

# **COURSE NAME: DBMS**

## **COURSE CODE:23AD2102A**

**Topic: DATA MANIPULATION LANGUAGE (DML)**

**Session - 8**

## AIM OF THE SESSION



To familiarize students with the basic concept of SQL languages and Create table command in detail.

## INSTRUCTIONAL OBJECTIVES



This Session is designed to:

1. Discuss the types of SQL languages.
2. Various command under different types of SQL Languages.
3. Introduction of Create table command with its syntax and examples.

## LEARNING OUTCOMES



At the end of this session, you should be able to understand the basic commands of SQL and learn how to write queries with SQL commands.

## DML Commands in SQL

DML is an abbreviation of **Data Manipulation Language**.

The DML commands in Structured Query Language change the data present in the SQL database. We can easily access, store, modify, update and delete the existing records from the database using DML commands.

**Following are the four main DML commands in SQL:**

1. SELECT Command
2. INSERT Command
3. UPDATE Command
4. DELETE Command

**1) SELECT command:** This DML command allows us to access the stored records from the tables. We can also use the condition in the SELECT command for accessing the particular rows.

## Syntax of SELECT DML command

```
SELECT      column_Name_1,      column_Name_2,      .....,      column_Name_N  
FROM Name_of_table;
```

Here, **column\_Name\_1, column\_Name\_2, ....., column\_Name\_N** are the names of those columns whose data we want to retrieve from the table.

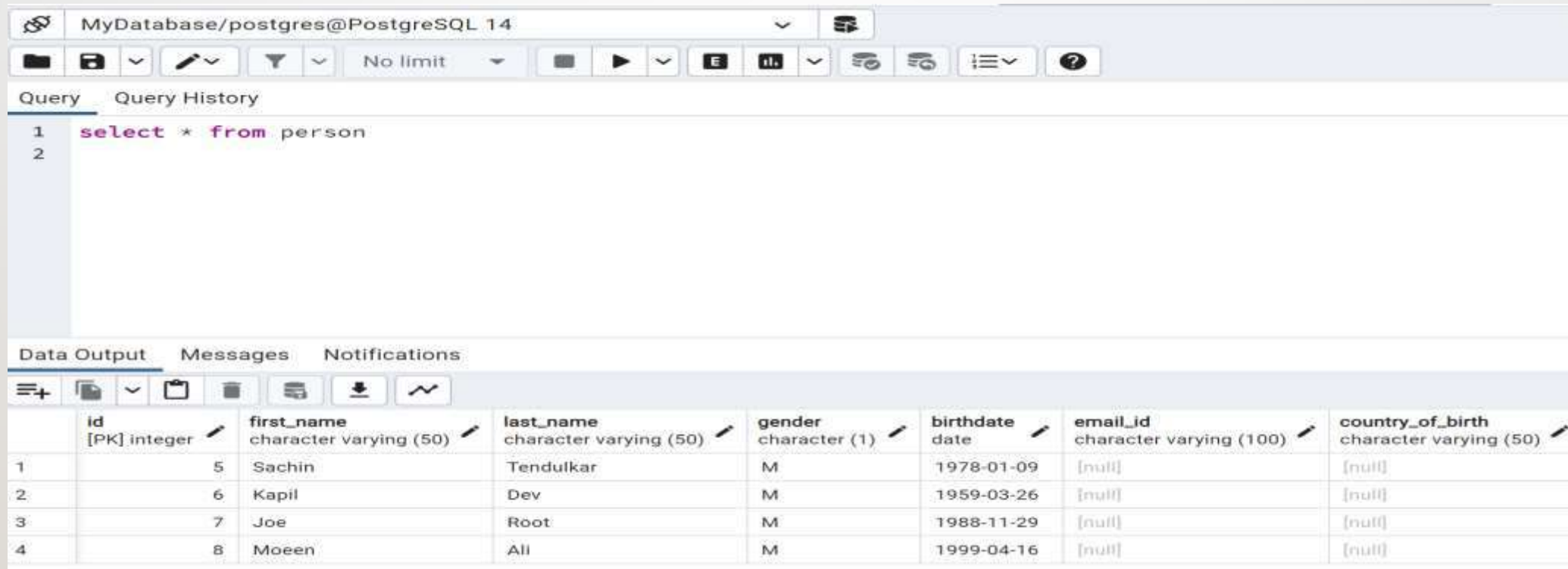
**If we want to retrieve all the data from all the columns of the table, we have to use the following SELECT command:**

```
SELECT * FROM table_name;
```

