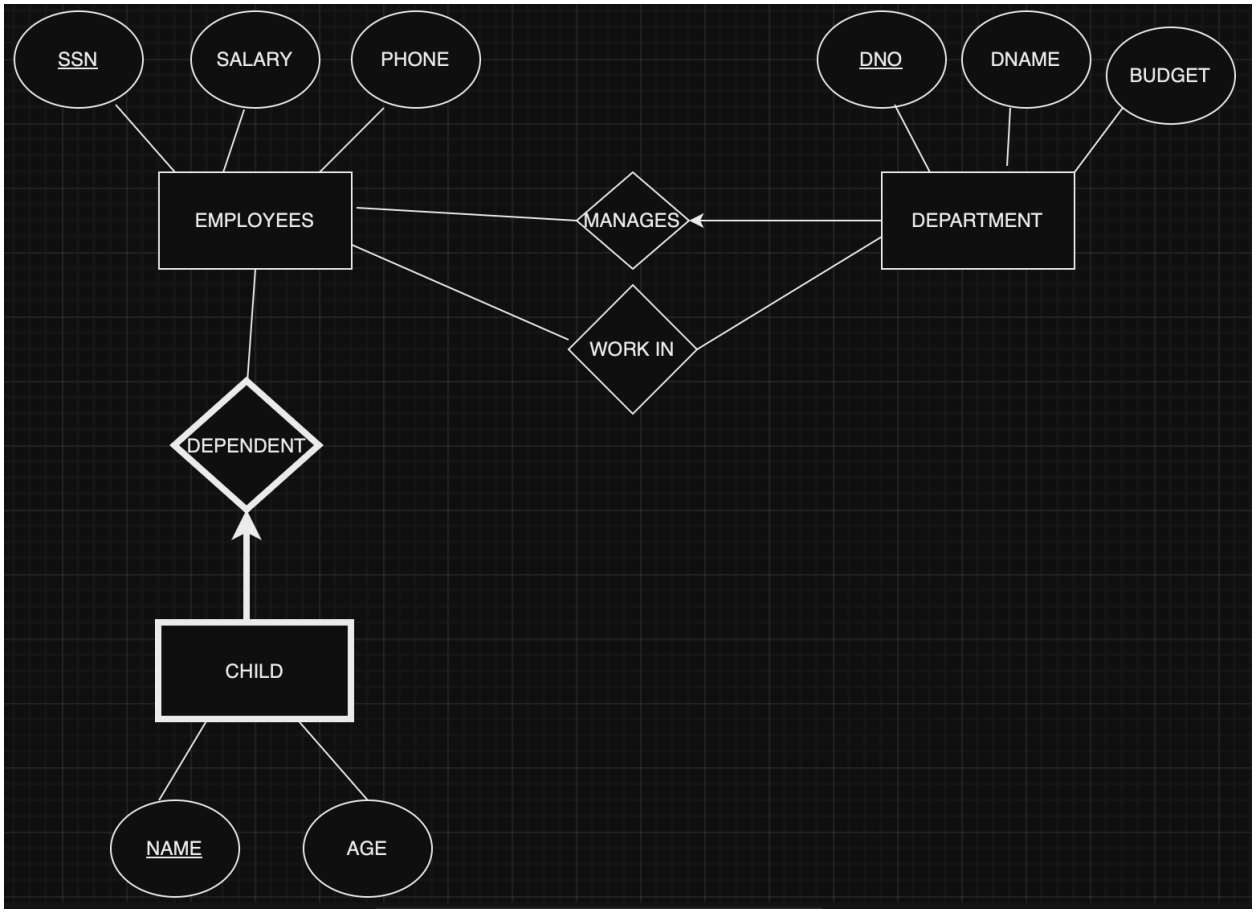
Homework 1

