**CSE 216**

DATABASE MANAGEMENT SYSTEM

# TERM PROJECT



Team Members : Mursheda Tasnim

Umama Akhtar Turna

Student ID’s : 1605092

1605095

Supervisor : Md. Iftekharul Islam Sakib

Level/Term : 2/2

Section : B

**INTRO**

Magoosh is an online test preparation company based in Berkeley, California that teaches students through video lessons, practice test questions, and online email support. It was founded in 2009 and it’s Headquarter is in Berkeley, California, United States. The company is founded by Bhavin Parikh, Vikram Shenoy, Pejman Pour-Moezzi and Hansoo Lee.

Here we have created a Database Management System which acts like the Database Management System of Magoosh. Then we have created a GUI which shows most of the activities of our Database Management System.

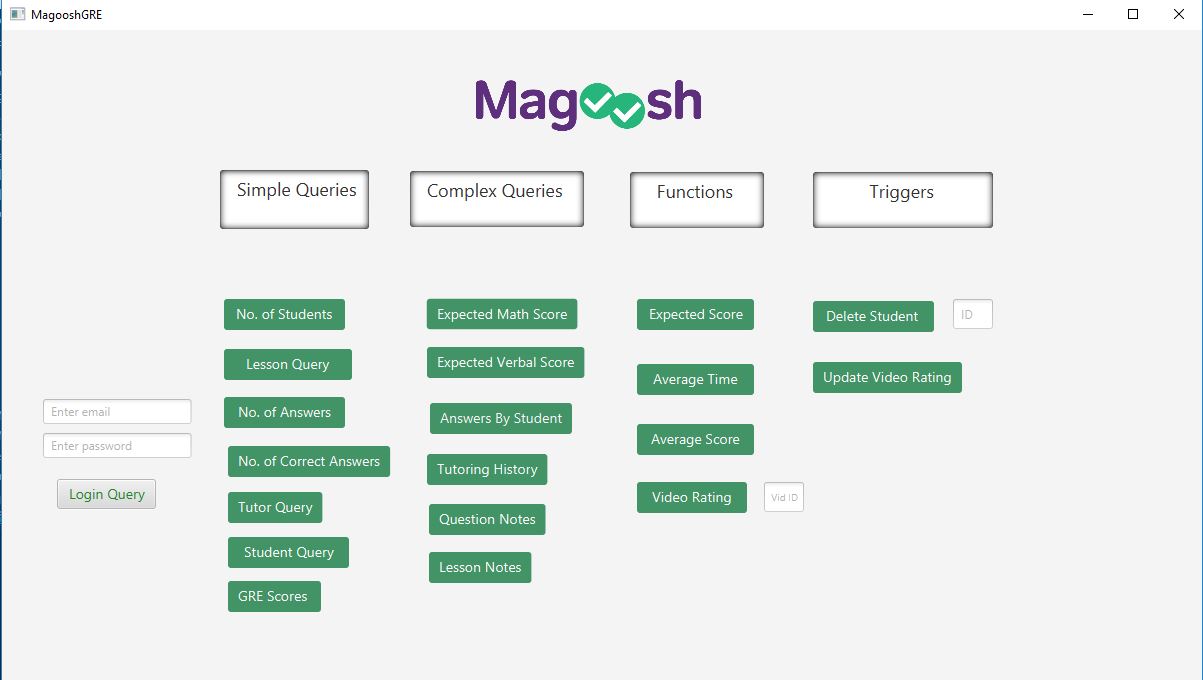
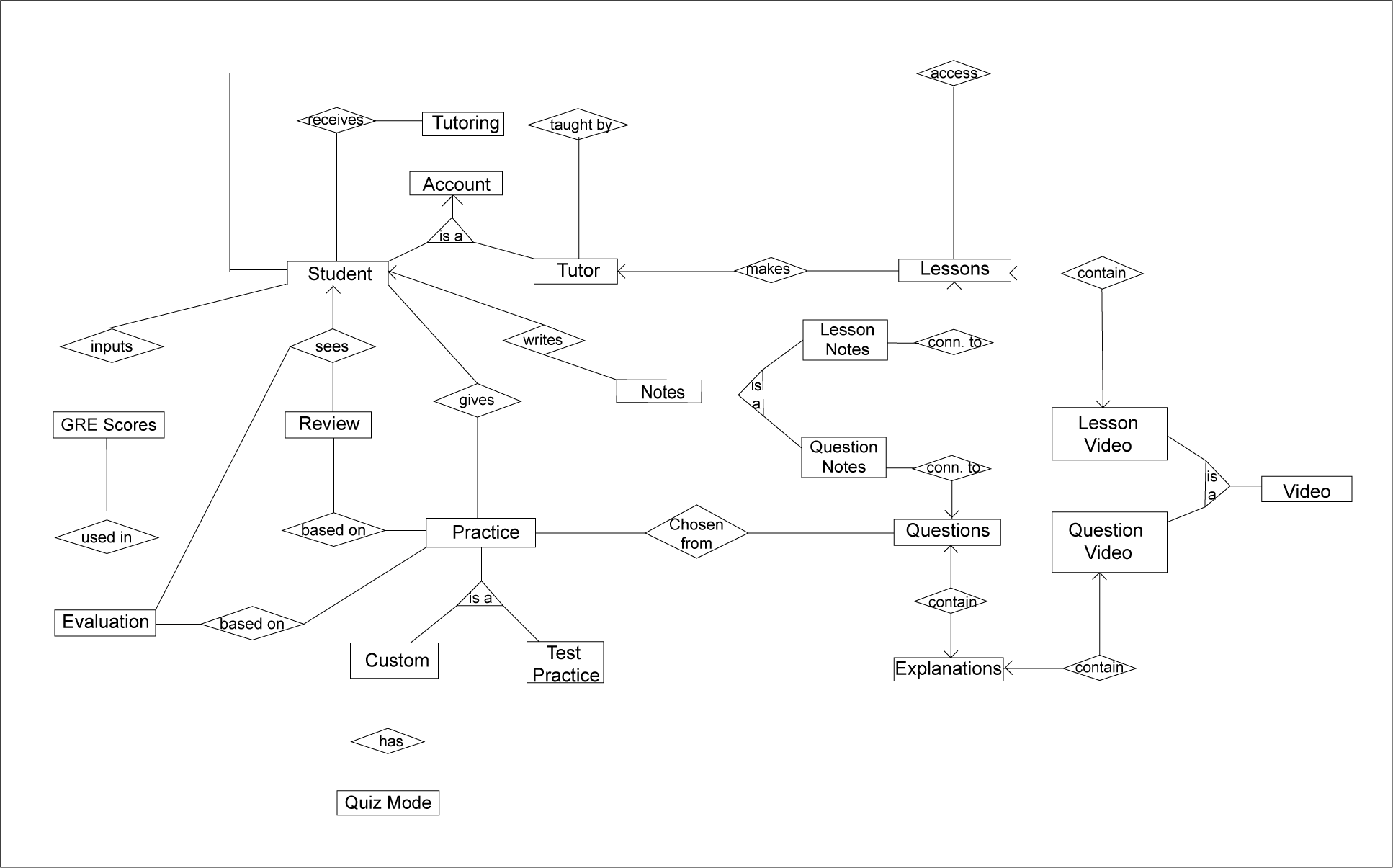