## Examples of SELECT Command

**Example 1:** This example shows all the values of every column from the table.

```
SELECT * FROM person;
```

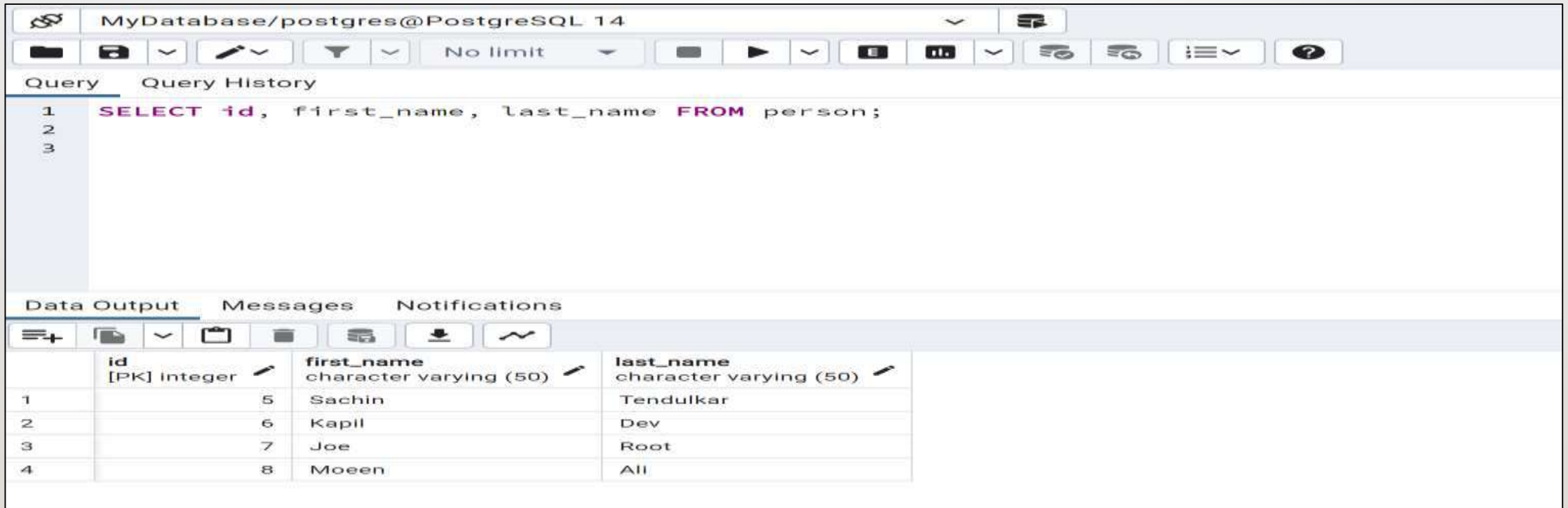


The screenshot shows a PostgreSQL query editor interface. The query editor has a toolbar with icons for saving, running, and other database operations. The query entered is `SELECT * FROM person;`. Below the query editor, the 'Data Output' tab is active, displaying a table with 8 columns and 4 rows of data.

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	5	Sachin	Tendulkar	M	1978-01-09	[null]	[null]
2	6	Kapil	Dev	M	1959-03-26	[null]	[null]
3	7	Joe	Root	M	1988-11-29	[null]	[null]
4	8	Moeen	Ali	M	1999-04-16	[null]	[null]

**Example 2:** The following **SELECT** statement retrieves the data from multiple columns of the employee table.

```
SELECT id, first_name, last_name FROM person;
```



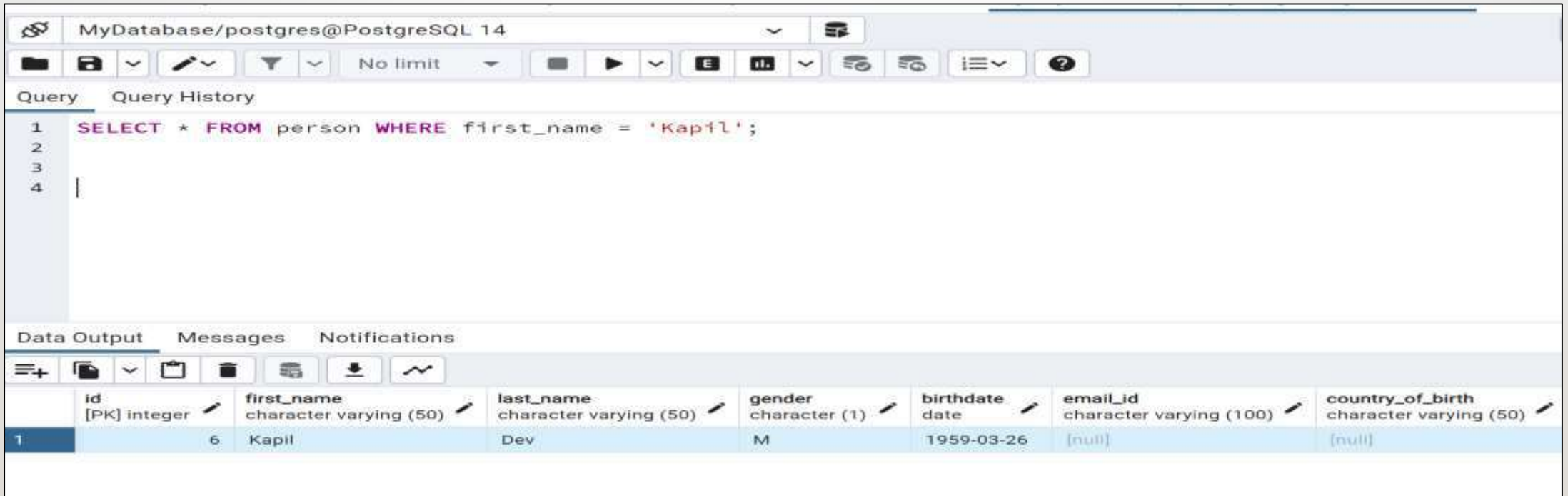
The screenshot shows a database query tool interface. The top bar indicates the connection is 'MyDatabase/postgres@PostgreSQL 14'. Below the toolbar, the 'Query' tab is active, displaying the SQL statement: `SELECT id, first_name, last_name FROM person;`. The 'Data Output' tab is also visible, showing the results of the query in a table format. The table has four columns: 'id' (integer, primary key), 'first\_name' (character varying (50)), and 'last\_name' (character varying (50)). The results show four rows of data.

	id [PK] integer	first_name character varying (50)	last_name character varying (50)
1	5	Sachin	Tendulkar
2	6	Kapil	Dev
3	7	Joe	Root
4	8	Moeen	All

**Example 3:** This example describes how to use the **WHERE clause with the SELECT** command.

If we want to access all the records of those person whose first\_name is Kapil.

```
SELECT * FROM person WHERE first_name = 'Kapil';
```



The screenshot shows a PostgreSQL query editor interface. The query entered is:

```
1 SELECT * FROM person WHERE first_name = 'Kapil';
```

The results are displayed in a table with the following columns and data:

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	6	Kapil	Dev	M	1959-03-26	[null]	[null]



**2) INSERT command:** INSERT is another most important data manipulation command in Structured Query Language, which allows users to insert data in database tables.

```
INSERT INTO <table-name> (<column1>, <column2>, ...)  
VALUES (<value1>, <value2>, ...) RETURNING *;
```

- Use the INSERT INTO clause with the table-name where you want to insert the data. If you want to insert data to all columns of a table, then specifying the list of columns is optional.
- If you want to insert data to some columns, then provide a list of comma-separated values after the VALUES clause.
- The RETURNING clause is optional which will return a list of all inserted values or the value of the specified column.