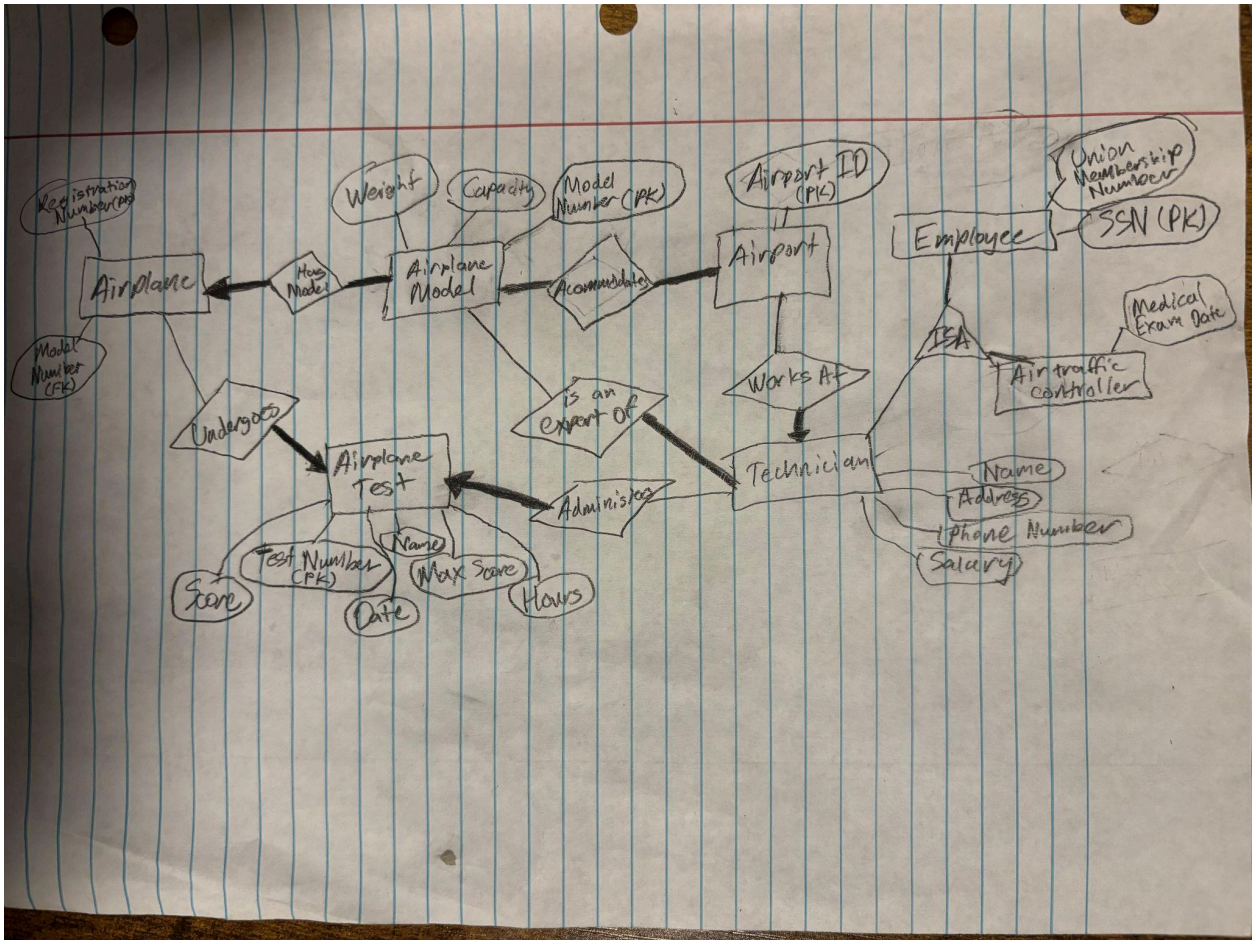
Alex Ojemann and Dehkontee Chea Cuppah

2.4:

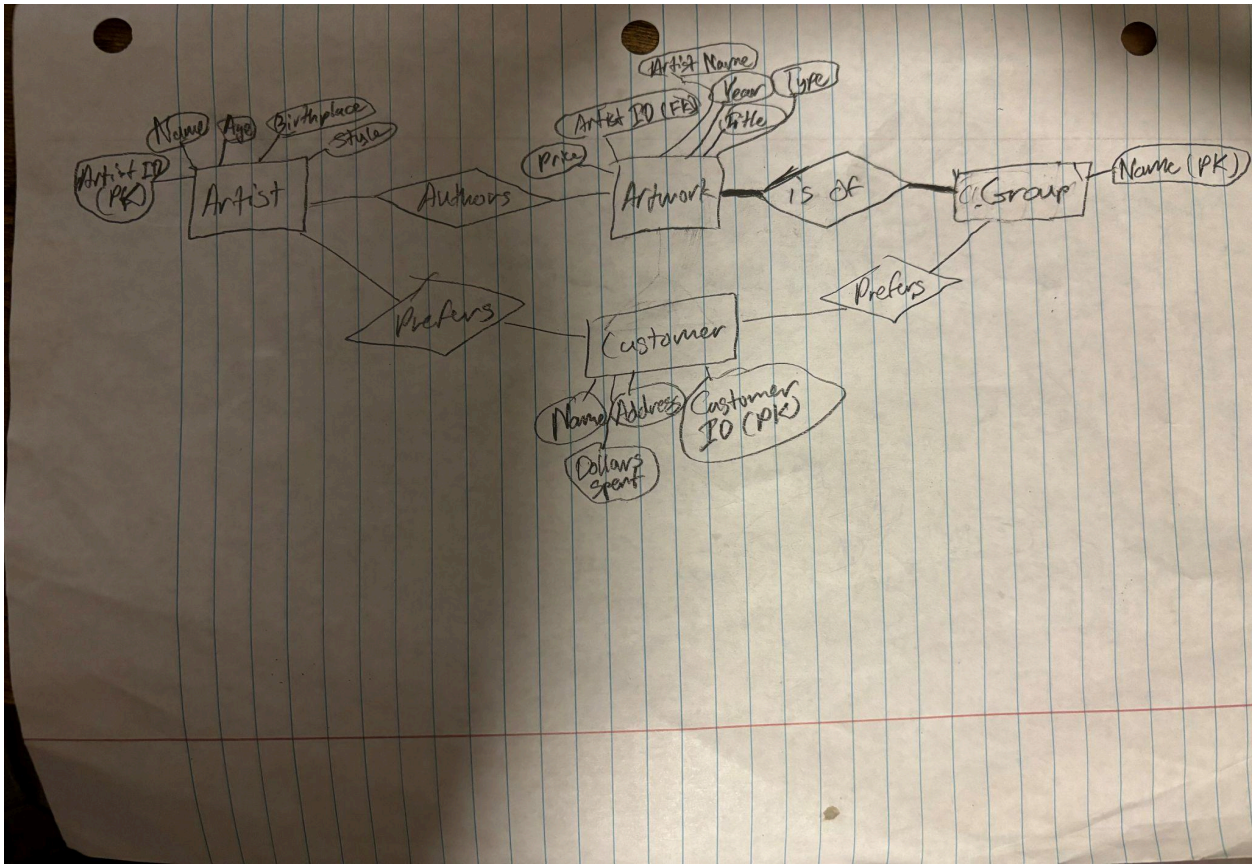


2.6:

1.


Registration Number (PK)
Weight
Capacity
Model Number (PK)
Airport ID (PK)
Union Membership Number
Employee
SSN (PK)
Airplane
Has Model
Airplane Model
Accommodates
Airport
Model Number (FK)
ISA
Medical Exam Date
Air traffic controller
Undergoes
Is an expert of
Works At
Airplane Test
Administers
Technician
Name
Address
Phone Number
Salary
Test Number (PK)
Name
Max Score
Score
Date
Hours

2.8:



3.14:

a. SQL command to create the Employees Table:

```
CREATE TABLE EMPLOYEES (
      ssn CHAR (10) PRIMARY KEY,
      salary INTEGER,
      phone CHAR (15),
  )
```

2. SQL command to create the Department Table:

```
CREATE TABLE DEPARTMENT (
        dno INTEGER PRIMARY KEY,
        dname CHAR (35),
        budget INTEGER,
     )
```

TO map one entity to another, we will create three different tables

3. SQL command to create the Works_in Table:

```
CREATE TABLE Works_in (
ssn CHAR (10),
dno INTEGER,
PRIMARY KEY (ssn, dno),
 FOREIGN KEY (ssn), “References EMPLOYEES table”
Foreign Key (dno) “References DEPARTMENT  table”
          )
```

4. SQL command to create the Manages Table:

```
CREATE TABLE Works_in (
ssn CHAR (10),
dno INTEGER,
PRIMARY KEY ( dno),
FOREIGN KEY (ssn), “References EMPLOYEES table”
Foreign Key (dno) “References DEPARTMENT  table”
          )
```

5. SQL command to create the Dependents Table:

```
CREATE TABLE Works_in (
ssn CHAR (10),
name CHAR(10),
age INTEGER,
PRIMARY KEY (ssn, name),
FOREIGN KEY (ssn)
          )
```

3.16:
Plane (reg_no: string)
Model (model_no: string, weight: integer, capacity: integer)
Test (FAA_no: string, name: string, score: integer)
Employee (ssn: string, union_mem_no: string)
Technician (ssn: string, name: string, address: string, phone_no: string, salary: integer)
Traffic Controller (ssn: string, exam_date: date)

