

DATABASE DESIGN FOR EDUCATION TESTING SERVICES

CS/SE 6360.002 FINAL PROJECT

Group Members:

Pooja Kudav (pvk170030)

Sohini Mitra (sxm166731)

TABLE OF CONTENTS

SR. NO.		TOPIC	PAGE NO.
1		Requirements	3
2		Modelling ER-diagram	5
3		Mapping ERD to Relational Schema	7
4		SQL statements to create tables in database.	10
5		PL/SQL-Triggers	14
		Trigger-I	14
		Trigger-II	15
6		PL/SQL-Procedures	16
		Procedure-I	16
		Procedure-II	17

REQUIREMENT ANALYSIS

STUDENT

The Student is required to first register within the testing system to avail all the facilities provided to him. The facilities include:

- Registering for the test
- Sending scores to the institution.
- View scores.
- Taking the test.
- Providing feedback.

FEEDBACK

The student provides a feedback regarding his experience while taking the test. Whether there is any scope for improvement. Which facilities need to be upgraded and other details which will help ETS to assess its progress and figure out the areas in which improvements needed to be done.

INVOICE

The user receives an invoice at the end of the transaction. The invoice will help the user to track his order details. The invoice is a commercial document issued by ETS to a student who places an order for a test, relating to a sale transaction and indicating the products, quantities, and standard prices for products or services ETS had provided to the student.

CENTER

The student will take the test in a particular centre which he had selected while registering for the test. This will provide information regarding the location and capacity of the centre. No student are allowed to register for the test if the capacity of the centre is exceeded for a particular test date.

INSTITUTES

The students send their scores to the institute. While ordering for the test or after the test the students can send the scores to the institution. The codes are institution specific. The student can send scores to multiple institutes.

ANSWER

The answer provided by student for all the questions of the test is stored in this table. This helps to calculate marks obtained by the student and thereby the percentile rank of all the students.

QUESTION

The test will have questions which will have multiple choices and a correct answer.

ORDER DETAILS

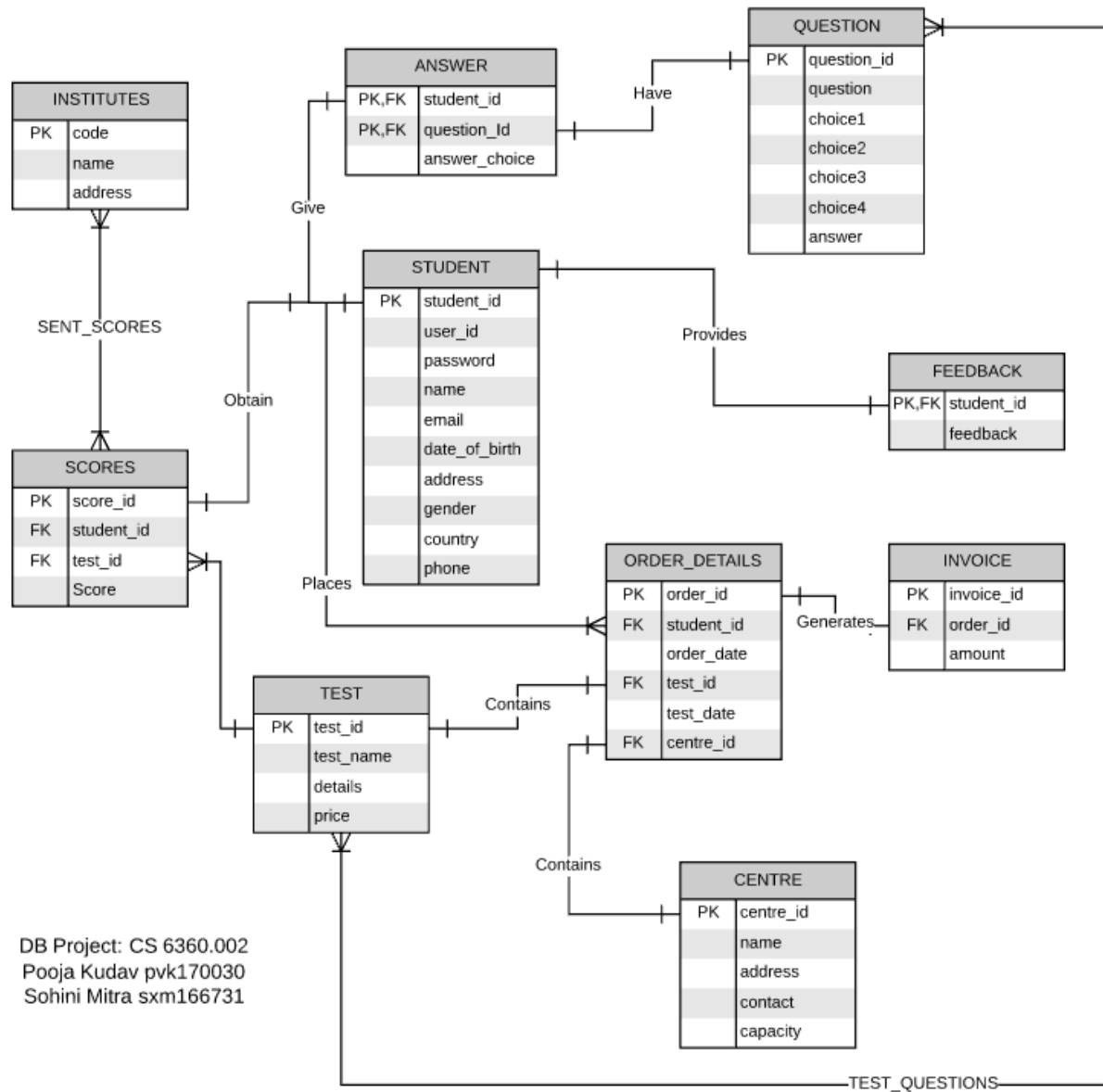
Provide customer with order specific details like date of the test, centre, and amount. The amount is fixed for a particular test. The student can order for more than one tests but he cannot give the same test within 21 days of taking the test i.e. the gap between two tests should be 21 days.