Now, the following INSERT statement will insert a single row in the person table.

```
INSERT INTO person VALUES (1, 'Annie', 'Smith', 'F', DATE
'1988-01-09', 'ani@email.com', 'Germany');
```



The screenshot shows a PostgreSQL database interface with the following components:

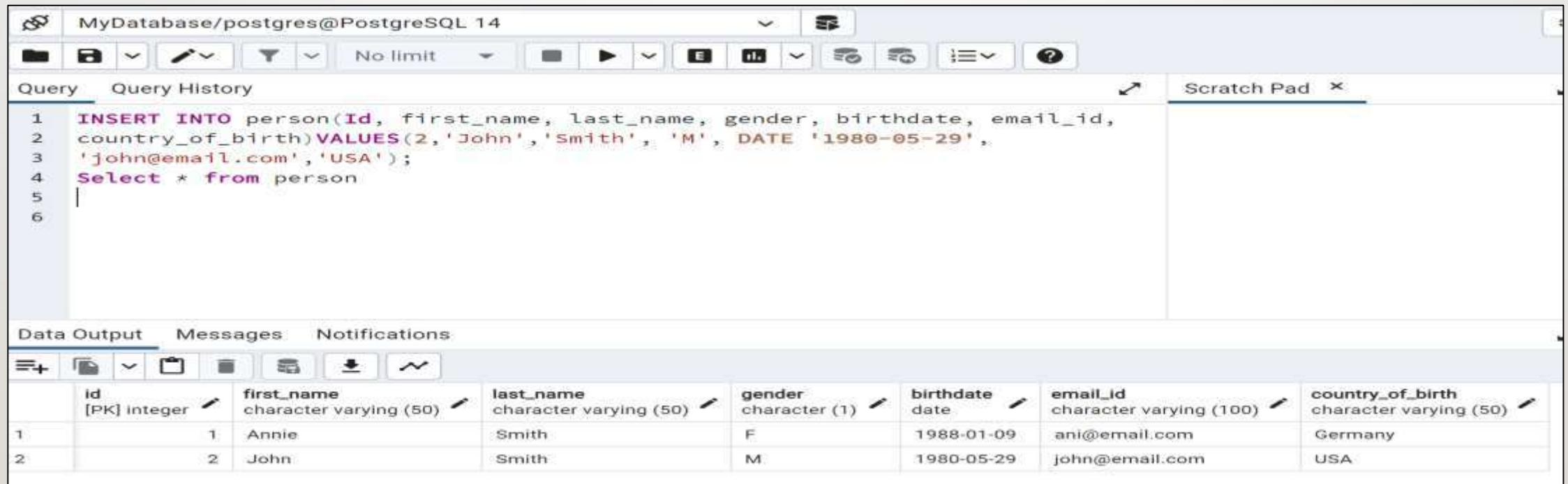
- Query Editor:** Contains the SQL statement:
 

```
1 INSERT INTO person VALUES (1, 'Annie', 'Smith', 'F', DATE '1988-01-09', 'ani@email.com', 'Germany')
2 select * from person
```
- Data Output:** Displays the result of the query as a table with 8 columns:
 

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	bod date	email_id text	country_of_birth character varying (50)
1	1	Annie	Smith	F	1988-01-09	ani@email.com	Germany

It is the best practice to specify columns names with the INSERT statement to insert data into correct columns and make it more maintainable. For example:

```
INSERT INTO person(Id, first_name, last_name, gender, birthdate,
email_id, country_of_birth) VALUES(2, John', 'Smith', 'M', '1980-05-29',
'john@email.com', 'USA');
```



The screenshot shows a PostgreSQL database interface with the following components:

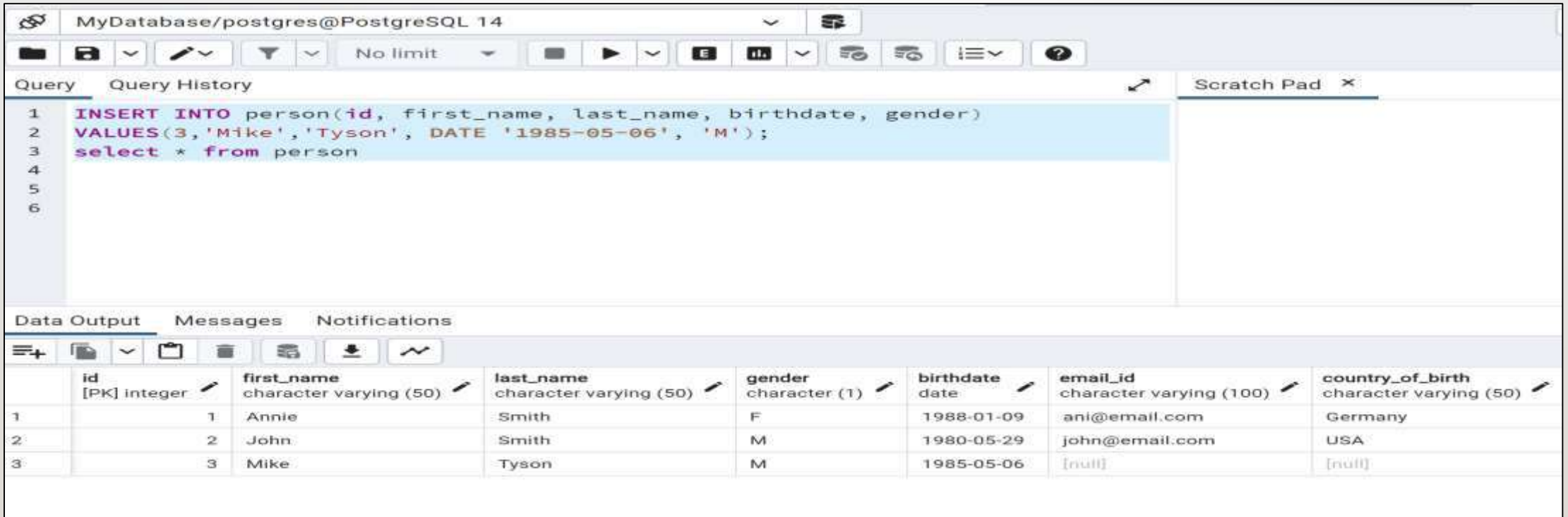
- Query Editor:** Contains the following SQL code:
 

```
1 INSERT INTO person(Id, first_name, last_name, gender, birthdate, email_id,
2 country_of_birth) VALUES(2, John', 'Smith', 'M', DATE '1980-05-29',
3 'john@email.com', 'USA');
4 Select * from person
5 |
6
```
- Data Output Table:** Displays the results of the query. The table has 8 columns: Id, first\_name, last\_name, gender, birthdate, email\_id, and country\_of\_birth. The data is as follows:
 

	Id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	1	Annie	Smith	F	1988-01-09	ani@email.com	Germany
2	2	John	Smith	M	1980-05-29	john@email.com	USA

We can change the order of the columns or remove columns from the INSERT statement as per your need.

```
INSERT INTO person (id, first_name, last_name, birthdate, gender)
VALUES (3, 'Mike', 'Tyson', DATE '1985-05-06', 'M');
```



The screenshot shows a PostgreSQL database interface with the following components:

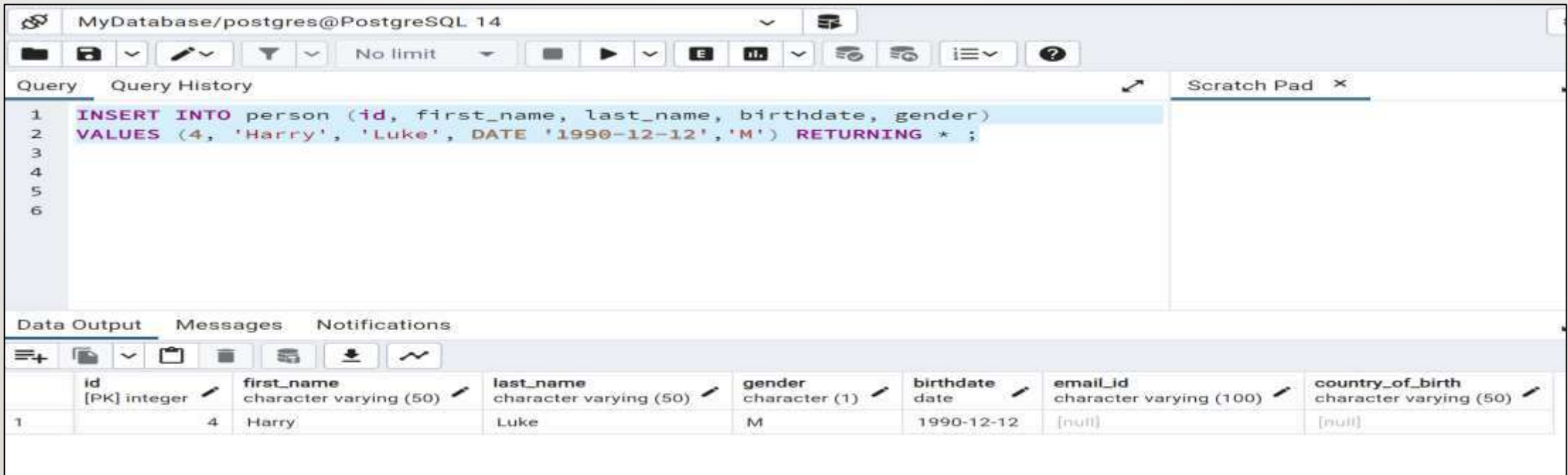
- Query Editor:** Contains the SQL query:
 

```
1 INSERT INTO person(id, first_name, last_name, birthdate, gender)
2 VALUES(3, 'Mike', 'Tyson', DATE '1985-05-06', 'M');
3 select * from person
4
5
6
```
- Data Output Table:** Displays the results of the query.
 

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	1	Annie	Smith	F	1988-01-09	ani@email.com	Germany
2	2	John	Smith	M	1980-05-29	john@email.com	USA
3	3	Mike	Tyson	M	1985-05-06	[null]	[null]

The RETURNING clause returns inserted column values. The RETURNING \* returns all the inserted values or RETURNING column-name returns the specified column value.

```
INSERT INTO person (id, first_name, last_name, birthdate, gender)
VALUES (4, 'Harry', 'Luke', DATE '1990-12-12', 'M') RETURNING *;
```



