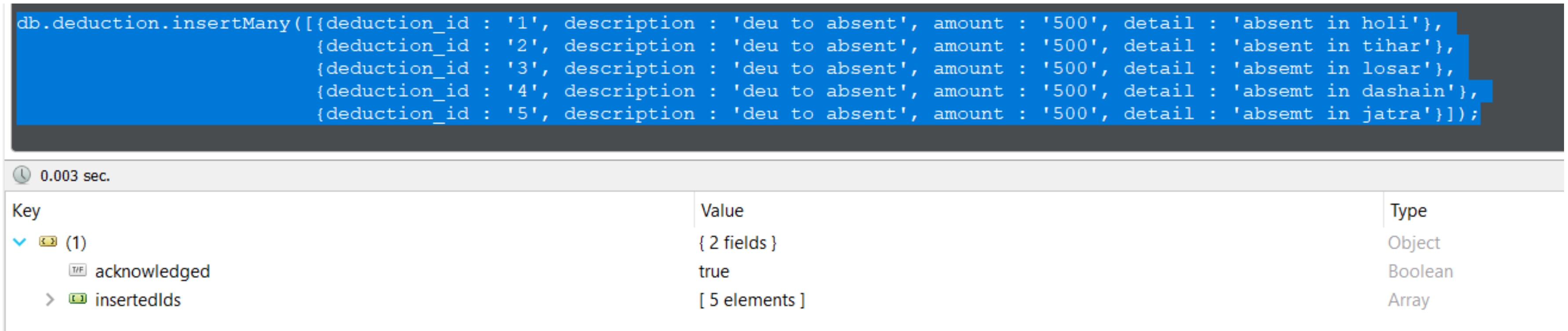
```
db.department.insertMany([{dept_id : '1', dep_name : 'it', address : 'bhaktapur'},
                          {dept_id : '2', dep_name : 'hr', address : 'kathmandu'}]);
```

```
0.003 sec.
Key                         Value            Type
⌄ (1)                       { 2 fields }     Object
    acknowledged            true             Boolean
  › insertedIds             [ 2 elements ]   Array
```