TESTS

Same test can be given more than once. The student has following option while sending the scores to the institutions.

- **Most Recent** — Send your scores from your most recent test administration.
- **All** — Send your scores from all test administrations in the last five years.
- **Any** — Send your scores from one OR as many test administrations as you like from the last five years.

ER-DIAGRAM



As per the ER diagram the requirements can be summarized as follows:

1. A STUDENT can provide only one FEEDBACK and only one feedback can be linked to a student (1:1).
2. Multiple SCORES can be sent to a Institute and an Institute can receive multiple scores (M:N).
3. A QUESTION can have only one ANSWER (the one which is given by student) and an ANSWER is linked to only one QUESTION (1:1).
4. A STUDENT can place one or more ORDER, whereas an ORDER is linked to a single STUDENT(1:M).
5. Details of a particular order generates only one INVOICE, whereas a particular INVOICE is linked to a single ORDER.(1:1).
6. A TEST can have many Question sets and thereby many questions, whereas a Question can be a part of Multiple TESTS (M:N)
7. A student can have multiple SCORES for a single TEST, whereas a particular SCORE can have only one TEST(M:1)

MAPPING ERD IN RELATIONAL SCHEMA:

STUDENT

<u>student_id</u>	user_id	password	name	email	date_of_birth	address	gender	country	phone
-------------------	---------	----------	------	-------	---------------	---------	--------	---------	-------

Primary key: student_id

TEST

<u>test_id</u>	test_name	details	price
----------------	-----------	---------	-------

Primary key: test_id

QUESTION

<u>question_id</u>	question	choice1	choice2	choice3	choice4	answer
--------------------	----------	---------	---------	---------	---------	--------

Primary key: question_id

CENTRE

<u>centre_id</u>	name	address	contact	capacity
------------------	------	---------	---------	----------

Primary key: centre_id

INSTITUTES

<u>code</u>	name	address
-------------	------	---------

Primary key: code

TEST_QUESTIONS

question_set_id	<u>test_id</u>	<u>question_id</u>
-----------------	----------------	--------------------

Primary Key: question_set_id, test_id, question_id

Foreign Key: question_id references QUESTION (question_id)
test_id references TEST(test_id)

EXAM_DATES

<u>test_id</u>	<u>centre_id</u>	exam_date
----------------	------------------	-----------

Primary key: test_id, centre_id

Foreign key: test_id references TEST(test_id)
centre_id references CENTRE(centre_id)

ANSWER

<u>student_id</u>	<u>question_id</u>	answer_choice
-------------------	--------------------	---------------

Primary Key: student_id, question_id

Foreign Key: student_id references STUDENT(student_id)
question_id references QUESTION(question_id)

