

Task No 12:Deleting record from Instructor Table

```
8
9 • DELETE FROM Teaches
0 WHERE Instructor_ID = 'I002';
1 • Delete From Instructor
2 where Instructor_ID='I002';
3 • select*from Instructor;
4
5 • SELECT * FROM Student;
6
7
8 • SELECT * FROM Course;
```

Input :

Action Output

#	Time	Action	Message
1	02:09:29	Delete From Instructor where Instructor_ID='I002'	Error Co
2	02:10:37	DELETE FROM Teaches WHERE Instructor_ID = 'I002'	1 row(s)
3	02:10:37	Delete From Instructor where Instructor_ID='I002'	1 row(s)

12. Delete Data (DML)

Deleting a record from the **Instructor** table.

```
DELETE FROM Instructor
WHERE Instructor_ID = 'I003';
```

The **Instructor** table after the deletion:

Instructor_ID	Name	Department	Email
I001	Dr. Steve	Computer Science	steve@university.edu
I002	Dr. Mary	Mathematics	mary@university.edu
I004	Dr. Alice	Biology	alice@university.edu
I005	Dr. Bob	Engineering	bob@university.edu

Explanation: Shows an error as the entry is not deleted from child table(Teaches) .It will work if the entry is deleted from the Teaches Table first

Task No 13: Performing Natural Join Operation

The screenshot displays a database query editor with the following SQL code:

```
174 • select Instructor.name ,Teaches.Course_ID
175 from Instructor
176 natural join Teaches ;
177
178
179
180
181
182
183
184
185
186
```

Below the query editor, the results are shown in a table:

name	Course_ID
Dr. Mary	C102
Dr. John	C103
Dr. Alice	C104
Dr. Bob	C105

On the right side of the image, there is a Google account interface. It shows a profile picture with the letter 'F' and the name 'Hi, Fahmid!'. Below this is a button that says 'Manage your Google Account'. Further down, there is a section for storage usage, indicating '98% of 15 GB used' and '14.79 GB of 15 GB'. At the bottom, there are links for 'Get storage' and 'Clean up space'.

Explanation: Natural Join connect multiple tables based on the common columns

Task No 14: Performing Drop Table Command

174 • `select *from Instructor;`
175 • `select *from Teaches;`
176 • `select Instructor.name ,`
177 `from Instructor`
178
179 `natural join Teaches ;`
180
181 • `drop table Teaches;`
182 • `select *from Teaches;`
183
184
185
186
187
188
189
190
191
192

Output

Action Output

#	Time	Action
✓ 68	21:29:06	<code>select *from Teaches LIMIT</code>
✓ 69	21:32:48	<code>select Instructor.name ,Tea</code>
✓ 70	21:33:41	<code>select Instructor.name ,Tea</code>
✓ 71	22:21:14	<code>drop table Teaches</code>
✗ 72	22:21:29	<code>select *from Teaches LIMIT</code>

F

Hi, Fahmid!

Manage your Google Account

Show more accounts

f Z S v

98% of 15 GB used

14.79 GB of 15 GB

Get storage Clean up space

Privacy Policy • Terms of Service

CONSTRAINTS

INSERT INTO Takes (Student_ID, Course_ID, Sem

VALUES (7, 'C102', 'Spring', 2023);

This will fail because **Student_ID = 7** does not ex

16. SQL DROP Command

Duration / Fetch

0.000 sec / 0.00

0.000 sec / 0.00

0.000 sec / 0.00

0.156 sec

0.015 sec

Task No 15: Foreign Key Constraints and Update

```
184
185 • INSERT INTO Takes (Student_ID, Course_ID, Semester, Year)
186 VALUES (7, 'C102', 'Spring', 2023);
187
188
189
190
191
192
193
194
```

Output

Action Output

#	Time	Action	Message
69	21:32:48	select Instructor.name, Teaches.Course_ID from Instructor natural jo...	4 row(s)
70	21:33:41	select Instructor.name, Teaches.Semester from Instructor natural joi...	4 row(s)
71	22:21:14	drop table Teaches	0 row(s)
72	22:21:29	select * from Teaches LIMIT 0, 10000	Error Co
73	22:24:37	INSERT INTO Takes (Student_ID, Course_ID, Semester, Year) VA...	Error Co

For example, if we try to insert a row into the **Takes** table that references a non-existing student, the command will fail due to the foreign key constraint.

```
INSERT INTO Takes (Student_ID, Course_ID, Semester, Year)
VALUES (7, 'C102', 'Spring', 2023);
```

This will fail because **Student_ID = 7** does not exist in the **Student** table.

