DATABASE HOMEWORK(60 pts)
31 May 2020

STUDENT: Halil Suheyb BECEREK - 295448

CONTACT: becerekh@student.mini.pw.edu.pl

"I certify that this assignment is entirely my own work, performed independently and without any help from the sources which are not allowed." - Halil Suheyb Becerek

[illegible]

Solution Description

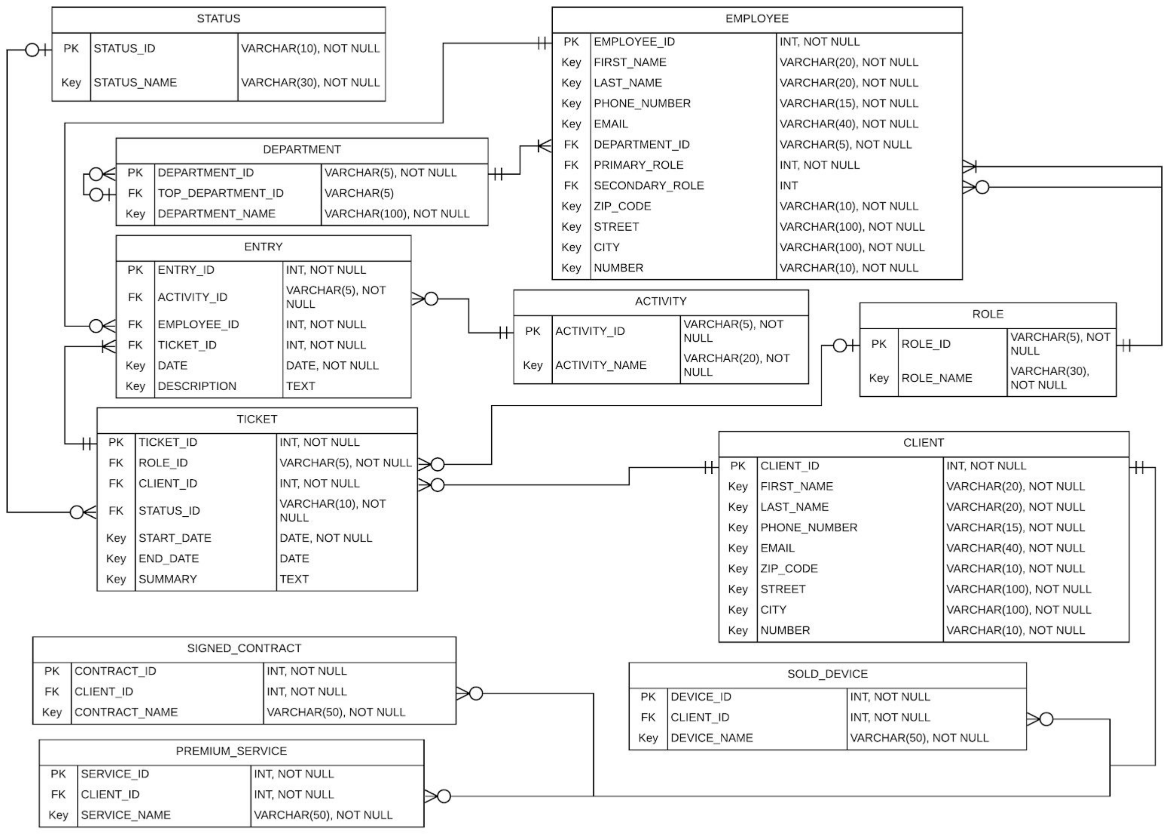
In this project we simulated an example database of a telecommunications company. Step Include:

- Designing Tables and ER diagram according to scenario
- Creating Tables, entering Mock Data for each table. Simulating a Resolve with few Updates on the rows of TICKET table
- Creating Indexes on certain tables and justification of Indexing
- Running required SQL queries on the mock data provided
- Creating required stored procedure *(Not Complete)*

Attachments to this file

- Queries.sql - file containing queries
- Homework.sql - file containing DDL operations
- proc.sql - file that has the stored procedure
- randomMNGR - a sub proc that allows to get a Random MNGR from department MNGNT

