Middle East Technical University Department of Computer Engineering

# CENG 352 - Database Management Systems
# Written Assignment 1

Alper KOCAMAN
2169589

05.04.2020

## 1 E/R Diagram

**a)** Necessary SQL DDLs to resemble the E/R diagram are below:

```
CREATE TABLE Department (
dept_id int not null,
location varchar(100),
name varchar(100),
PRIMARY KEY (dept_id));
```

```
CREATE TABLE Employee (
emp_id int,
surname varchar(100),
name varchar(100),
salary real,
gender varchar(100),
PRIMARY KEY (emp_id));
```

```
CREATE TABLE Reports_to (
supervisor_emp_id int,
subordinate_emp_id int,
PRIMARY KEY (supervisor_emp_id, subordinate_emp_id),
FOREIGN KEY (supervisor_emp_id) REFERENCES Employee(emp_id),
FOREIGN KEY (subordinate_emp_id) REFERENCES Employee(emp_id));
```

```
CREATE TABLE Works_in (
dept_id int not null,
emp_id int,
PRIMARY KEY (dept_id, emp_id),
FOREIGN KEY (dept_id) REFERENCES Department(dept_id) ON DELETE NO ACTION,
FOREIGN KEY (emp_id) REFERENCES Employee(emp_id) ON DELETE CASCADE);
```

```
CREATE TABLE Manages (
dept_id int,
emp_id int not null DEFAULT 101,
PRIMARY KEY (dept_id),
FOREIGN KEY (dept_id) REFERENCES Department(dept_id),
FOREIGN KEY (emp_id) REFERENCES Employee(emp_id) ON DELETE SET DEFAULT);
```

```
CREATE TABLE Runs_Project (
dept_id int,
project_id int,
budget int,
due_date date,
state varchar(100),
PRIMARY KEY (dept_id, project_id),
FOREIGN KEY (dept_id) REFERENCES Department(dept_id));
```

**b)** An assertion called Total to enforce the total participation constraint:

```
CREATE ASSERTION Total CHECK
(NOT EXISTS (
SELECT Employee.emp_id
FROM Employee
WHERE Employee.emp_id not in
(select distinct Works_in.emp_id
from Works_in)
) )
```

**c)** SQL CHECK clauses to implement the given constraints:

```
CHECK (Employee.salary > 3600)
```

```
CHECK (Department.name LIKE CONCAT('%', Department.location, '%'))
```

**d)** An after trigger to implement the given constraint:

```
CREATE TRIGGER CheckBudgetOnChange
AFTER UPDATE OF budget ON Runs_Project
REFERENCING
OLD ROW AS OldRow
NEW ROW AS NewRow
FOR EACH ROW
WHEN (OldRow.budget > NewRow.budget)
UPDATE Runs_Project SET state = 'Unsuccessful'
WHERE Runs_Project.dept_id = OldRow.dept_id AND
Runs_Project.project_id = OldRow.project_id
```

## 2 Maximum Number of Tuples

**a)** Assume that the Product table consists of 100 rows with unique id's, the Store table has 5 rows with unique id's and the Person table contains 1000 rows with unique id's.

> We will need to have unique tuples with (product, store). Since we have 100 different products and 5 different stores, we will have 500 records at most.

**b)** Assume that the Product table consists of 100 rows with unique id's, the Store table has 5 rows with unique id's and among 1000 Person entries there are 10 SalesPerson.

> We need to have distinct tuples with (product, person). Since there are $1000 - 10 = 990$ person and 100 products, we have max 99000 records.

# 3 Armstrong's Axioms

**a)** CB → G

1) CB → B (Trivial Rule)
2) CB → B → E (Transitive Closure Rule)
3) CB → F (given as Functional Dependency)
4) By using 2 and 3, CB → FE (Splitting rule and Combing rule)
5) CB → FE → G (Transitive Closure Rule)
Hence, CB → G.

**b)** AB → EF

1) AB → A (Trivial Rule)
2) AB → B (Trivial Rule)
3) AB → A → C (Transitive Closure Rule)
4) AB → B → E (Transitive Closure Rule)
5) By using 2 and 3, AB → CB (Splitting rule and Combing rule)
6) By using 5, AB → CB → F (Transitive Closure Rule)
7) By using 4 and 6, AB → EF (Splitting rule and Combing rule)
Hence, AB → EF.

# 4 Normal Forms

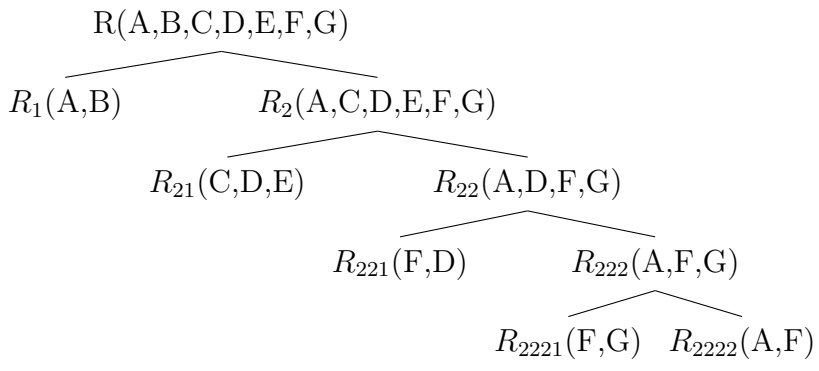
**a)** All keys are:

Firstly, I checked each column,
$A^+ = \{A, B\}$, $B^+ = \{B\}$, $C^+ = \{C\}$, $D^+ = \{C, D\}$, $E^+ = \{E, G\}$, $F^+ = \{D, F\}$, $G^+ = \{G\}$. Since none of the closure sets has all elements, I proceeded with two columns.
$AB^+ = \{A, B\}$, $AC^+ = \{A, B, C\}$, $AD^+ = \{A, B, C, D\}$, $AE^+ = \{A, B, E, G\}$, $AF^+ = \{A, B, C, D, E, F\}$, $AG^+ = \{A, B, G\}$.
Since $AF^+ = \{A, B, C, D, E, F\}$ and $AF^+$ contains all elements, AF is key.
If we check all two columns, we'll see that there are no candidate keys with 2 columns.

**b)** Is R in BCNF or not?

R is not in BCNF since there is an A $\rightarrow$ B dependency which is non-trivial. Then A should have been superkey, however it is not. In BCNF, all non-trivial dependencies should include super keys.

**c)** If it is not in BCNF form decompose it into a collection of BCNF relations.




**d)** Show that the above decomposition is,

dependency-preserving or not?

In this decomposition, $E \rightarrow G$ and $AC \rightarrow D$ functional dependencies are not preserved. Thus, this decomposition is not dependency-preserving.

lossless-join or not?

BCNF decomposition is always lossless.

# 5 Normalization

**a)** List of all FDs you identified and the corresponding SQL queries to discover them at the end of step 3 above.

$A \rightarrow E$

```
SELECT a
FROM givenTable
GROUP BY a
having count(distinct e) > 1
```

$AB \rightarrow C$

```
SELECT a
FROM givenTable
GROUP BY a, b
having count(distinct c) > 1
```

$AB \rightarrow E$

```
SELECT a
FROM givenTable
GROUP BY a, b
having count(distinct e) > 1
```

$AC \rightarrow B$

```
SELECT a
FROM givenTable
GROUP BY a, c
having count(distinct b) > 1
```

$AC \rightarrow E$

```
SELECT a
FROM givenTable
GROUP BY a, c
having count(distinct e) > 1
```

$AD \rightarrow E$

```
SELECT a
FROM givenTable
GROUP BY a, d
having count(distinct e) > 1
```

$BC \rightarrow A$

```
SELECT b
FROM givenTable
GROUP BY b, c
having count(distinct a) > 1
```

$BC \rightarrow E$

```
SELECT b
FROM givenTable
GROUP BY b, c
having count(distinct e) > 1
```

$BE \rightarrow A$

```
SELECT b
FROM givenTable
GROUP BY b, e
having count(distinct a) > 1
```

$BE \rightarrow C$

```
SELECT b
FROM givenTable
GROUP BY b, e
having count(distinct c) > 1
```

$C \rightarrow A$

```
SELECT c
FROM givenTable
GROUP BY c
```

```
having count(distinct a) > 1
```

$C \rightarrow B$

```
SELECT c
FROM givenTable
GROUP BY b
having count(distinct b) > 1
```

$C \rightarrow E$

```
SELECT c
FROM givenTable
GROUP BY e
having count(distinct e) > 1
```

$CD \rightarrow A$

```
SELECT c
FROM givenTable
GROUP BY c, d
having count(distinct a) > 1
```

$CD \rightarrow B$

```
SELECT c
FROM givenTable
GROUP BY c, d
having count(distinct b) > 1
```

$CD \rightarrow E$

```
SELECT c
FROM givenTable
GROUP BY c, d
having count(distinct e) > 1
```

$CE \rightarrow A$

```
SELECT c
FROM givenTable
GROUP BY c, e
having count(distinct a) > 1
```

$CD \rightarrow B$

```
SELECT c
FROM givenTable
GROUP BY c, d
having count(distinct b) > 1
```

$DE \rightarrow A$

```
SELECT d
FROM givenTable
GROUP BY d, e
having count(distinct a) > 1
```

$E \rightarrow A$

```
SELECT e
FROM givenTable
GROUP BY e
having count(distinct a) > 1
```

**b)** List of all SQL statements to create normalized tables.

```
CREATE TABLE table_AE (
A varchar(20),
E varchar(20),
PRIMARY KEY(A));
```

```
CREATE TABLE table_ABC (
A varchar(20),
B varchar(20),
C int,
```

```
PRIMARY KEY(C)
FOREIGN KEY(A) REFERENCES table_AE(A));
```

```
CREATE TABLE table_CD (
C int,
D int,
PRIMARY KEY(C,D));
```

**c)** List of all SQL statements that load the contents of the tables.

```
INSERT INTO table_AE (a,e)
SELECT DISTINCT a,e
FROM givenTable
```

```
INSERT INTO table_ABC (a,b,c)
SELECT DISTINCT a,b,c
FROM givenTable
```

```
INSERT INTO table_CD (c,d)
SELECT DISTINCT c,d
FROM givenTable
```