16. SELECT Command

```
SELECT name
FROM instructor;
```

Output: This command selects the **name** column from the **Instructor** table.

Name

Explanation: An error occurs because the `student_ID` being inserted into the **Takes** table does not exist in the parent table **Student**. Since `student_ID` is defined as a foreign key referencing the **Student** table, the database enforces referential integrity. Therefore, inserting a record with a `student_ID` (e.g., 7) that does not exist in the **Student** table violates the foreign key constraint and results in an error.

Task No 16: Performing Select Command

The screenshot shows a database practice environment. On the left, a SQL editor window titled 'PRACTICE2' contains the query `select name from Instructor;`. Below the editor, a 'Result Grid' displays the output of the query, showing a list of names: Dr. John, Dr. Alice, and Dr. Bob. On the right, a PDF document titled 'Database Lab.pdf' is open, showing the SQL query `FROM Student, StudentCourse;` and an explanation: 'Output: This query generates a Cartesian product of the Student and StudentCourse tables. It combines each row of the Student table with each row of the StudentCourse table.' Below this text is a table showing the result of the Cartesian product:

Name	Age	COURSE_ID
John Smith	20	C101
John Smith	20	C102
John Smith	20	C103
John Smith	20	C104
John Smith	20	C105
Alice Brown	22	C101
Alice	22	C102

On the far right, a Chrome browser sidebar is visible, showing the user's profile 'Fahmid' with the email 'fahmiduxxaman@gmail.com' and various account management options.

Explanation: Shows the remaining entries of Instructor Table as I have done delete operation earlier

Task No 17: Performing Cross Join on Student and StudentCourse Table

```
7 select Student.Name, Student.Age, StudentCourse.Course_ID
8 from Student, StudentCourse;
9
```

Name	Age	Course_ID
John Smith	20	C105
John Smith	20	C104
John Smith	20	C103
John Smith	20	C102
John Smith	20	C101
Alice Brown	22	C105
Alice Brown	22	C104
Alice Brown	22	C103
Alice Brown	22	C102
Alice Brown	22	C101
Bob White	21	C105
Bob White	21	C104
Bob White	21	C103
Bob White	21	C102
Bob White	21	C101
Carol Green	23	C105
Carol Green	23	C104
Carol Green	23	C103
Carol Green	23	C102
Carol Green	23	C101
Dave Black	24	C105

```
INSERT INTO Takes (StudentID, CourseID, SectionID)
VALUES (7, 'C102', 'Spring 2019');


This will fail because StudentID 7 does not exist in the Student table.

16. SELECT Command

SELECT name
FROM instructor;


Output: This command returns the names of all instructors in the
Instructor table.


Name
Dr. Steve
Dr. Mary
```



Hi, Fahmid!

[Manage your Google Account](#)

Show more accounts 

 98% of 15 GB used
14.79 GB of 15 GB

[Get storage](#) [Clean up](#)

[Privacy Policy](#) • [Terms of Service](#)

Explanation : Cross Join (also known as a Cartesian product) combines every row from the first table with every row from the second table, creating all possible combinations

Task No 18: Cross Join (Alternative Syntax)

```
187 select Student.Name, Student.Age, StudentCourse.Course_ID
188 from Student cross join StudentCourse;
189
190
191
192
193
194
```

Result Grid

Name	Age	Course_ID
John Smith	20	C105
John Smith	20	C104
John Smith	20	C103
John Smith	20	C102
John Smith	20	C101
Alice Brown	22	C105
Alice Brown	22	C104
Alice Brown	22	C103
Alice Brown	22	C102
Alice Brown	22	C101
Bob White	21	C105
Bob White	21	C104
Bob White	21	C103
Bob White	21	C102
Bob White	21	C101
Carol Green	23	C105
Carol Green	23	C104
Carol Green	23	C103
Carol Green	23	C102
Carol Green	23	C101
Dave Black	24	C105

File Edit View Insert Format Tools Extensions ...

18. Cross Join (Alternative Syntax)

SELECT Student.NAME, S
FROM Student CROSS JO

Output: This will yield the same result as the previous example, but it's using a different syntax.

Name	Age	COURSE_ID
John Smith	20	C105
John Smith	20	C104
John Smith	20	C103
John Smith	20	C102
John Smith	20	C101

fahmiduxxaman@gmail.com

Hi, Fahmid!