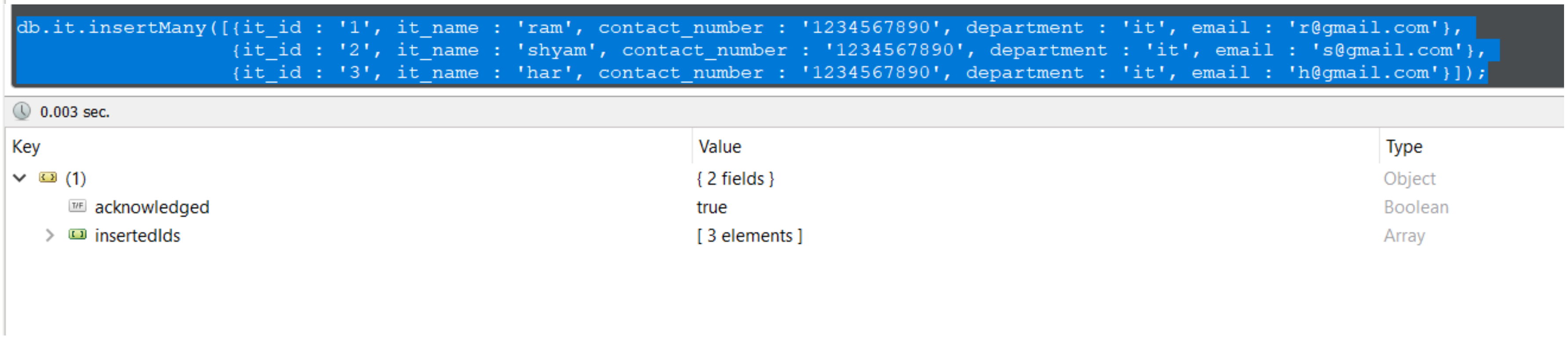
```
db.salary.insertMany([{salary_id : '1', labour_hour : '8', over_time : '2', total: '10', detail : 'holi fest'},
                      {salary_id : '2', labour_hour : '8', over_time : '2', total: '10', detail : 'tihar fest'},
                      {salary_id : '3', labour_hour : '8', over_time : '2', total: '10', detail : 'deshain fest'},
                      {salary_id : '4', labour_hour : '8', over_time : '2', total: '10', detail : 'losar fest'},
                      {salary_id : '5', labour_hour : '8', over_time : '2', total: '10', detail : 'jatra fest'}]);
```

```
0.003 sec.
Key                         Value            Type
⌄ (1)                       { 2 fields }     Object
    acknowledged            true             Boolean
  › insertedIds             [ 5 elements ]   Array
```



```
db.deduction.insertMany([{deduction_id : '1', description : 'deu to absent', amount : '500', detail : 'absent in holi'},
                         {deduction_id : '2', description : 'deu to absent', amount : '500', detail : 'absent in tihar'},
                         {deduction_id : '3', description : 'deu to absent', amount : '500', detail : 'absemt in losar'},
                         {deduction_id : '4', description : 'deu to absent', amount : '500', detail : 'absemt in dashain'},
                         {deduction_id : '5', description : 'deu to absent', amount : '500', detail : 'absemt in jatra'}]);
```

```
0.003 sec.
Key                          Value              Type
(1)                          { 2 fields }       Object
    acknowledged             true               Boolean
    insertedIds              [ 5 elements ]     Array
```



```
db.it.insertMany([{it_id : '1', it_name : 'ram', contact_number : '1234567890', department : 'it', email : 'r@gmail.com'},
                  {it_id : '2', it_name : 'shyam', contact_number : '1234567890', department : 'it', email : 's@gmail.com'},
                  {it_id : '3', it_name : 'har', contact_number : '1234567890', department : 'it', email : 'h@gmail.com'}]);
```

```
0.003 sec.
Key                          Value              Type
(1)                          { 2 fields }       Object
    acknowledged             true               Boolean
    insertedIds              [ 3 elements ]     Array
```

## Tasks Four to Six

**Query a: A join of three or more tables – you should consider various types of join in this query (e.g. inner join, left/right/full outer joins, etc.) and the query must include a restriction on the rows selected**

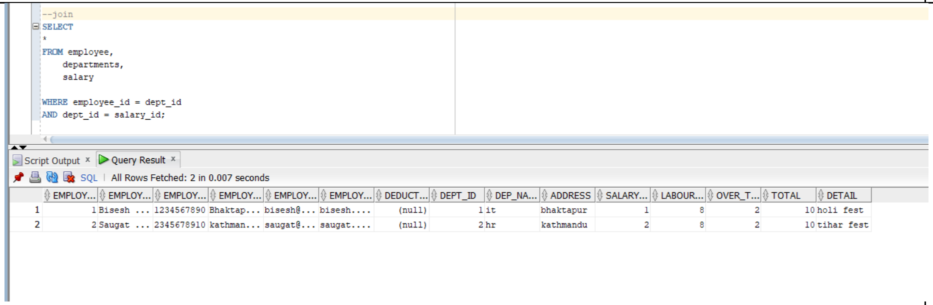
get all the emails registered to department or employees

| **SQL code** | **MongoDB code** |
|---|---|
| *SELECT*<br>***<br>*FROM employee,*<br>   *departments,*<br>   *salary* | db.employee.aggregate([<br>{ $unionWith: "salary" },<br>{ $unionWith: "department" },<br>{<br>$group: { |

*WHERE employee_id = dept_id*
*AND dept_id = salary_id;*

```
        _id: "$_id",
        salary: { $push: "$detail" },
        department: { $push: "$dep_name" },
        employee: { $push: "$employee_name" }}},
 {$project: {
        _id: 1,
        employee: 1,
        department: 1,
        salary: 1,
    }
 },
 { $unwind: { path:
"$employee",preserveNullAndEmptyArrays: true } },
 { $unwind: { path:
"$department",preserveNullAndEmptyArrays: true } },
 { $unwind: { path: "$salary",
preserveNullAndEmptyArrays:true } },
 { $sort: { _id: 1 } },]);
```