Design of Database



STATUS		
PK	STATUS_ID	VARCHAR(10), NOT NULL
Key	STATUS_NAME	VARCHAR(30), NOT NULL

EMPLOYEE		
PK	EMPLOYEE_ID	INT, NOT NULL
Key	FIRST_NAME	VARCHAR(20), NOT NULL
Key	LAST_NAME	VARCHAR(20), NOT NULL
Key	PHONE_NUMBER	VARCHAR(15), NOT NULL
Key	EMAIL	VARCHAR(40), NOT NULL
FK	DEPARTMENT_ID	VARCHAR(5), NOT NULL
FK	PRIMARY_ROLE	INT, NOT NULL
FK	SECONDARY_ROLE	INT
Key	ZIP_CODE	VARCHAR(10), NOT NULL
Key	STREET	VARCHAR(100), NOT NULL
Key	CITY	VARCHAR(100), NOT NULL
Key	NUMBER	VARCHAR(10), NOT NULL

DEPARTMENT		
PK	DEPARTMENT_ID	VARCHAR(5), NOT NULL
FK	TOP_DEPARTMENT_ID	VARCHAR(5)
Key	DEPARTMENT_NAME	VARCHAR(100), NOT NULL

ENTRY		
PK	ENTRY_ID	INT, NOT NULL
FK	ACTIVITY_ID	VARCHAR(5), NOT NULL
FK	EMPLOYEE_ID	INT, NOT NULL
FK	TICKET_ID	INT, NOT NULL
Key	DATE	DATE, NOT NULL
Key	DESCRIPTION	TEXT

ACTIVITY		
PK	ACTIVITY_ID	VARCHAR(5), NOT NULL
Key	ACTIVITY_NAME	VARCHAR(20), NOT NULL

ROLE		
PK	ROLE_ID	VARCHAR(5), NOT NULL
Key	ROLE_NAME	VARCHAR(30), NOT NULL

TICKET		
PK	TICKET_ID	INT, NOT NULL
FK	ROLE_ID	VARCHAR(5), NOT NULL
FK	CLIENT_ID	INT, NOT NULL
FK	STATUS_ID	VARCHAR(10), NOT NULL
Key	START_DATE	DATE, NOT NULL
Key	END_DATE	DATE
Key	SUMMARY	TEXT

CLIENT		
PK	CLIENT_ID	INT, NOT NULL
Key	FIRST_NAME	VARCHAR(20), NOT NULL
Key	LAST_NAME	VARCHAR(20), NOT NULL
Key	PHONE_NUMBER	VARCHAR(15), NOT NULL
Key	EMAIL	VARCHAR(40), NOT NULL
Key	ZIP_CODE	VARCHAR(10), NOT NULL
Key	STREET	VARCHAR(100), NOT NULL
Key	CITY	VARCHAR(100), NOT NULL
Key	NUMBER	VARCHAR(10), NOT NULL

SIGNED_CONTRACT		
PK	CONTRACT_ID	INT, NOT NULL
FK	CLIENT_ID	INT, NOT NULL
Key	CONTRACT_NAME	VARCHAR(50), NOT NULL

SOLD_DEVICE		
PK	DEVICE_ID	INT, NOT NULL
FK	CLIENT_ID	INT, NOT NULL
Key	DEVICE_NAME	VARCHAR(50), NOT NULL

PREMIUM_SERVICE		
PK	SERVICE_ID	INT, NOT NULL
FK	CLIENT_ID	INT, NOT NULL
Key	SERVICE_NAME	VARCHAR(50), NOT NULL

1.Each consultant is assigned to a company department, each department might be a part higher level company division up to the CEO board (multiple levels).

- Created a department table containing the ID of the department as PK and FK referring to the department one level above this department. We say that a top department may have many sub departments but a department only can have one department as a super department. FK key is *nullable* since there will be a root department a.k.a ceo board

2. The personal data for each employee include:first name, last name, address, phone number and email.

- Created the Employee table with given personal information from the first part **Each consultant is assigned to a company department** also added a foreign key with relationship *exactly one to one or many*

3. The company provides different types of services –online sales, advertising, maintenance, each employee has a specific primary and optionally secondary role assigned.