[Manage your Google Account](#)

Show more accounts

98% of 15 GB used

14.79 GB of 15 GB

[Get storage](#) [Clean up space](#)

[Privacy Policy](#) • [Terms of Service](#)

Task No 19: Performing Explicit Inner Join operation

```
199 • INSERT INTO Teaches (Instructor_ID, Course_ID, Semester, Year) VALUES
200 ('I002', 'C102', 'Spring', 2023),
201 ('I003', 'C103', 'Fall', 2023),
202 ('I004', 'C104', 'Spring', 2023),
203 ('I005', 'C105', 'Fall', 2023);
204 select Name, Course_ID
205 from Instructor, Teaches
206 where Instructor.Instructor_ID=Teaches.Instructor_ID;
```

which course?

```
CREATE TABLE Teaches (
  Instructor_ID VARCHAR(10),
  Course_ID VARCHAR(10),
  Semester VARCHAR(10),
  Year INT,
  PRIMARY KEY (Instructor_ID, Course_ID, Semester, Year),
  KEY (Instructor_ID) REFERENCES Instructor (Instructor_ID),
  KEY (Course_ID) REFERENCES Course (Course_ID),
  KEY (Semester, Year) REFERENCES Semester_Year (Semester, Year);
```

Instructor_ID	Course_ID	Semester
I001	C101	Fall
I002	C102	Spring
I003	C103	Fall
I004	C104	Spring
I005	C105	Fall

Name	Course_ID
Dr. Mary	C102
Dr. John	C103
Dr. Alice	C104
Dr. Bob	C105

fahmiduxxaman@gmail.com

Hi, Fahmid!

[Manage your Google Account](#)

Show more accounts

98% of 15 GB used

14.79 GB of 15 GB

[Get storage](#) [Clean up space](#)

Task No 20:Performing Natural Join

209

210

211

212

213

214

215

216

217

•

select Name, Course_ID

From Instructor Natural Join Teaches;

Result Grid

Filter Rows

Export

Wrap Cell Contents

Name	Course_ID
Dr. Mary	C102
Dr. John	C103
Dr. Alice	C104
Dr. Bob	C105

20. Natural Join Simplified

SELECT name, course_id
FROM instructor NATURAL JOIN teaches;

Output: This is the simplified natural join of the instructor and teaches tables. It has the same output as the previous query.

Name	course_id
Dr. Steve	C101
Dr. Mary	C102
Dr. John	C103
Dr. Alice	C104
Dr. Bob	C105

F

Hi, Fahmid!

Manage your Google Account

Show more accounts

98% of 15 GB used

14.79 GB of 15 GB

Get storage

Clean up space

Privacy Policy




Terms of Service

Task No 21: Performing Multiple Natural Join

213 • Select Name, Course_ID, Age
214 from Instructor natural join Teaches natural join Student;
215
216

VS

ER

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

	Name	Course_ID	Age
--	------	-----------	-----

Explanation: It shows empty columns as there's no common columns among the three tables

Task No 22:Renaming Table

```
215 • alter table Student Rename To Student_Info;
216 • select *from Student_Info;
217
218
219
220
221
222
223
224
225
226
```

Result Grid					
Filter Rows:					
Edit: Export/Import: Wrap Cell Content:					
Student_ID	Name	Age	Major	Email	
1	John Smith	20	Computer Science	john@email.com	
2	Alice Brown	22	Mathematics	alice@email.com	
3	Bob White	21	Physics	bob@email.com	
4	Carol Green	23	Biology	carol@email.com	
5	Dave Black	24	Engineering	dave@email.com	
6	Eve White	25	Chemistry	eve@email.com	
* NULL	NULL	NULL	NULL	NULL	

ALTER TABLE Student R

Output: After renaming
name is now changed in

Table Name

Students_Info

Now, you would refer to
future operations.

23. STRING Manipulation

SELECT CONCAT(Name,

fahmiduxxaman@gmail.com

F

Hi, Fahmid!

Manage your Google Account

Show more accounts

98% of 15 GB used
14.79 GB of 15 GB
Get storage Clean up space

Privacy Policy Terms of Service

Task No 23: String Manipulation

215 • alter table Student Rename To Student_Info;

216 • select *from Student_Info;

217 • select concat(Name, '-',Major)As Student_Info

218 from Student_Info;

219

220

221

222

223

224

225

226

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

Student_Info

John Smith-Computer Science

Alice Brown-Mathematics

Bob White-Physics

Carol Green-Biology

Dave Black-Engineering

Eve White-Chemistry

23. STRING Manipulation Ex

SELECT CONCAT(Name, ' - ', M

FROM Student;

Output: This command uses th

and Major columns with a sepa

Student_Info.