**Screenshots**

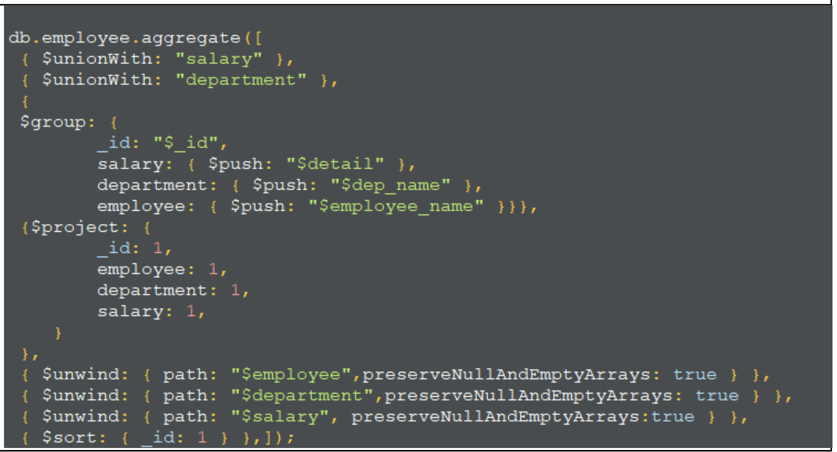


```
--join
SELECT
*
FROM employee,
    departments,
    salary

WHERE employee_id = dept_id
AND dept_id = salary_id;
```