Type (reg_no: string, model_no: string)
Expert (ssn: string, model_no: string)
Test_info(reg_no: string, FAA_no: string, ssn: string, hours: integer, score: integer, date: date)

```
CREATE TABLE Plane(
reg_no CHAR (10),
PRIMARY KEY(reg_no)
      )

CREATE TABLE Model(
model_no CHAR(10),
weight INTEGER,
capacity INTEGER,
PRIMARY KEY(model_no)
       )

CREATE TABLE Test(
FAA_no CHAR(10),
name CHAR(20),
score INTEGER
    )

CREATE TABLE Employee(
ssn CHAR(9),
union_mem_no CHAR(10),
PRIMARY KEY(ssn)
          )

CREATE TABLE Technician(
ssn CHAR(9),
name CHAR(20),
address CHAR(50),
phone_no CHAR(13),
salary INTEGER, PRIMARY KEY(ssn),
FOREIGN KEY(ssn) REFERENCES Employee(ssn)
```

```
  )

CREATE TABLE TRAFFIC CONTROLLER(
ssn CHAR(9),
exam_date DATE,
PRIMARY KEY(ssn),
FOREIGN KEY(ssn) REFERENCES Employee(ssn)
    )



CREATE TABLE Type (
reg_no CHAR(10),
model_no CHAR(10),
PRIMARY KEY(reg_no),
FOREIGN KEY(reg_no) REFERENCES Plane(reg_no),
FOREIGN KEY(model_no) REFERENCES Model(model_no)
                )

CREATE TABLE Expert (
ssn CHAR(9),
model_no CHAR(10),
PRIMARY KEY(ssn),
FOREIGN KEY(ssn) REFERENCES Plane(ssn),
FOREIGN KEY(model_no) REFERENCES Model(model_no)
                    )

CREATE TABLE Test_info(
FAA_no CHAR(10),
reg_no CHAR(10),
ssn CHAR(9),
hours INTEGER,
score INTEGER,
date DATE,
PRIMARY KEY(FAA_n),
FOREIGN KEY(ssn) REFERENCES Plane(ssn),
FOREIGN KEY(reg_no) REFERENCES Plane(reg_no)
          )
```

Part B

```
CREATE TABLE Plane(
 reg_no CHAR(10) ,
 model_no CHAR(10) ,
 PRIMARY KEY (reg_no),
 FOREIGN KEY (model_no) REFERENCES Model(model_no)
);


CREATE TABLE Model (
 model_num CHAR(10) ,
 capacity INTEGER,
 Weight INTEGER ,
 PRIMARY KEY (model_num)
);



CREATE TABLE model (
reg_no CHAR(10) ,
model_num CHAR(10) ,
PRIMARY KEY (reg_no,model_num),
FOREIGN KEY (reg_no) REFERENCES Plane(reg_no),
FOREIGN KEY (model_num) REFERENCES Model(model_num)
);



CREATE TABLE Employee (
 ssn CHAR(9) ,
 union_mem_no CHAR(20) ,
 PRIMARY KEY (ssn)
);



CREATE TABLE Technician (
 ssn CHAR(9) ,
 name CHAR(30),
 address CHAR(50),
```

```
 phone_noCHAR(10),
 Salary INTEGER,
 model_no CHAR(10) ,
 PRIMARY KEY (ssn),
 FOREIGN KEY (ssn) REFERENCES Employee(ssn),
 FOREIGN KEY (model_no) REFERENCES Model(model_no)
);



CREATE TABLE Traffic_Controller (
 ssn CHAR(9) ,
 exam _date DATE,
 PRIMARY KEY (ssn),
 FOREIGN KEY (ssn) REFERENCES Employee(ssn)
);



CREATE TABLE Expert (
 ssn CHAR(9) ,
 model_no CHAR(20) ,
 PRIMARY KEY (ssn, model_num),
 FOREIGN KEY (ssn) REFERENCES Employee(ssn),
 FOREIGN KEY (model_no) REFERENCES Model(model_no)
);



CREATE TABLE Test (
FAA_no CHAR(20) ,
Score INTEGER,
name CHAR(30),
PRIMARY KEY (FAA_no),
);


CREATE TABLE Test_info(
 date DATE,
 Hours INTEGER,
 scoreDECIMAL(10,2),
 reg_no CHAR(10) ,
 FAA_no CHAR(20) ,
 Ssn CHAR(9) ,
```

```
 PRIMARY KEY (reg_no, FAA_no, ssn),
 FOREIGN KEY (reg_no) REFERENCES Plane(reg_no),
 FOREIGN KEY (FAA_no) REFERENCES Test(FAA_no),
 FOREIGN KEY (ssn) REFERENCES Technician(ssn)
);
```