Student_Info
John Smith - Computer Science
Alice Brown - Mathematics
Bob White - Physics

fahmiduxxaman@gmail.com

F

Hi, Fahmid!

Manage your Google Account

Show more accounts

98% of 15 GB used

14.79 GB of 15 GB

Get storage

Clean up space

Privacy Policy • Terms of Service

Task No 24: Sorting Using Order By

```

218 FROM Student_Info;
219 • select Name, Age
220 from Student_Info
221 order by Age desc;
222
223
224
225
226
227
228
229

```

24. ORDER BY Example

SELECT Name, Age
FROM Student
ORDER BY Age DESC;

Output: This command sorts the Student table and orders it by Age in descending order.

Name	Age
Dave Black	24
Carol Green	23
Alice Brown	22
Bob White	21

Task No 25: Example of Sorting with both ASC and DESC

```

226 • insert into Student_Info(Student_ID,Name,Age,Major,Email
227 values
228 (8,'Zamann',23,'Arts','zamann@gmail.com');
229 • select Name,Age,Major from Student_Info
230 order by Major asc ,Age desc;
231
232
233
234
235

```

FROM Student
ORDER BY Major ASC, Age DESC;

Output: This command sorts the Student table by Major in ascending order, and then by Age in descending order.

Name	Age	Major
Bob White	21	Physics
John Smith	20	Computer Science
Dave Black	24	Engineering
Alice Brown	22	Mathematics

Explanation : Here order by prioritize the first condition so checks the Majors in ascending order then if there are equal Major then sorts according to the 2nd condition age in descending order here Arts major is lexicographically smallest so it comes first

and there are two Rows Containing Major Arts so the got sorted by age in descending order according to the 2nd condition

Full Code->

https://github.com/Fahmiduzzaman2003/DBMS_Lab/blob/deab0e18db5ae4a2bf74026c4dad949918834f7d/Assignment2.sql

The screenshot displays a database management interface with three main components:

- SQL Editor:** Contains the following SQL code:

```
199 • INSERT INTO Teaches (Instructor_ID, Course_ID, Semester, Year) VALUES
200 ('I002', 'C102', 'Spring', 2023),
201 ('I003', 'C103', 'Fall', 2023),
202 ('I004', 'C104', 'Spring', 2023),
203 ('I005', 'C105', 'Fall', 2023);
204 select Name, Course_ID
205 from Instructor, Teaches
206 where Instructor.Instructor_ID=Teaches.Instructor_ID;
```
- Table Structure:** Shows the schema for the 'Teaches' table:

```
CREATE TABLE Teaches (
  Instructor_ID VARCHAR(10),
  Course_ID VARCHAR(10),
  Semester VARCHAR(10),
  Year INT,
  PRIMARY KEY (Instructor_ID, Course_ID),
  FOREIGN KEY (Instructor_ID) REFERENCES Instructor (Instructor_ID),
  FOREIGN KEY (Course_ID) REFERENCES Course (Course_ID);
```
- Google Account Notification:** A sidebar notification for 'fahmiduxxaman@gmail.com' with a profile picture 'F', the name 'Hi, Fahmid!', and a 'Manage your Google Account' button. It also shows storage usage: '98% of 15 GB used' (14.79 GB of 15 GB) with 'Get storage' and 'Clean up space' links.

184
185 •
186
187
188
189
190
191
192
193
194

```
INSERT INTO Takes (Student_ID, Course_ID, Semester, Year)
VALUES (7, 'C102', 'Spring', 2023);
```

Output

Action Output

#	Time	Action	Message
69	21:32:48	select Instructor.name ,Teaches.Course_ID from Instructor natural jo...	4 row(s)
70	21:33:41	select Instructor.name ,Teaches.Semester from Instructor natural joi...	4 row(s)
71	22:21:14	drop table Teaches	0 row(s)
72	22:21:29	select *from Teaches LIMIT 0, 10000	Error Co
73	22:24:37	INSERT INTO Takes (Student_ID, Course_ID, Semester, Year) VA...	Error Co

For example, if we try to insert a row into the **Takes** table that r
non-existing student, the command will fail due to the foreign k
constraint.

```
INSERT INTO Takes (Student_ID, Course_ID, Semester, Year)
VALUES (7, 'C102', 'Spring', 2023);
```

This will fail because **Student_ID = 7** does not exist in the **Stude**
16. SELECT Command

```
SELECT name
FROM instructor;
```

Output: This command selects the **name** column from the
Instructor table.

Name