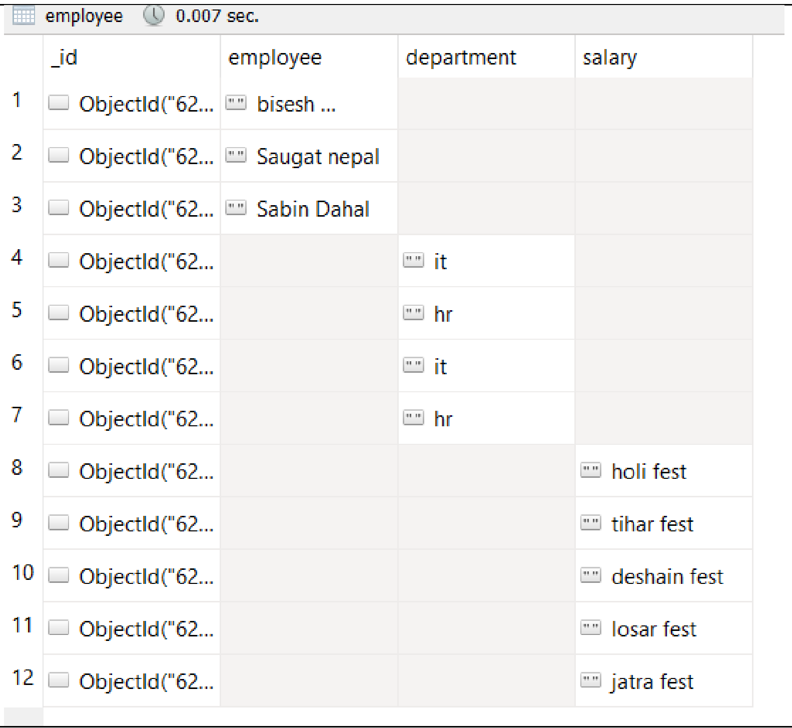
Script Output × | Query Result ×

SQL | All Rows Fetched: 2 in 0.007 seconds

| | EMPLOY... | EMPLOY... | EMPLOY... | EMPLOY... | EMPLOY... | EMPLOY... | DEDUCT... | DEPT_ID | DEP_NA... | ADDRESS | SALARY... | LABOUR... | OVER_T... | TOTAL | DETAIL |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | Bisesh ... | 1234567890 | Bhaktap... | bisesh@... | bisesh.... | (null) | 1 | it | bhaktapur | 1 | 8 | 2 | 10 | holi fest |
| 2 | 2 | Saugat ... | 2345678910 | kathman... | saugat@... | saugat.... | (null) | 2 | hr | kathmandu | 2 | 8 | 2 | 10 | tihar fest |



```
db.employee.aggregate([
 { $unionWith: "salary" },
 { $unionWith: "department" },
 {
 $group: {
        _id: "$_id",
        salary: { $push: "$detail" },
        department: { $push: "$dep_name" },
        employee: { $push: "$employee_name" }}},
 {$project: {
        _id: 1,
        employee: 1,
        department: 1,
        salary: 1,
    }
 },
 { $unwind: { path: "$employee",preserveNullAndEmptyArrays: true } },
 { $unwind: { path: "$department",preserveNullAndEmptyArrays: true } },
 { $unwind: { path: "$salary", preserveNullAndEmptyArrays:true } },
 { $sort: { _id: 1 } },]);
```



employee 0.007 sec.

| | _id | employee | department | salary |
|---|---|---|---|---|
| 1 | ObjectId("62... | bisesh ... | | |
| 2 | ObjectId("62... | Saugat nepal | | |
| 3 | ObjectId("62... | Sabin Dahal | | |
| 4 | ObjectId("62... | | it | |
| 5 | ObjectId("62... | | hr | |
| 6 | ObjectId("62... | | it | |
| 7 | ObjectId("62... | | hr | |
| 8 | ObjectId("62... | | | holi fest |
| 9 | ObjectId("62... | | | tihar fest |
| 10 | ObjectId("62... | | | deshain fest |
| 11 | ObjectId("62... | | | losar fest |
| 12 | ObjectId("62... | | | jatra fest |

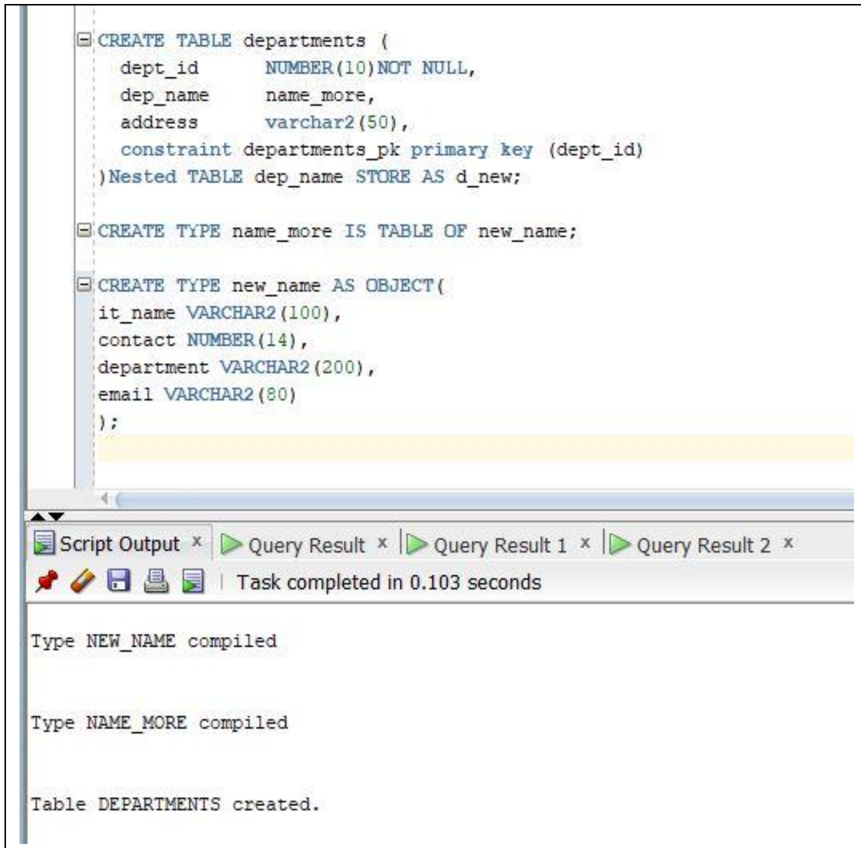
**Discussion:**
The major goal of the translated oracle questions into mongo DB queries is to deliver the same results regardless of the technique, not to perfectly duplicate the functionality of the oracle queries. Here, the findings of a research on the techniques for converting MySQL into MongoDB (Ha & Shichkina, 2021) may be used. According to Ha and Shichkina, the two essential criteria for translation are that the queries be constructed without relying on table structure and that the new query's results match those of the original query exactly. Multiple join types had to be examined for this query; I ultimately chose the outer join, which MongoDB does not support but which can be replicated using unions. In the mongo query, I union 3 collections to obtain the emails. In the SQL query, I outer join 3 tables to retrieve all the emails. The difficult part of this question was trying to implement various join types, but I was unable to come up with a query that called for it.