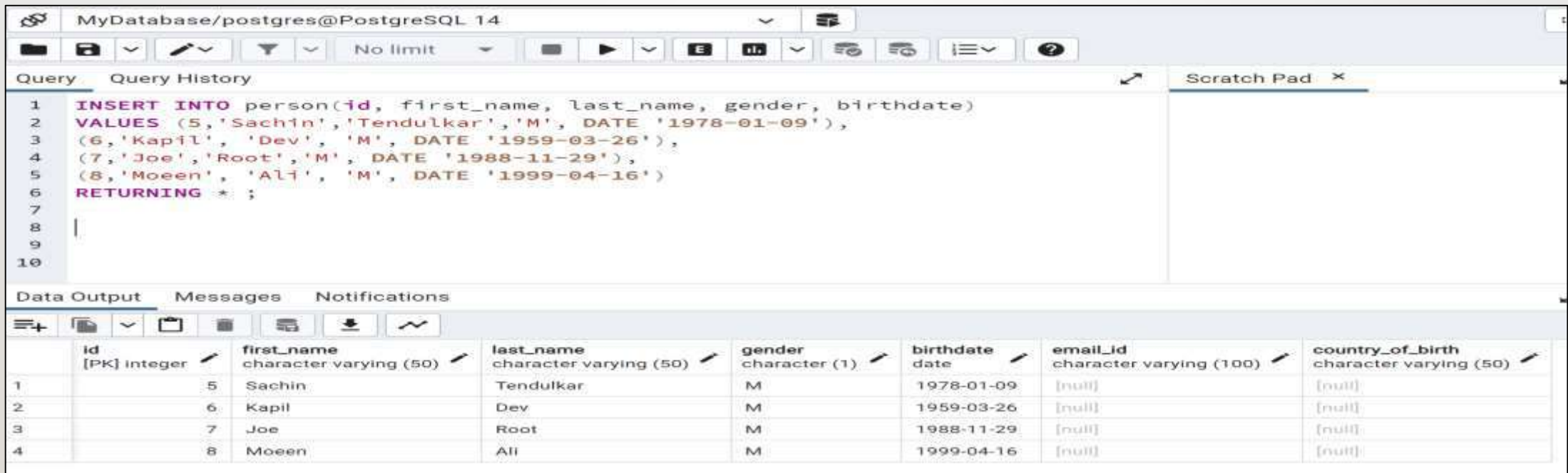
The screenshot shows a PostgreSQL client interface with the following components:

- Query Editor:** Contains the SQL statement: `INSERT INTO person (id, first_name, last_name, birthdate, gender) VALUES (4, 'Harry', 'Luke', DATE '1990-12-12', 'M') RETURNING * ;`
- Data Output Tab:** Displays the result of the query as a table with 8 columns: `id`, `first_name`, `last_name`, `gender`, `birthdate`, `email_id`, and `country_of_birth`. The data row shows: `4`, `Harry`, `Luke`, `M`, `1990-12-12`, `[null]`, and `[null]`.

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	4	Harry	Luke	M	1990-12-12	[null]	[null]

The INSERT statement can also add multiple rows to a table in a single query by specifying multiple VALUES clauses.

```
INSERT INTO person(id, first_name, last_name, gender, birthdate) VALUES
(5, 'Sachin', 'Tendulkar', 'M', DATE '1978-01-09'), (6, 'Kapil', 'Dev', 'M', DATE '1959-03-26'), (7, 'Joe', 'Root', 'M', DATE '1988-11-29'), (8, 'Moeen', 'Ali', 'M', DATE '1999-04-16');
```



The screenshot shows a PostgreSQL query editor with the following SQL query:

```
1 INSERT INTO person(id, first_name, last_name, gender, birthdate)
2 VALUES (5, 'Sachin', 'Tendulkar', 'M', DATE '1978-01-09'),
3 (6, 'Kapil', 'Dev', 'M', DATE '1959-03-26'),
4 (7, 'Joe', 'Root', 'M', DATE '1988-11-29'),
5 (8, 'Moeen', 'Ali', 'M', DATE '1999-04-16')
6 RETURNING *;
```

The "Data Output" tab shows the following table:

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	5	Sachin	Tendulkar	M	1978-01-09	[null]	[null]
2	6	Kapil	Dev	M	1959-03-26	[null]	[null]
3	7	Joe	Root	M	1988-11-29	[null]	[null]
4	8	Moeen	Ali	M	1999-04-16	[null]	[null]



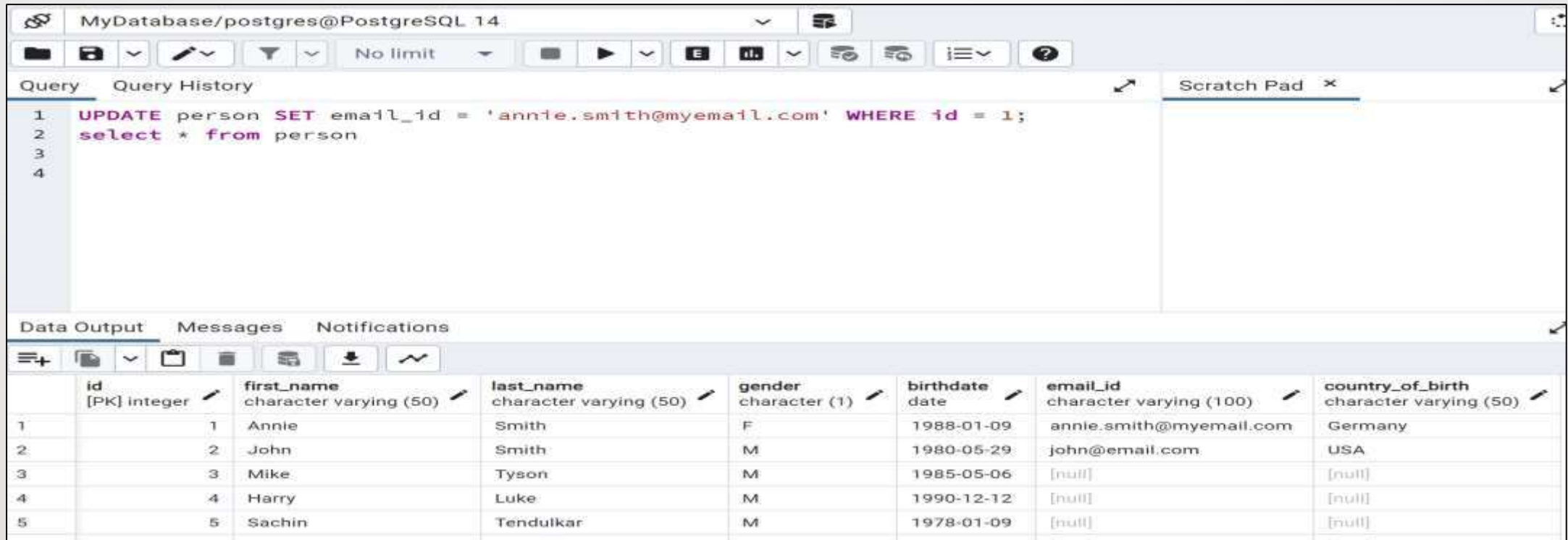
**3) UPDATE command:** This DML command allows the database users to change the existing record or rows in the tables.

```
UPDATE <table_name> SET <column1> = <value1>, <column2>
= <value2>, ... WHERE <condition>
```