Fig: Front End of Magoosh DBMS

**ERD**



ENTITIES (15):

1. Student

2. Tutor

3. Tutoring

4. Questions

5. Question Videos

6. Question Notes

7. Lessons

8. Lesson Videos

9. Lesson Notes

10. Video Feedback

11. Evaluation

12. GRE Scores

13. Subjects

14. Test Practice

15. Quiz

SIMPLE QUERIES (8):

1. **Student Log In**  
   – Matches email, password from input with “Student” table
2. **No. Of Students**  
   - Counts number of rows in Student Table.  
   This is needed in many other functions, like calculating average time or score.
3. **Lesson Query**– Shows all lessons and their subjects in a Table  
   This is needed when the student wants to choose lessons on a subject or a specific lesson.
4. **No. Of Answers**  
    – Total questions answered by logged in student.  
   This is needed in the Review part of the Magoosh app. Students can see how many questions they answered and how they did.
5. **No. Of Correct answers**   
   – No. of correct answers by logged in student.  
   This is also an important part in the review section of Magoosh.
6. **Tutor information**   
   – All information of signed up tutors in a Table.
7. **Student information**   
   – All information of signed up students in a Table
8. **GRE scores**   
   – Past GRE scores of all students that used Magoosh.  
   This is used to estimate the projected GRE score of a student by comparing his score range with those of the past students and using the trend to estimate his score.