**Query b: A query which requires use of either a nested table or subtypes**

Inserting all data into tables

| **SQL code** | **MongoDB code** |
|---|---|

| | |
|---|---|
| CREATE TABLE departments (<br>  dept_id      NUMBER(10)NOT NULL,<br>  dep_name     name_more,<br>  address      varchar2(50),<br>  constraint departments_pk primary key (dept_id)<br>)Nested TABLE dep_name STORE AS d_new;<br><br>CREATE TYPE name_more IS TABLE OF new_name;<br><br>CREATE TYPE new_name AS OBJECT(<br>it_name VARCHAR2(100),<br>contact NUMBER(14),<br>department VARCHAR2(200),<br>email VARCHAR2(80)<br>); | db.parts.aggregate([ { $project: { _id: 1, gpu_core_clock: 1 } }, { $match: { gpu_core_clock: { $ne: null } } },]); |
| **Screenshots** | |



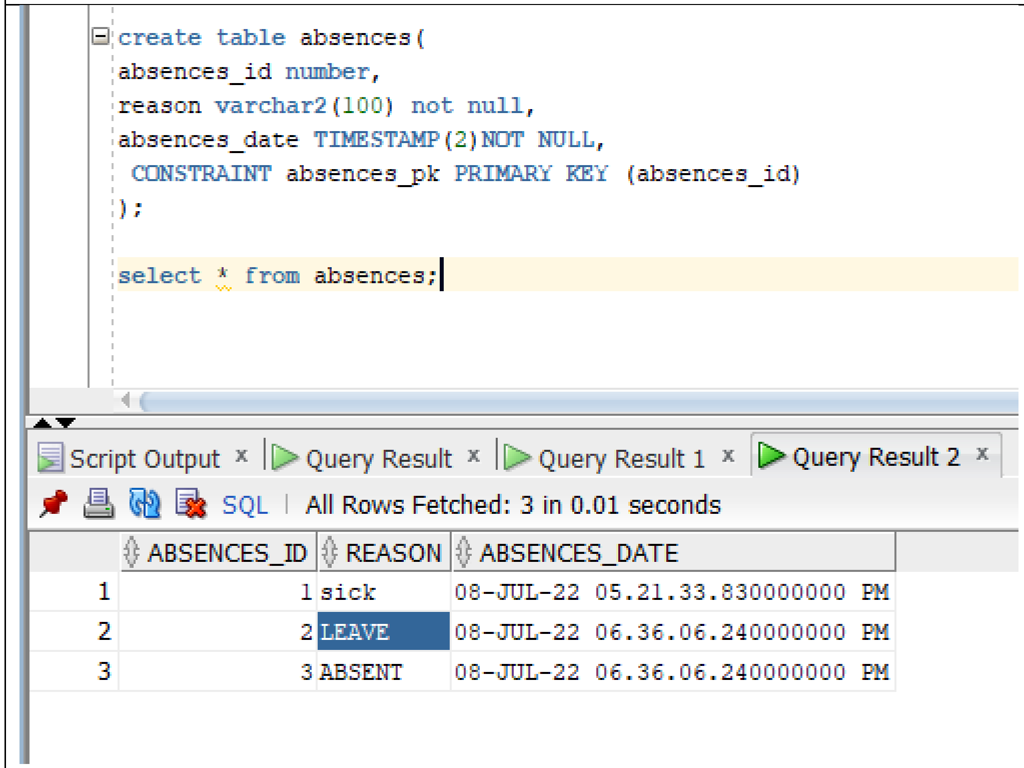
```
CREATE TABLE departments (
  dept_id      NUMBER(10)NOT NULL,
  dep_name     name_more,
  address      varchar2(50),
  constraint departments_pk primary key (dept_id)
)Nested TABLE dep_name STORE AS d_new;

CREATE TYPE name_more IS TABLE OF new_name;

CREATE TYPE new_name AS OBJECT(
it_name VARCHAR2(100),
contact NUMBER(14),
department VARCHAR2(200),
email VARCHAR2(80)
);
```

```
Task completed in 0.103 seconds

Type NEW_NAME compiled


Type NAME_MORE compiled


Table DEPARTMENTS created.
```