Altering Testing_Event table to add the given constraint:

```
ALTER TABLE Testing_Event
ADD CONSTRAINT FAA_Regulation CHECK (
 EXISTS (
   SELECT 1
   FROM Technicians
   WHERE Technician.ssn = Testing_Event.ssn
   AND Technician.model_num LIKE '%TestingEvent.model_num%'
 )
);
```

4.4:

1. $\pi_{sname}(\pi_{sid}(\sigma_{color='red'} Parts)$!XI $(O'cost<lOoCatalog)$!XI $Suppliers)$

It computes the ID of suppliers with Catalog that cost less than 100 and its parts are red.

3. $(\pi_{sname}((O'color'='red' Parts) \bowtie (crcost<looCatalog)$!XI $Suppl'iers))$ n

$$(\pi_{sname}((\sigma_{color='green'} Parts) \bowtie (\sigma_{cost<100} Catalog) \bowtie Suppliers))$$

This returns a name of suppliers with Catalog that cost less than 100 and its parts are red and green.

5. $\pi_{sname}((\pi_{sid,sname}((\sigma_{color='red'} Parts) \bowtie (\sigma_{cost<100} Catalog) \bowtie Suppliers))$ ∩

$$(\pi_{sid,sname}((\sigma CO!07'='green' Parts)$$!XI $(\sigma cost< lOoCatalog) \bowtie Suppliers)))$$

It computes the name and ID of suppliers with Catalog that cost less than 100 and its parts are red and green.

5.4:

2.
```
SELECT d.did AS department_id, SUM(CASE WHEN w.pct_time = 100 THEN 1 ELSE
w.pct_time / 100 END) AS full_time_equivalent
FROM Dept d
JOIN Works w ON d.did = w.did
GROUP BY d.did
HAVING SUM(CASE WHEN w.pct_time = 100 THEN 1 ELSE w.pct_time / 100 END) > 20;
```

3.
```
SELECT ename
FROM Emp e
WHERE e.Salary > ALL (SELECT d.budget FROM dept d JOIN Works w ON d.did= w.did AND
e.eid=w.eid);
```

5.
```
SELECT e.ename
FROM Emp e
JOIN Dept d ON e.eid = d.managerid
WHERE d.budget = (SELECT MAX(d2.budget) FROM Dept d2);
```

6.
```
SELECT d.managerid
FROM Dept d
GROUP BY d.managerid
HAVING SUM(d.budget) > 5000000;
```

5.8:

1.
```
CREATE TABLE Student (
        snum INT NOT NULL,
        sname VARCHAR(255),
        major VARCHAR(255),
        level VARCHAR(255),
        age INT,
        PRIMARY KEY (snum)
);
CREATE TABLE Faculty (
        fid int NOT NULL,
        fname VARCHAR(255),
```

```
        deptid INT
);

CREATE TABLE Class(
        name VARCHAR(255) NOT NULL,
        meets.at TIME,
        room VARCHAR(255),
        fid INT,
        PRIMARY KEY (narne),
        FOREIGN KEY (fid) REFERENCES Faculty(fid)
);

CREATE TABLE Enrolled(
        snum INT,
        cname VARCHAR(255),
        PRIMARY KEY (snum,cname)
        FOREIGN KEY (snum) REFERENCES Student(snum)
        FOREIGN KEY (cname) REFERENCES Class(name)
);
```

2.

a. This can't be expressed as an SQL constraint because it deals with multiple tables (Enrolled and Class). To enforce this, insertions into the class table must be monitored so that no class can be added if there aren't enough students enrolled or are too many students enrolled. Also, both insertions into and deletions from the enrolled table must be monitored so that the enrollment of a class does not exceed 30 or drop below 5.

c. This can't be expressed as an SQL constraint because it deals with multiple tables (Faculty and Class). To enforce this, insertions into the faculty table must be monitored to prevent the insertion of a faculty member who does not teach at least two classes, and deletions from the class table must be monitored to prevent a faculty member who only has two classes from dropping below the constraint if one of their classes is deleted.