COMPLEX QUERIES (6):

1. **Expected Math Score**   
   – uses student’s correct answers, average score & past students’ GRE scores on Quantitative part.  
   This is used in Dashboard of the Magoosh app.
2. **Expected Verbal Score**   
   – uses student’s correct answers, average score & past students’ GRE scores on Verbal part.  
   This is used in Dashboard of the Magoosh app.
3. **Answer Table** – shows student’s score & time on each question along with question titles and subjects.  
   These information are needed for review section of the app.
4. **Tutoring History of Student**– shows name of tutor and tutoring details of logged in student
5. **Question Notes** – shows notes taken by student on questions along with the question title
6. **Lesson Notes**   
   – shows notes taken by student on lessons along with the lesson title

FUNCTIONS (9):

1. **Expected Score**   
   – Projected total GRE score of student, by matching his average math/verbal score range with scores of past students and their subsequent GRE score
2. **Average Time**   
   – Average time to answer questions (math/verbal)
3. **Average Score**   
   – Average score on all answered questions (math/verbal)
4. **Video Rating**   
   –Rating of people who clicked on Like on a video
5. **Video like percenage**   
   – Percentage of people who clicked on Like on a video
6. **Expected Math Score**  
   – Projected total GRE score of student, by matching his average math/verbal score range with scores of past students and their subsequent GRE score
7. **Expected Verbal Score**  
   – Projected total GRE score of student, by matching his average math/verbal score range with scores of past students and their subsequent GRE score
8. **Average Math Score**  
   – Average score of student only in math part
9. **Average Verbal Score**  
   – Average score of student only in verbal part

TRIGGERS (4):

1. Delete a Student’s Tutoring history if the Student is deleted from Student table
2. Delete Video if Like percentage drops below 20% when feedback table is Update
3. Delete Feedback from feedback table if Video is deleted from Question Video or Lesson Video table *(Not shown in GUI)*
4. Delete a Student’s feedbacks if the Student is deleted from Student table *(Not shown in GUI)*

Here are some samples from our code.

### SIMPLE QUERIES

SELECT password FROM Public.student WHERE email=mail;

select count(\*) from videos where vid\_id=vid;

select sum(likes) from videos where vid\_id=vid;

select count(\*) from public.answered where std\_id=std;

select avg(time\_sec) from answered where std\_id=std;

select sum(correct) from answered where std\_id=std;

select avg(time\_sec) from answered where q\_id=q;

### COMPLEX QUERIES

SELECT avg(grescore\_math) FROM grescores WHERE math\_expected >= ((SELECT avg(scores) FROM answered WHERE std\_id=std) - 5) and math\_expected <= ((SELECT avg(scores) FROM answered WHERE std\_id=std) + 5) ;

select count(\*) into crr from public.answered as a, public.stident as b where a.std\_id=b.”ID” std\_id=std and correct=1;

SELECT sum(grescore) FROM grescores WHERE verbal\_expected >= ((SELECT avg(scores) FROM answered WHERE std\_id=std) - 5) and verbal\_expected <= ((SELECT avg(scores) FROM answered WHERE std\_id=std) + 5) ;

select title from public.answered, public.subjects, public.student where std\_id=std and subj\_name=ans\_sub and std\_id = student.”ID”;

select sum(likes) from videos inner join public.subjects where vid\_id=vid and videos.subject = subjects.subj\_name;

**FUNCTIONS**

**1. AVERAGE SCORE**

CREATE OR REPLACE FUNCTION public.avg\_score(

std integer,

sub integer)

RETURNS real

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$ DECLARE

tm real;

n real;

av real;

BEGIN

if (sub=0)

then

select howmanyans(std) into n;

select sum(correct) into tm from answered where std\_id=std;

end if;

if (sub=1)

then

select count(\*) into n from answered where std\_id=std and subj='m';

select sum(correct) into tm from answered where std\_id=std and subj='m';

end if;

if (sub=2)

then

select count(\*) into n from answered where std\_id=std and subj='v';

select sum(correct) into tm from answered where std\_id=std and subj='v';

end if;

select (tm\*170/n) into av;

RETURN av;

END;$BODY$;

ALTER FUNCTION public.avg\_score(integer, integer)

OWNER TO postgres;

**2. AVERAGE TIME**

CREATE OR REPLACE FUNCTION public.avg\_time(

std integer)