Here, 'UPDATE', 'SET', and 'WHERE' are the SQL keywords, and 'Table\_name' is the name of the table whose values we want to update.

For example, the following UPDATE statement will update an email of a person whose id=1.

```
UPDATE person SET email_id = 'annie.smith@myemail.com'
WHERE id = 1;
```



The screenshot shows a PostgreSQL query editor with the following SQL statement:

```
1 UPDATE person SET email_id = 'annie.smith@myemail.com' WHERE id = 1;
2 select * from person
3
4
```

The results are displayed in a table with the following columns: id, first\_name, last\_name, gender, birthdate, email\_id, and country\_of\_birth.

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	1	Annie	Smith	F	1988-01-09	annie.smith@myemail.com	Germany
2	2	John	Smith	M	1980-05-29	john@email.com	USA
3	3	Mike	Tyson	M	1985-05-06	[null]	[null]
4	4	Harry	Luke	M	1990-12-12	[null]	[null]
5	5	Sachin	Tendulkar	M	1978-01-09	[null]	[null]

If you don't specify the WHERE clause, then it will update the email column value in all the rows.

```
UPDATE person SET email_id = 'annie.smith@myemail.com';
```

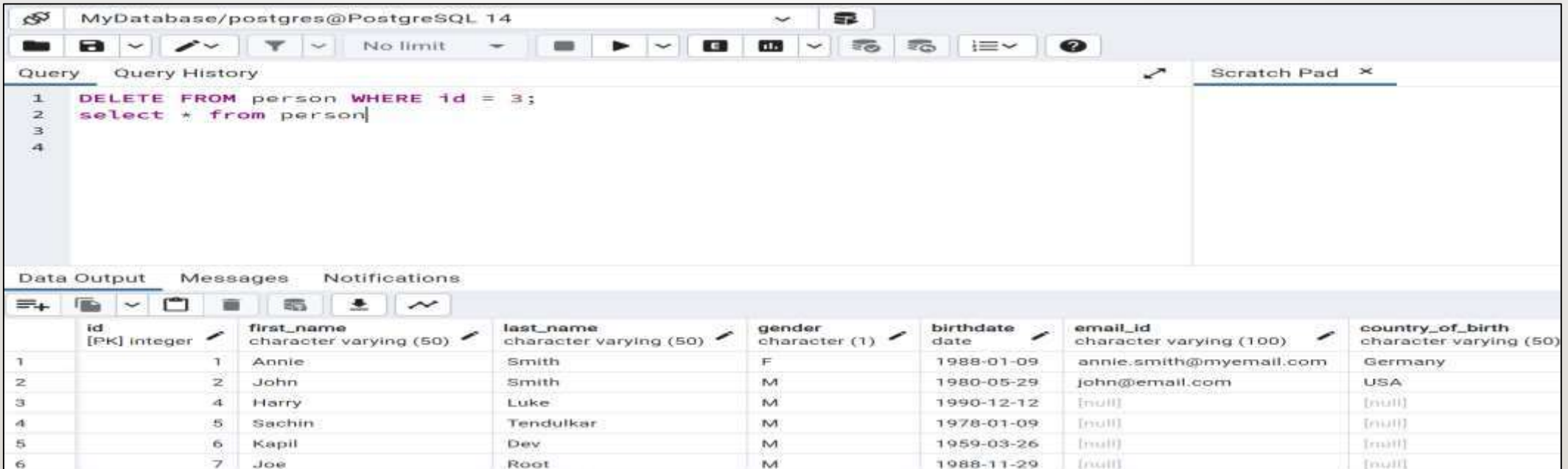


**3) DELETE command:** DELETE is a DML command which allows SQL users to remove single or multiple existing records from the database tables.

`DELETE FROM Table_Name WHERE condition;`

let's remove data from the person table whose id is 3.