- Created a table ROLE with ID and Name. In the employee table created two FKs one is nullable other is not since secondary role is optional.

4. The company collects information about clients including their personal data and also all signed contracts, sold devices and active premium services.

- Created a client table and other tables that have a FK and referencing the Client table's PK. A signed contract has to have exactly one client but a client may have 0 or more contracts at the same time created the relationship in this logic this similar logic applies to all 3 tables.

5. Each contact with a customer is registered in a dedicated service ticket including type of service (contract, issue, advertising), start and end date and customer id.
   - Created a Ticket table with a FK references to Role table, Client table allowed the end date to be nullable
6. The ticket tracks progress of a particular case by recording each activity performed by the employee using a specific type of activity, date, and arbitrarily long description.
   - Created new table called entry with FK references to ticket table, employee table also created a new activity table which checks the activity of the entry
7. Each ticket has a status flag (e.g. registered, in progress) and for closed tickets the closure summary is added
   - Created table status to keep different types of status options also added a Summary nullable field to add summary when the status is updated to closed

Database Definition Language

While creating the tables first the tables with primary keys are created in the relationships. Tables that have INT fields as the primary key are designed as auto increment.
Order creating tables due to referencing was:
Department, Role, Client, Client data tables, Status, Ticket, Activity, Entry, Employee
Similar order was adapted while inserting the Mock data to the database.

Designing Indexes

```
CREATE NONCLUSTERED INDEX  employee_last_name_first_name_idx
ON EMPLOYEE(LAST_NAME , FIRST_NAME)
```

- Created a nonclustered index on the employee table since it might make it easier when checking a specific employee's performance based on lastname and first name. Index is non-clustered since the clustered index on this table is used by the PK

```
CREATE NONCLUSTERED INDEX client_last_name_first_name_idx
ON CLIENT (LAST_NAME, FIRST_NAME)
```

- Similar index created for the client table with again last name and first name might speed up the queries done when resolving a specific client's problem. Clustered index is used by Primary Key once again

```
CREATE NONCLUSTERED INDEX ticket_progress_register_idx
ON TICKET (STATUS_ID, START_DATE )
```

- We may want to run queries for HR or IA - internal audit - to check Start date of Tickets and their progress to fasten up such queries we created a nonclustered index based on status and start date.

```
CREATE NONCLUSTERED INDEX entry_employee_id_idx
ON ENTRY (EMPLOYEE_ID)
INCLUDE (ACTIVITY_ID, DATE, TICKET_ID)
```

- We might have many entries and we might want to run queries regarding entry history including date what activities are done on the ticket hence we created a nonclustered index on employee_id which will speed up queries based on employee_id on entry table

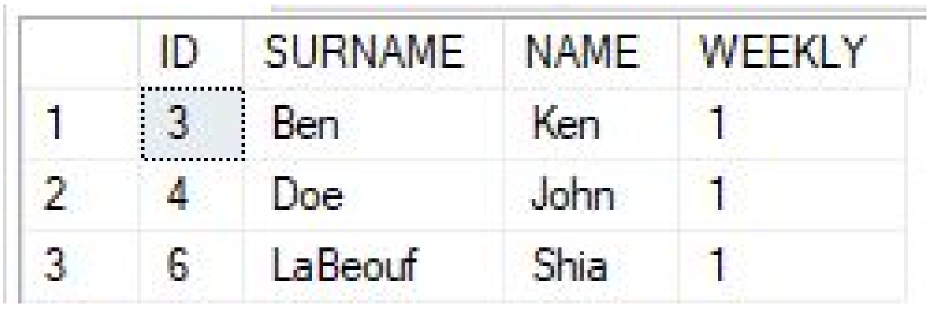
```
CREATE NONCLUSTERED INDEX contract_client_id_idx
ON SIGNED_CONTRACT(CLIENT_ID)

CREATE NONCLUSTERED INDEX service_client_id_idx
ON PREMIUM_SERVICE(CLIENT_ID)

CREATE NONCLUSTERED INDEX device_client_id_idx
ON SOLD_DEVICE(CLIENT_ID)
```

- Lastly we create indexes on client data tables that may speed up the queries based when we are looking for a specific client's data.