RETURNS numeric

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$ DECLARE

tm INTEGER;

n INTEGER;

mint integer;

sec integer;

str character(20);

BEGIN

select howmanyans(std) into n;

select avg(time\_sec) into tm from answered where std\_id=std;

mint = tm/60;

sec = tm%60;

select (mint || ' minutes ' || sec || ' seconds') into str;

RETURN tm;

END;$BODY$;

ALTER FUNCTION public.avg\_time(integer)

OWNER TO postgres;

**3. VIDEO RATING**

CREATE OR REPLACE FUNCTION public.likepercent(

vid integer)

RETURNS character

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$ DECLARE

perc real;

n real;

summ real;

str character(10);

BEGIN

select count(\*) into n from videos where vid\_id=vid;

select sum(likes) into summ from videos where vid\_id=vid;

select (summ\*100/n) into perc;

select (perc||'%') into str;

RETURN str;

END;$BODY$;

ALTER FUNCTION public.likepercent(integer)

OWNER TO postgres;

**5. EXPECTED SCORE**

CREATE OR REPLACE FUNCTION public.expectedscore(

std integer)

RETURNS integer

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$DECLARE

exp\_ integer;

n integer;

ownmath real;

y real;

BEGIN

select avg\_score(std,1)\*100/170 into ownmath;

select avg(grescore\_math) from grescores where math\_expected>=(ownmath-5) and math\_expected<=(ownmath+5) into exp\_;

RETURN exp\_;

END;

$BODY$;

ALTER FUNCTION public.expectedscore\_math(integer)

OWNER TO postgres;

**6. LOGIN**

CREATE OR REPLACE FUNCTION public.login(

mail text,

pass text)

RETURNS integer

LANGUAGE 'plpgsql'

COST 100

VOLATILE

AS $BODY$DECLARE

checkp integer;

passd character varying;

BEGIN

SELECT password into passd

FROM Public.student

where email=mail;

checkp=0;

if passd = pass then

checkp=1;

end if;

RETURN checkp;

END;

$BODY$;

ALTER FUNCTION public.login(text, text)

OWNER TO postgres;

## TRIGGERS

**1. Delete Student -> Delete Tutoring History**

CREATE TRIGGER del\_std

BEFORE DELETE

ON public.student

FOR EACH ROW

EXECUTE PROCEDURE public.delstd();

CREATE FUNCTION public.delstd()

RETURNS trigger

LANGUAGE 'plpgsql'

COST 100

VOLATILE NOT LEAKPROOF

AS $BODY$BEGIN

IF TG\_OP='DELETE' THEN

DELETE FROM tutoring where "Student id"=OLD."ID";

RETURN OLD;

END IF;

END;$BODY$;

ALTER FUNCTION public.delstd()

OWNER TO postgres;

**2. Delete Video if Feedback < 25%**

CREATE TRIGGER delfeedback

AFTER INSERT OR DELETE OR UPDATE

ON public.videos

FOR EACH ROW

EXECUTE PROCEDURE public.del\_if\_low\_like();

CREATE FUNCTION public.del\_if\_low\_like()

RETURNS trigger

LANGUAGE 'plpgsql'

COST 100

VOLATILE NOT LEAKPROOF

AS $BODY$BEGIN

IF TG\_OP='UPDATE' THEN

DELETE FROM public."Question Video" where vid\_id=NEW.vid\_id AND like\_per(NEW.vid\_id) < 25;

DELETE FROM lessons where vid\_id=NEW.vid\_id AND like\_per(NEW.vid\_id) < 25;

RETURN NEW;

END IF;

END;$BODY$;

ALTER FUNCTION public.del\_if\_low\_like()

OWNER TO postgres;

# **DDLs**

**1.**

CREATE TABLE public.answered

(

sl integer NOT NULL DEFAULT nextval('answered\_sl\_seq'::regclass),

std\_id integer,

q\_id integer,

correct integer,

subj character(1) COLLATE pg\_catalog."default",

time\_sec integer,

CONSTRAINT answered\_pkey PRIMARY KEY (sl)

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.answered

OWNER to postgres;

**2.**

CREATE TABLE public."Lesson Notes"

(

std\_id integer,

lesson\_id integer,

note text COLLATE pg\_catalog."default" NOT NULL,

sl integer NOT NULL DEFAULT nextval('"Lesson Notes\_sl\_seq"'::regclass),

CONSTRAINT sl PRIMARY KEY (sl),

CONSTRAINT lesson\_id FOREIGN KEY (lesson\_id)