`DELETE FROM person WHERE id = 3;`



The screenshot shows a PostgreSQL query editor window titled "MyDatabase/postgres@PostgreSQL 14". The query editor displays the following SQL commands:

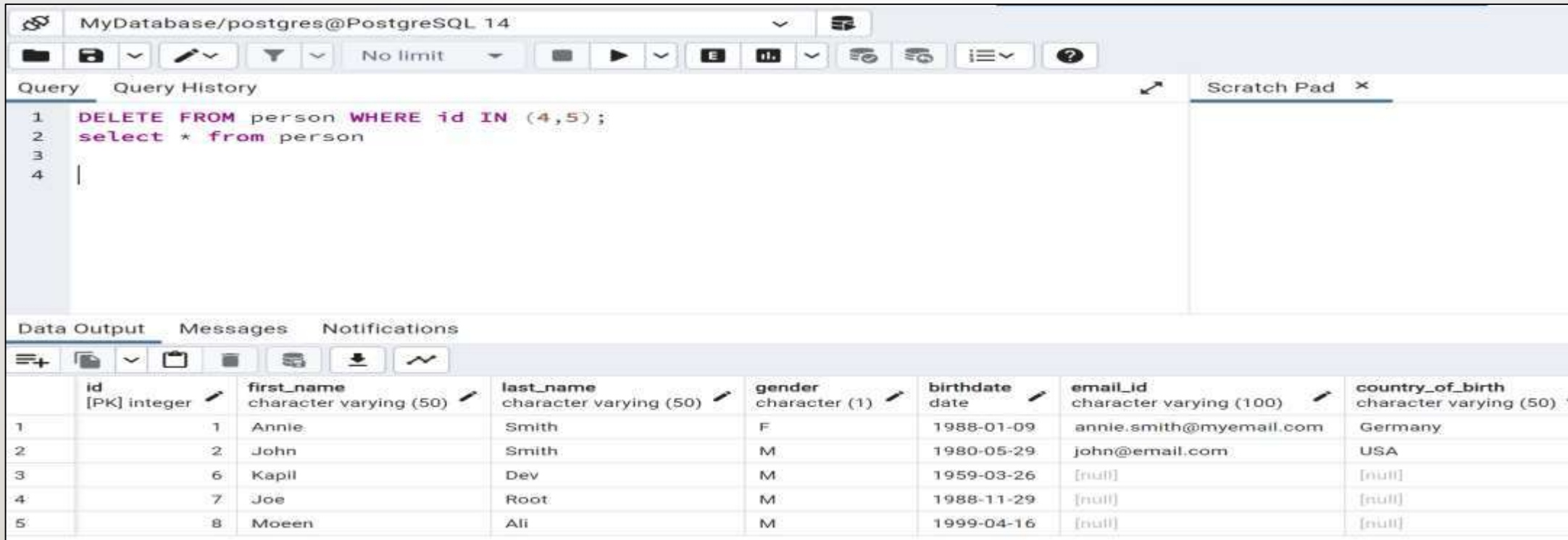
```
1 DELETE FROM person WHERE id = 3;
2 select * from person
3
4
```

Below the query editor, the "Data Output" tab is active, showing the result of the query. The table has 8 columns: id, first\_name, last\_name, gender, birthdate, email\_id, and country\_of\_birth. The data is as follows:

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	1	Annie	Smith	F	1988-01-09	annie.smith@myemail.com	Germany
2	2	John	Smith	M	1980-05-29	john@email.com	USA
3	4	Harry	Luke	M	1990-12-12	[null]	[null]
4	5	Sachin	Tendulkar	M	1978-01-09	[null]	[null]
5	6	Kapil	Dev	M	1959-03-26	[null]	[null]
6	7	Joe	Root	M	1988-11-29	[null]	[null]

The DELETE statement can remove multiple records from the person table. For example, now we will remove rows where id = 4 and 5.

```
DELETE FROM person WHERE id IN (4,5);
```



The screenshot shows a PostgreSQL database interface with the following components:

- Query Editor:** Contains the SQL query:
 

```
1 DELETE FROM person WHERE id IN (4,5);
2 select * from person
3
4 |
```
- Data Output:** Displays the results of the query in a table format.
 

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	birthdate date	email_id character varying (100)	country_of_birth character varying (50)
1	1	Annie	Smith	F	1988-01-09	annie.smith@myemail.com	Germany
2	2	John	Smith	M	1980-05-29	john@email.com	USA
3	6	Kapil	Dev	M	1959-03-26	[null]	[null]
4	7	Joe	Root	M	1988-11-29	[null]	[null]
5	8	Moeen	Ali	M	1999-04-16	[null]	[null]

## 3) Data Control Language (DCL)

Data Control Language allows DBA to manage the rights and permissions on the data in the database. Following are the two DCL Languages or commands used in the SQL queries:

1. Grant DCL Command
2. Revoke DCL Command

**A. Grant:** It is used to give user access privileges to a database.

**GRANT** privileges **ON** object **TO** user;

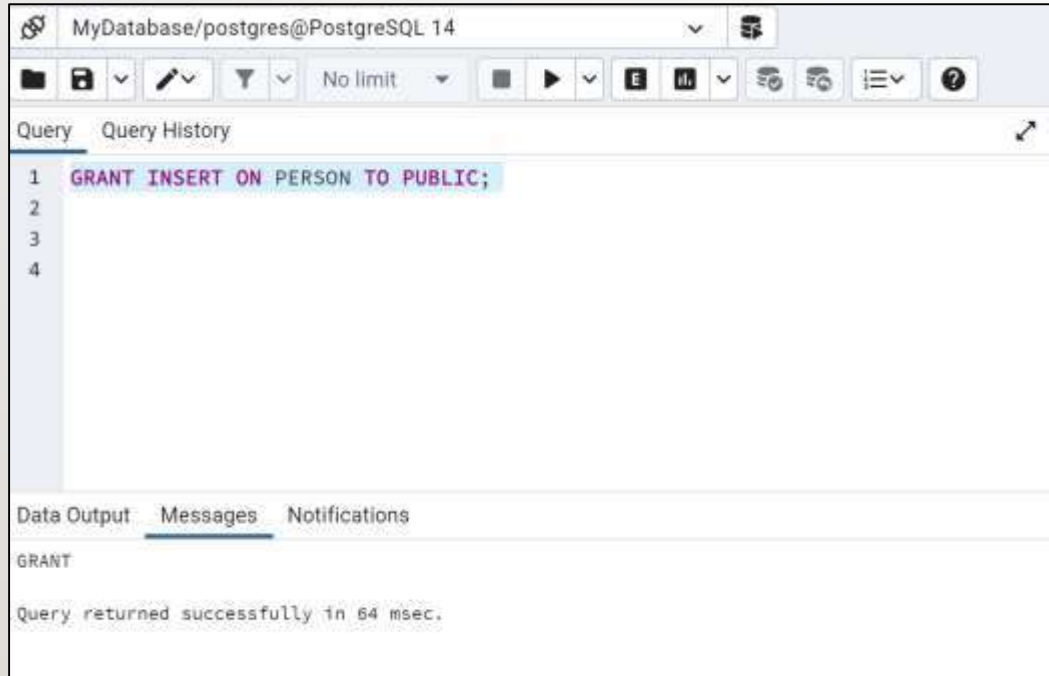
Privileges can be SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, TRIGGER, CREATE, ALL. You can also specify combination of these privileges in a statement.