Queries

```
--Query #1 WORKS
SELECT EMPLOYEE.EMPLOYEE_ID AS ID,EMPLOYEE.LAST_NAME AS
SURNAME,EMPLOYEE.FIRST_NAME AS NAME, COUNT(*) AS WEEKLY FROM
EMPLOYEE
JOIN ENTRY ON EMPLOYEE.EMPLOYEE_ID = ENTRY.EMPLOYEE_ID
JOIN TICKET ON TICKET.TICKET_ID = ENTRY.TICKET_ID
WHERE TICKET.END_DATE >=  dateadd(wk, datediff(wk, 0, getdate()) -
1, 0)  --first day of last week
and TICKET.END_DATE < dateadd(wk, datediff(wk, 0, getdate()), 0)
--first day of this week
GROUP BY EMPLOYEE.EMPLOYEE_ID, EMPLOYEE.LAST_NAME,
EMPLOYEE.FIRST_NAME
```

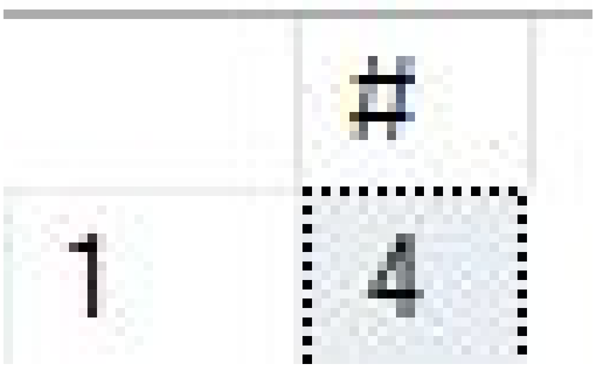
- Counting the tickets that are ended (which has an end date) between the first day of last week and the first day of this week.



	ID	SURNAME	NAME	WEEKLY
1	3	Ben	Ken	1
2	4	Doe	John	1
3	6	LaBeouf	Shia	1

```
--Query #2 WORKS
SELECT COUNT(*) AS # FROM
(SELECT TICKET.TICKET_ID,EMPLOYEE.LAST_NAME,COUNT(*) AS NUMBER
FROM ACTIVITY
INNER JOIN ENTRY ON ACTIVITY.ACTIVITY_ID = ENTRY.ACTIVITY_ID
INNER JOIN EMPLOYEE ON ENTRY.EMPLOYEE_ID = EMPLOYEE.EMPLOYEE_ID
INNER JOIN TICKET ON ENTRY.TICKET_ID = TICKET.TICKET_ID
--WHERE ENTRY.ACTIVITY_ID = 'RGSTR' --added this line since we are checking for tickets that an employe worked >=3 times but couldn't resolve
GROUP BY TICKET.TICKET_ID,EMPLOYEE.LAST_NAME
HAVING COUNT(ENTRY.ACTIVITY_ID) >= 3) c
```

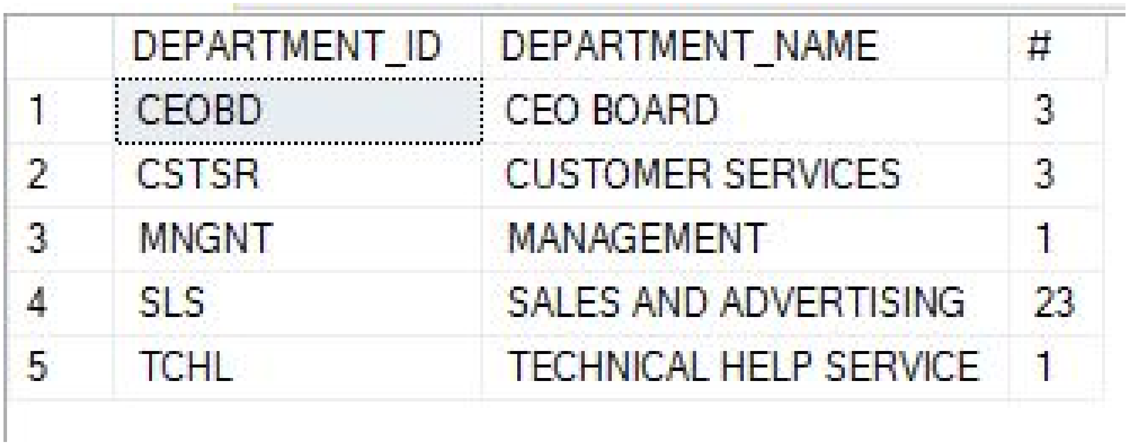
- Counting the number of rows that are grouped by Employee.last_name and Ticket.Ticket_id that has at least 3 entries with the same activity id and performed by the same employee