e. This can't be expressed as an SQL constraint because it deals with multiple tables (Student and Enrolled). To enforce this, insertions into the student table must be monitored so that no student can be inserted who isn't taking the Math 101 course. Deletions from the enrolled table must be monitored so that a student's enrollment in Math 101 can't be removed. Updates to this table must also be monitored so that a student's enrollment in Math 101 can't be changed to enrollment in a different class name. Deletions and updates from the class table must also be monitored so that the Math 101 class can't be deleted or renamed.

g. This constraint can be expressed as: "ADD CONSTRAINT no_same_time_same_room UNIQUE(meets.at,room)" and added to the class table. This will monitor insertions into the class

table to avoid a class being inserted into the table that has the same meeting time and room as an existing class. It will also monitor updates to the class table to prevent anyone from updating the meeting time and/or the room of a course to be the same as another course in the table.

l. This can't be expressed as an SQL constraint because it deals with multiple tables (Student, Class, and Enrolled). To enforce this, insertions into and deletions from the enrolled table must be monitored so that no students majoring in math can add a course distinct from all of the other courses taken by math majors if that addition would make the number of distinct courses taken by CS majors less than or equal to that of math majors and no students majoring in CS can drop a course distinct from all of the other courses taken by CS majors if that deletion would make the number of distinct courses taken by CS majors less than or equal to that of math majors. Insertions and deletions from the student table must also be monitored so that no students majoring in math can be added if they are taking a course distinct from all of the other courses taken by math majors if that addition would make the number of distinct courses taken by CS majors less than or equal to that of math majors and no students majoring in CS can be deleted if they are taking a course distinct from all of the other courses taken by CS majors if that deletion would make the number of distinct courses taken by CS majors less than or equal to that of math majors. In addition, no courses can be added or deleted from the class table if they are taken by CS majors but no math majors or math majors but no CS majors and the addition or deletion of the course results in the number of distinct courses taken by CS majors becoming less than or equal to that of math majors.

m. This can't be expressed as an SQL constraint because it deals with multiple tables (Faculty, Student, Class, and Enrolled). To enforce this, deletions from the enrolled table must be monitored so that no enrollment can be removed that would make the number of enrollments in courses taught by faculty with deptid = 33 less than or equal to the number of math majors, insertions into the student table must be monitored so that a math major can't be added that would make the number of enrollments in courses taught by faculty with deptid = 33 less than or equal to the number of math majors, deletions from the faculty table would need to be monitored so that a faculty member with deptid = 33 can't be removed they taught courses which if removed would make the number of enrollments in courses taught by faculty with deptid = 33 less than or equal to the number of math majors, and deletions from the class table so that no course could be removed if that course was taught by a faculty member with deptid = 33 such that its removal would make the number of enrollments in courses taught by faculty with deptid = 33 less than or equal to the number of math majors.

8.6:

File Type	**Scan**	**Equality Search**	**Range Search**	**Insert**	**Delete**
Heap file	**BD**	**0.5BD**	**BD**	**2D**	**Search + D**

Sorted file	**BD**	**Dlog2b**	**Dlog2B+# Search+ Search+ matching pages**	**Search + BD**	**Search + BD**
Clustered file	**1.5BD**	**DlogF 1.5B**	**DlogF1.5 B + #matches**	**Search + D**	**Search + D**
Unclustered tree index	**BD(R+0.15)**	**D(1+LogF0.15B)**	**D(1+ logF0.15B + #matches)**	**D(logF0.1 5B)**	**Search + 2D**
Unclustered hash index	**BD(R+0.125)**	**2D**	**BD**	**4D**	**Search + 2D**

8.10:

1 - To enforce eid as a key using the indexes, I would create a unique constraint or primary key on eid. This would prevent duplicate eid values from being inserted. The clustered index on eid supports efficient enforcement of uniqueness.

2 - An update that is speeded up because of the available indexes will be an update to all employees ages. This is because the index on attribute age is unclustered which makes it easy to find specific values for it. The unclustered index can be used to locate the given value faster and the update can be done on the specific records thus speeding up the process.

3 - An update that is slowed down because of the indexes is updating the eid. This process is slowed down because updating the eid basically change the value and disrupt the order of the clustered indexes. Considering eid is a clustered index, the best way to successfully update it without compromising the database integrity and uniqueness is to do that in a new location which is time consuming.

4 - An update that is neither sped up or slowed down would be an update to the salary of employees. Since attribute sal doesn’t have any indexes on it, the update can be done directly to it without any indexing which makes its performance neither speeded up or slowed.