**Discussion:**

This second query was chosen because it was easy to extract the data from the subtypes and since all of the parts were subtypes in the parts table in the SQL code. The mongo technique was easier, in part because type inheritance was absent. I only needed to obtain the necessary info. Mongo once again used aggregation because it is the basis for all read operations. Because it is used in the next two questions as well, it is crucial to keep in mind that aggregation queries are substantially quicker than SQL read queries for the identical results for simple read operations like these (Gomes, 2021)

**Query c: A query using temporal features (e.g., timestamps, intervals, etc.) of Oracle SQL**

| *TIMESTAMPS* | |
|---|---|
| **SQL code** | **MongoDB code** |
| *create table absences(*<br>*absences_id number,*<br>*reason varchar2(100) not null,*<br>*absences_date TIMESTAMP(2)NOT NULL,*<br>*CONSTRAINT absences_pk PRIMARY KEY (absences_id)*<br>*);* | db..absence.aggregate([ { $addFields: { year: { $year: "$absence_date" } } }, { $match: { year: 2022 } }, { $group: { _id: { $month: "$absences_date" }, reason: { $sum: 1 } } }, ]); |
| **Screenshots** | |
|  | |
| **Discussion:** | |

```
create table absences(
absences_id number,
reason varchar2(100) not null,
absences_date TIMESTAMP(2)NOT NULL,
 CONSTRAINT absences_pk PRIMARY KEY (absences_id)
);

select * from absences;
```