	#
1	4

```
SELECT DEPARTMENT.DEPARTMENT_ID,DEPARTMENT.DEPARTMENT_NAME ,
MAX(tmp.cnt) # FROM DEPARTMENT
JOIN
(
    SELECT TICKET.TICKET_ID, DEPARTMENT.DEPARTMENT_ID,
COUNT(ENTRY.ENTRY_ID) cnt FROM TICKET
    JOIN ENTRY ON ENTRY.TICKET_ID = TICKET.TICKET_ID
    JOIN EMPLOYEE ON EMPLOYEE.EMPLOYEE_ID = ENTRY.EMPLOYEE_ID
    JOIN DEPARTMENT ON DEPARTMENT.DEPARTMENT_ID =
EMPLOYEE.DEPARTMENT_ID
    WHERE EMPLOYEE.DEPARTMENT_ID = DEPARTMENT.DEPARTMENT_ID
    GROUP BY TICKET.TICKET_ID, DEPARTMENT.DEPARTMENT_ID)tmp ON
DEPARTMENT.DEPARTMENT_ID = tmp.DEPARTMENT_ID
    GROUP BY DEPARTMENT.DEPARTMENT_ID,DEPARTMENT.DEPARTMENT_NAME
```

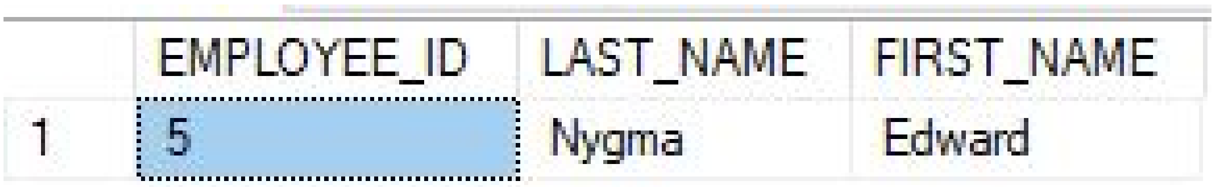
- First finding the entry count of each ticket and the department then finding the maximums of cnt column and grouping by department



	DEPARTMENT_ID	DEPARTMENT_NAME	#
1	CEOBD	CEO BOARD	3
2	CSTSR	CUSTOMER SERVICES	3
3	MNGNT	MANAGEMENT	1
4	SLS	SALES AND ADVERTISING	23
5	TCHL	TECHNICAL HELP SERVICE	1

```
--Query #4 WORKS ching
SELECT
    EMPLOYEE.EMPLOYEE_ID,
    EMPLOYEE.LAST_NAME,
    EMPLOYEE.FIRST_NAME
    FROM EMPLOYEE
JOIN ENTRY ON ENTRY.EMPLOYEE_ID = EMPLOYEE.EMPLOYEE_ID
GROUP BY EMPLOYEE.EMPLOYEE_ID,    EMPLOYEE.LAST_NAME,
    EMPLOYEE.FIRST_NAME
HAVING COUNT(ENTRY.ENTRY_ID) >=
(SELECT CAST(1.25 * COUNT(ENTRY.ACTIVITY_ID)/COUNT(DISTINCT
EMPLOYEE.EMPLOYEE_ID)AS float)
                                                              FROM
EMPLOYEE JOIN ENTRY ON EMPLOYEE.EMPLOYEE_ID = ENTRY.EMPLOYEE_ID
                                                              WHERE
ENTRY.ACTIVITY_ID = 'ADVTS' AND EMPLOYEE.DEPARTMENT_ID = 'SLS')
```