REFERENCES public.lessons ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT std\_id FOREIGN KEY (std\_id)

REFERENCES public.student ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public."Lesson Notes"

OWNER to postgres;

CREATE INDEX fki\_lesson\_id

ON public."Lesson Notes" USING btree

(lesson\_id)

TABLESPACE pg\_default;

CREATE INDEX fki\_std\_id

ON public."Lesson Notes" USING btree

(std\_id)

TABLESPACE pg\_default;

**3.**

CREATE TABLE public.tutoring

(

"Subject" character(30) COLLATE pg\_catalog."default",

"start date" date,

"end date" date,

feedback text COLLATE pg\_catalog."default",

"Student id" integer,

"Tutor id" integer,

"tutoring id" integer NOT NULL DEFAULT nextval('"Tutoring\_tutoring id\_seq"'::regclass),

CONSTRAINT "Tutoring\_pkey" PRIMARY KEY ("tutoring id"),

CONSTRAINT "Student id" FOREIGN KEY ("Student id")

REFERENCES public.student ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT "Tutor id" FOREIGN KEY ("Tutor id")

REFERENCES public.tutor ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.tutoring

OWNER to postgres;

CREATE INDEX "fki\_Student id"

ON public.tutoring USING btree

("Student id")

TABLESPACE pg\_default;

CREATE INDEX "fki\_Tutor id"

ON public.tutoring USING btree

("Tutor id")

TABLESPACE pg\_default;

**4.**

CREATE TABLE public."Question Video"

(

question\_id integer,

link text COLLATE pg\_catalog."default",

explanation text COLLATE pg\_catalog."default",

vid\_id integer,

sl integer NOT NULL DEFAULT nextval('"Question Video\_sl\_seq"'::regclass),

title character(50) COLLATE pg\_catalog."default",

CONSTRAINT "Question Video\_pkey" PRIMARY KEY (sl),

CONSTRAINT q\_id FOREIGN KEY (question\_id)

REFERENCES public.questions ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public."Question Video"

OWNER to postgres;

CREATE INDEX fki\_vid\_id

ON public."Question Video" USING btree

(vid\_id)

TABLESPACE pg\_default;

CREATE TRIGGER delvid\_ques

BEFORE INSERT OR DELETE OR UPDATE

ON public."Question Video"

FOR EACH ROW

EXECUTE PROCEDURE public.delvid\_q();

**5.**

CREATE TABLE public.evaluation

(

"time" numeric,

score numeric,

std\_id integer,

q\_id integer,

sl integer NOT NULL DEFAULT nextval('evaluation\_sl\_seq'::regclass),

CONSTRAINT evaluation\_pkey PRIMARY KEY (sl),

CONSTRAINT std\_id FOREIGN KEY (std\_id)

REFERENCES public.student ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.evaluation

OWNER to postgres;

**6.**

CREATE TABLE public.lessons

(

subject character(30) COLLATE pg\_catalog."default",

"tutor id" integer,

"ID" integer NOT NULL,

v\_link text COLLATE pg\_catalog."default",

vid\_id integer,

title character(50) COLLATE pg\_catalog."default",

CONSTRAINT "Lessons\_pkey" PRIMARY KEY ("ID"),

CONSTRAINT subj FOREIGN KEY (subject)

REFERENCES public.subjects (subj\_name) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT "tutor id" FOREIGN KEY ("tutor id")

REFERENCES public.tutor ("ID") MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION,

CONSTRAINT vid\_id FOREIGN KEY (vid\_id)

REFERENCES public.videos (sl) MATCH SIMPLE

ON UPDATE NO ACTION

ON DELETE NO ACTION

)

WITH (

OIDS = FALSE

)

TABLESPACE pg\_default;

ALTER TABLE public.lessons

OWNER to postgres;

CREATE INDEX fki\_subj

ON public.lessons USING btree

(subject COLLATE pg\_catalog."default")

TABLESPACE pg\_default;

CREATE INDEX "fki\_tutor id"

ON public.lessons USING btree

("tutor id")

TABLESPACE pg\_default;

CREATE TRIGGER delvid\_ques

BEFORE INSERT OR DELETE OR UPDATE

ON public.lessons

FOR EACH ROW

EXECUTE PROCEDURE public.delvid\_l();