FEEDBACK

<u>student_id</u>	feedback
-------------------	----------

Primary Key: student_id

Foreign Key: student_id references STUDENT(student_id)

ORDER_DETAILS

<u>order_id</u>	student_id	order_date	test_id	test_date	centre_id
-----------------	------------	------------	---------	-----------	-----------

Primary Key: order_id

Foreign Key: student_id references STUDENT (student_id)
test_id references TEST (test_id)
centre_id references CENTRE (centre_id)

SCORES

<u>score_id</u>	student_id	test_id	score
-----------------	------------	---------	-------

Primary Key: score_id

Foreign Key: student_id references STUDENT (student_id)
test_id references TEST (test_id)

SEND SCORES

<u>score_id</u>	code
-----------------	------

Primary Key: **score_id**, code

Foreign Key: score_id references SCORE (score_id)
code references INSTITUTES (code)

INVOICE

<u>invoice_id</u>	order_id	amount
-------------------	----------	--------

Primary Key: invoice_id

Foreign Key: order_id references ORDER (order_id)

SQL STATEMENTS TO CREATE TABLES IN DATABASE

```
create table STUDENT
( student_id int primary key,
  user_id varchar(10) unique,
  password varchar(10) not null,
  name varchar(10) not null,
  email varchar(20) not null,
  date_of_birth date not null,
  address varchar(20) not null,
  phone varchar(10) not null,
  gender varchar(1) not null,
  country varchar(10) not null
);

create table TEST
( test_id int primary key,
  test_name varchar(10) unique not null,
  details varchar(20),
  price decimal not null
);

create table QUESTION
( question_id int primary key,
  question varchar(50) unique not null,
  choicel varchar(20) not null,
  choice2 varchar(20) not null,
  choice3 varchar(20),
  choice4 varchar(20),
  answer varchar(20) not null
);

create table CENTRE
( centre_id int primary key,
  name varchar(10) not null,
  address varchar(20) not null,
  contact varchar(10) not null,
  capacity int not null
);

create table INSTITUTES
( code int primary key,
  name varchar(20) not null,
  address varchar(30) not null
);

create table TEST_QUESTIONS
```

```
( question_set_id int not null,
  test_id int not null,
  question_id int not null,
  primary key (question_set_id, test_id, question_id),
  foreign key (question_id) references QUESTION(question_id ) on
delete cascade,
  foreign key (test_id) references TEST(test_id) on delete cascade
);
```

```
create table EXAM_DATES
( test_id int not null,
  centre_id int not null,
  exam_date date not null,
  primary key(test_id, centre_id),
  foreign key (test_id) references TEST(test_id),
  foreign key (centre_id) references CENTRE(centre_id)
);
```

```
create table ANSWER
( student_id int not null,
  question_id int not null,
  answer_choice varchar(1),
  primary key(student_id, question_id),
  foreign key(student_id) references STUDENT(student_id),
  foreign key(question_id) references QUESTION(question_id)
);
```

```
create table FEEDBACK
( student_id int primary key,
  feedback varchar(50),
  foreign key(student_id) references STUDENT(student_id)
);
```

```
create table ORDER_DETAILS
( order_id int primary key,
  order_date date not null,
  student_id int not null,
  centre_id int not null,
  test_id int not null,
  test_date date not null,
  foreign key(student_id) references STUDENT(student_id),
  foreign key(test_id) references TEST(test_id),
  foreign key(centre_id) references CENTRE(centre_id)
);
```

```
create table SCORES
( score_id int primary key,
  student_id int not null,
  test_id int not null,
  score int not null,
  foreign key (student_id) references STUDENT(student_id),
  foreign key (test_id) references TEST(test_id)
```

```
);
```

```
create table SEND_SCORES  
( score_id int not null,  
  code int not null,  
  foreign key (score_id) references SCORES(score_id)  
);
```

```
create table INVOICE  
( invoice_id int primary key,  
  order_id int not null,  
  amount decimal not null,  
  foreign key(order_id) references ORDER_DETAILS(order_id)  
);
```

NORMALIZATION OF RELATIONAL SCHEMA INTO 3NF

1. STUDENT -> {student_id -> user_id, password, name, email, date_of_birth, address, phone, gender, country}
2. FEEDBACK -> {student_id -> feedback}
3. CENTRE -> {centre_id -> name, address, contact, capacity}
4. QUESTION -> {question_id -> question, choice1, choice2, choice3, choice4, answer}
5. ANSWER -> {(student_id, question_id) -> answer_choice}
6. INVOICE -> {invoice_id -> order_id, amount}
7. ORDER_DETAIL -> {(order_id) -> order_date, student_id, centre_id, test_id, test_date,}
8. INSTITUTES -> {code -> name, address}
9. TEST -> {test_id -> test_name, details, price}
10. SCORES -> {score_id -> student_id, test_id, score}
11. SEND_SCORES -> {score_id, code}
12. EXAM_DATES -> {test_id, centre_id, exam_date}

The above functional dependencies cause the schema to be in third normal form.