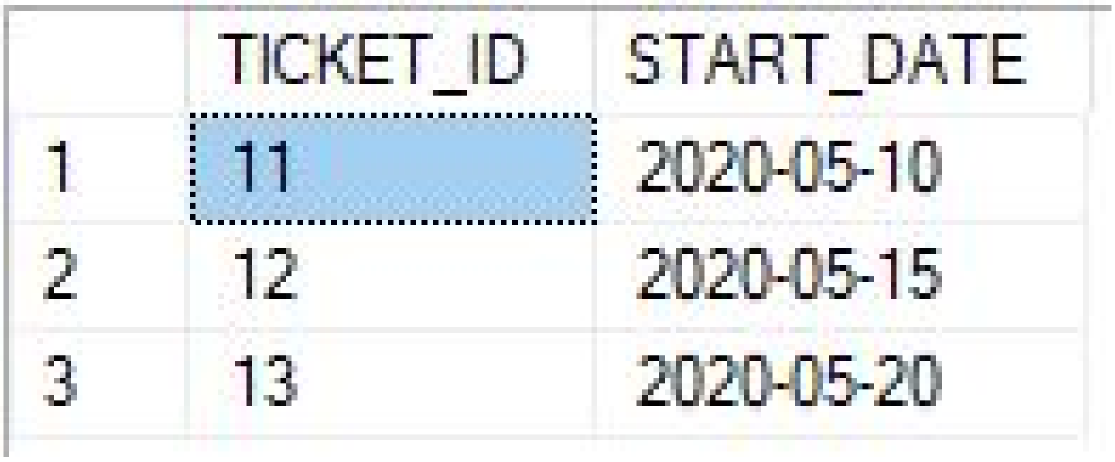
- There are two queries subquery is calculating the average for an employee on ‘SLS’ department since that is the department responsible for advertisement then selects the information of employee having entries greater than 125% of average.



	EMPLOYEE_ID	LAST_NAME	FIRST_NAME
1	5	Nygma	Edward

```
SELECT TOP 3
       *
FROM
( SELECT
    TICKET.TICKET_ID,
    TICKET.START_DATE
  FROM
    TICKET
JOIN ENTRY ON ENTRY.TICKET_ID = TICKET.TICKET_ID
WHERE TICKET.STATUS_ID != 'CLSD'
GROUP BY TICKET.TICKET_ID, TICKET.START_DATE
HAVING COUNT(ENTRY.ENTRY_ID) >= 2

)c ORDER BY c.START_DATE ASC
```

- Selecting the tickets that are not yet closed and having at least 2 entries then ordering them by start_date ascending and selecting top 3 of all to find the oldest 3.



	TICKET_ID	START_DATE
1	11	2020-05-10
2	12	2020-05-15
3	13	2020-05-20

Stored Procedure

Unfortunately stored procedures are not working. I am going to put here the output and error due to requirements.

```
Msg 547, Level 16, State 0, Procedure dbo.spTicket_Funktion, Line 18 [Batch
Start Line 53]
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK__ENTRY__EMPLOYEE___5441852A". The conflict occurred in database "TELECOM",
table "dbo.EMPLOYEE", column 'EMPLOYEE_ID'.
The statement has been terminated.
Msg 547, Level 16, State 0, Procedure dbo.spTicket_Funktion, Line 18 [Batch
Start Line 53]
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK__ENTRY__EMPLOYEE___5441852A". The conflict occurred in database "TELECOM",
table "dbo.EMPLOYEE", column 'EMPLOYEE_ID'.
The statement has been terminated.
Msg 547, Level 16, State 0, Procedure dbo.spTicket_Funktion, Line 18 [Batch
Start Line 53]
The INSERT statement conflicted with the FOREIGN KEY constraint
"FK__ENTRY__EMPLOYEE___5441852A". The conflict occurred in database "TELECOM",
table "dbo.EMPLOYEE", column 'EMPLOYEE_ID'.
The statement has been terminated.
Msg 16924, Level 16, State 1, Procedure dbo.spTicket_Funktion, Line 36 [Batch
Start Line 53]
Cursorfetch: The number of variables declared in the INTO list must match that
of selected columns.

Completion time: 2020-05-31T22:25:36.9010316+02:00
```