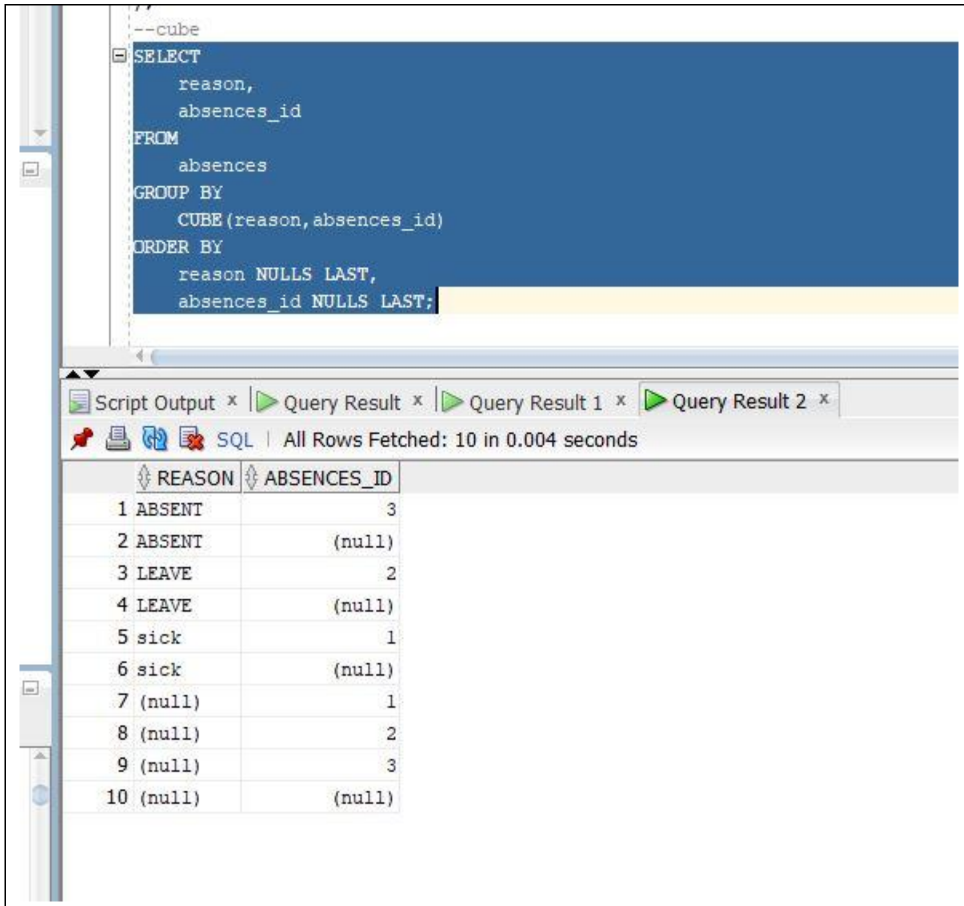
```
All Rows Fetched: 3 in 0.01 seconds

     ABSENCES_ID REASON   ABSENCES_DATE
1              1 sick     08-JUL-22 05.21.33.830000000 PM
2              2 LEAVE    08-JUL-22 06.36.06.240000000 PM
3              3 ABSENT   08-JUL-22 06.36.06.240000000 PM
```

Old database states must be kept because many database applications require accountability and traceability. For a transaction-time database to enable this, a committed transaction's serialization order must match the timestamps used to identify when database records are or were current (Massias, 1999). I focused on the timestamp for the transaction date of the bank user in my attempt to determine the time date of the transaction using the timestamp datatype.

**Query d: A query using OLAP (e.g., ROLLUP, CUBE, PARTITION) features of Oracle SQL**

*CUBE*

| **SQL code** | **MongoDB code** |
|---|---|
| *SELECT<br>   reason,<br>   absences_id<br>FROM<br>   absences<br>GROUP BY<br>   CUBE(reason,absences_id)<br>ORDER BY<br>   reason NULLS LAST,<br>   absences_id NULLS LAST;* | |
| **Screenshots** | |



```
--cube
SELECT
    reason,
    absences_id
FROM
    absences
GROUP BY
    CUBE(reason,absences_id)
ORDER BY
    reason NULLS LAST,
    absences_id NULLS LAST;
```

Script Output × | Query Result × | Query Result 1 × | Query Result 2 ×

SQL | All Rows Fetched: 10 in 0.004 seconds

|  | REASON | ABSENCES_ID |
|---|---|---|
| 1 | ABSENT | 3 |
| 2 | ABSENT | (null) |
| 3 | LEAVE | 2 |
| 4 | LEAVE | (null) |
| 5 | sick | 1 |
| 6 | sick | (null) |
| 7 | (null) | 1 |
| 8 | (null) | 2 |
| 9 | (null) | 3 |
| 10 | (null) | (null) |

**Discussion:**
An OLAP cube is a collection of data having 0 or more dimensions. The acronym "OLAP" stands for "online analytical processing". By examining company data, OLAP is a computer-based technique for gathering business insight. 2020 (Jensen, n.d.)
The CUBE extension adds a row with an account balance and a null value in the account number field to the above sql query. The outcome is identical to that of the ROLLUP function.

# References

Jensen, C. S., n.d. *https://www.researchgate.net/publication/324986147_Overview_of_OLAP_cubes_importance_build_Considerations_and_queryin.* [Online].