PL/SQL

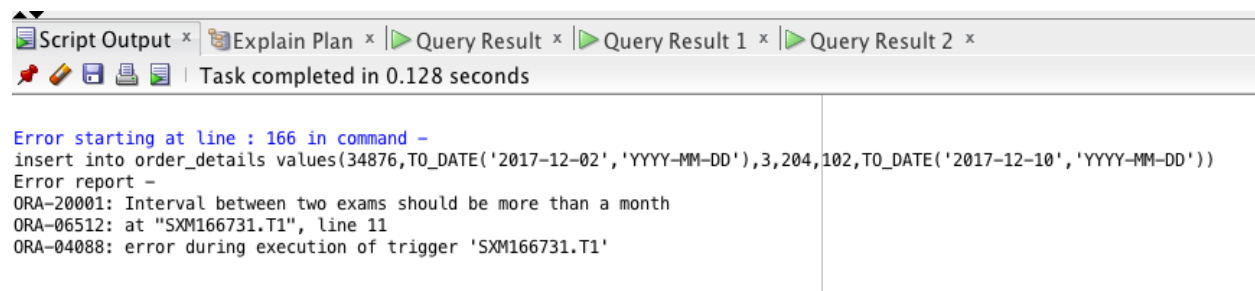
TRIGGER-I

If a student wishes to give take a test that he/she has already taken before, then the student is allowed to reserve a date that is at least one month after the previous attempt. The first trigger ensures that

```
204 SET SERVEROUTPUT ON size 30000
205 CREATE OR REPLACE TRIGGER T1
206 BEFORE INSERT ON order_details
207 FOR EACH ROW
208 DECLARE
209 old_date date;
210 new_date date;
211 BEGIN
212 dbms_output.enable;
213 new_date := :NEW.test_date;
214 select order_details.test_date into old_date from order_details
215 where order_details.student_id = :NEW.student_id and order_details.test_id = :NEW.test_id;
216 if (ADD_MONTHS(old_date, 1) > new_date) then
217     raise_application_error(-20001, 'Interval between two exams should be more than a month');
218 end if;
219 END;
220
```

Output

Upon trying to insert a row that where the student is attempting to reserve a date within the one month interval, for the same test taken before, the trigger is fired.



The screenshot shows the SQL Developer interface with the 'Script Output' tab selected. It displays an error message starting at line 166 in the command, which is an insert statement into the 'order_details' table. The error report shows 'ORA-20001: Interval between two exams should be more than a month' and 'ORA-06512: at "SXM166731.T1", line 11'. The task completed in 0.128 seconds.

```
Error starting at line : 166 in command -
insert into order_details values(34876,TO_DATE('2017-12-02','YYYY-MM-DD'),3,204,102,TO_DATE('2017-12-10','YYYY-MM-DD'))
Error report -
ORA-20001: Interval between two exams should be more than a month
ORA-06512: at "SXM166731.T1", line 11
ORA-04088: error during execution of trigger 'SXM166731.T1'
```

However, when the interval is more than a month, the record is allowed to be inserted.

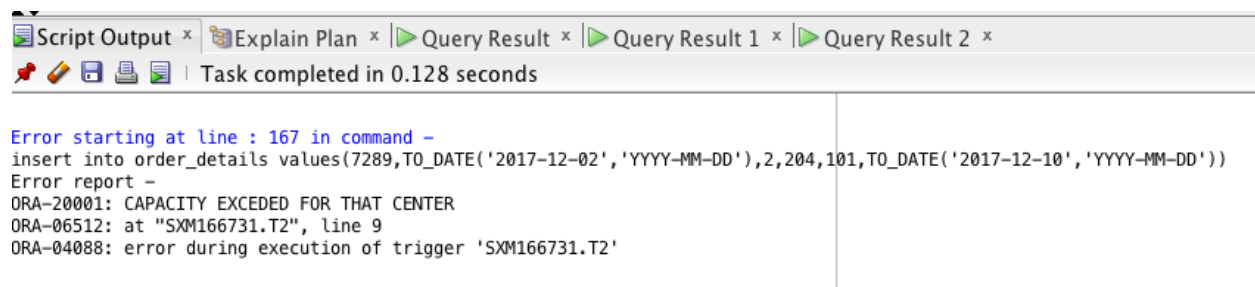
	ORDER_ID	ORDER_DATE	STUDENT_ID	CENTRE_ID	TEST_ID	TEST_DATE
1	34876	02-DEC-17	3	204	102	10-DEC-18
2	1234	02-FEB-17	1	204	101	03-DEC-17
3	7489	06-APR-17	1	201	104	09-AUG-17
4	1289	07-JAN-17	3	201	102	05-DEC-17

TRIGGER-II

The second trigger is fired whenever a reservation is being attempted at a center whose maximum capacity has been reached.

```
222 SET SERVEROUTPUT ON;
223 CREATE OR REPLACE TRIGGER T2
224 BEFORE INSERT ON order_details
225 FOR EACH ROW
226 DECLARE
227 total int;
228 c int;
229 BEGIN
230 select distinct count(*) into total from order_details
231 where order_details centre_id = :NEW.centre_id and order_details.test_date = :NEW.test_date;
232 select capacity into c from centre where centre_id= :NEW.centre_id;
233 if((total+1) > c) then
234 raise_application_error(-20001,'CAPACITY EXCEDED FOR THAT CENTER');
235 end if;
236 END;
237
```

Output



The screenshot shows the SQL Developer interface with a script window titled 'Script Output x'. The script contains an INSERT statement and a trigger. The output shows an error starting at line 167 in the command, which is the INSERT statement. The error report shows three messages: ORA-20001: CAPACITY EXCEDED FOR THAT CENTER, ORA-06512: at "SXM166731.T2", line 9, and ORA-04088: error during execution of trigger 'SXM166731.T2'.

```
Error starting at line : 167 in command -
insert into order_details values(7289,TO_DATE('2017-12-02','YYYY-MM-DD'),2,204,101,TO_DATE('2017-12-10','YYYY-MM-DD'))
Error report -
ORA-20001: CAPACITY EXCEDED FOR THAT CENTER
ORA-06512: at "SXM166731.T2", line 9
ORA-04088: error during execution of trigger 'SXM166731.T2'
```

PROCEDURE-I

The first procedure retrieves the information of all the tests that have been attempted by the student using the student id.

```
259 set serveroutput on;
260 create or replace procedure P1(sid in STUDENT.student_id%type) as
261 cursor c is
262 select student.name as sname, order_details.test_date as tdate, test.test_name as tname, centre.name as cname, scores.score as sscore
263 from student, order_details, test, centre, scores
264 where student.student_id = sid and order_details.student_id = student.student_id
265 and order_details.test_id = test.test_id and order_details.centre_id = centre.centre_id and student.student_id = scores.student_id and test.test_id = scores.test_id;
266 iterator c%rowtype;
267 begin
268 open c;
269 loop
270 fetch c into iterator;
271 exit when(c%NOTFOUND);
272 dbms_output.put_line('-----');
273 dbms_output.put_line('TEST           : ' || iterator.tname);
274 dbms_output.put_line('DATE OF TEST    : ' || iterator.tdate);
275 dbms_output.put_line('CENTER          : ' || iterator.cname);
276 dbms_output.put_line('SCORE OBTAINED  : ' || iterator.sscore);
277 dbms_output.put_line('-----');
278 end loop;
279 close c;
280 end;
```

Output

PROCEDURE-II

The second procedure allows us to view the scores of a particular test in descending order.

```
295 set serveroutput on;
296 create or replace procedure p2(testname in TEST.test_name%type) as
297 cursor cu is
298 select score from SCORES, TEST
299 where scores.test_id = test.test_id and test.test_name = testname
300 order by score desc;
301 iterator cu%rowtype;
302 begin
303 open cu;
304 loop
305 fetch cu into iterator;
306 exit when (cu%NOTFOUND);
307 dbms_output.put_line(iterator.score);
308 end loop;
309 close cu;
310 end;
311
```

Output

```
Procedure P2 compiled

95
89
70
50

PL/SQL procedure successfully completed.
```