Grant insert privilege to all users on table person:

```
GRANT INSERT ON person TO PUBLIC;
```

Grant all available privileges to all users on table person:

```
GRANT ALL PRIVILEGES ON person TO PUBLIC;
```



MyDatabase/postgres@PostgreSQL 14

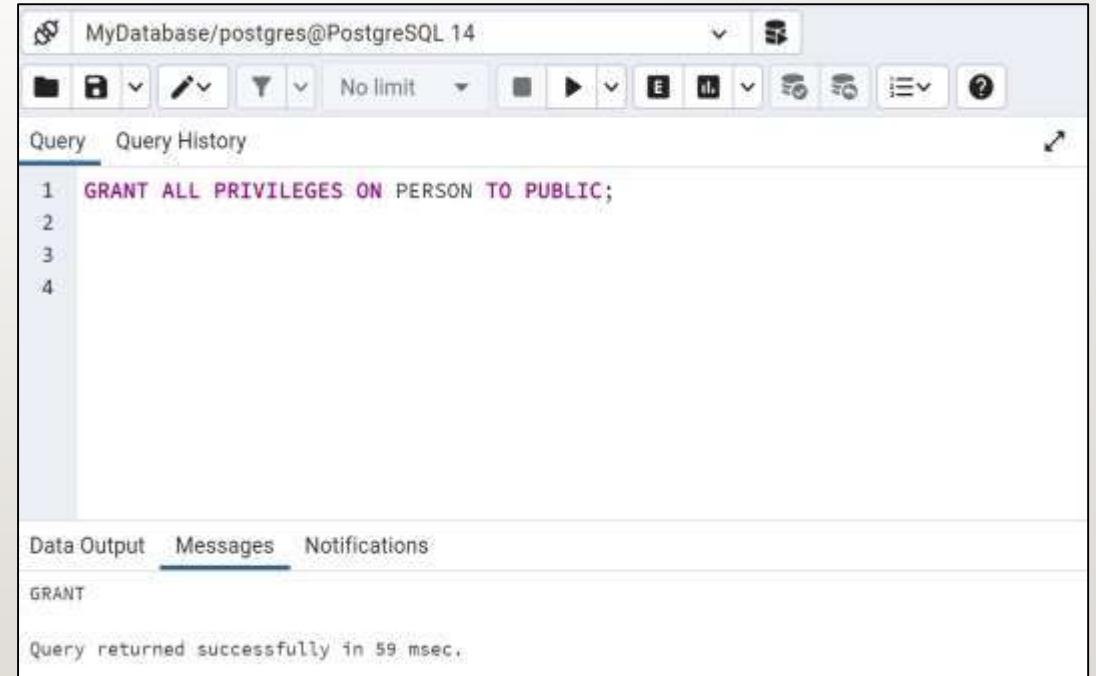
Query Query History

```
1 GRANT INSERT ON PERSON TO PUBLIC;
```

Data Output Messages Notifications

GRANT

Query returned successfully in 64 msec.



MyDatabase/postgres@PostgreSQL 14

Query Query History

```
1 GRANT ALL PRIVILEGES ON PERSON TO PUBLIC;
```

Data Output Messages Notifications

GRANT

Query returned successfully in 59 msec.

**B. Revoke:** This DCL command allows the database administrator to remove all the permissions applied by the GRANT DCL command.

**REVOKE** PRIVILEGES **ON** object **FROM** USER;

For example, to revoke the already granted insert privilege to all users on table person:

**REVOKE** INSERT ON PERSON **FROM** PUBLIC;



## Transaction Control Language (TCL)

TCL commands can only be used with DML commands like INSERT, DELETE and UPDATE only. Following are the two TCL commands:

**A. Commit:** This command allows the database users to save the operations in the database.

```
DELETE FROM Student WHERE AGE = 25;
```

```
COMMIT;
```

**B. Rollback:** Rollback command allows you to undo transactions that have not already been saved to the database.

```
DELETE FROM Student WHERE AGE = 25;
```

```
ROLLBACK;
```



# CREATE TABLE COMMAND

This DDL command allows us to create the new table.

```
CREATE TABLE [IF NOT EXISTS] <table_name> ( <column1> <data_type(length)>
[column_constraint], <column2> <data_type(length)> [column_constraint], ... <columnN>
<data_type(length)> [column_constraint], [table_constraints] );
```

```
CREATE TABLE IF NOT EXISTS person ( Id INT PRIMARY KEY, first_name VARCHAR(50) NOT NULL, last_name
VARCHAR(50) NOT NULL, gender CHAR(1), birthdate DATE, email_id VARCHAR(100) UNIQUE, country_of_birth
VARCHAR(50) );
```



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is to 'MyDatabase/postgres@PostgreSQL 14'. The 'Query' tab is active, displaying the following SQL command:

```
1 CREATE TABLE IF NOT EXISTS person ( Id INT PRIMARY KEY,
2 first_name VARCHAR(50) NOT NULL,
3 last_name VARCHAR(50) NOT NULL,
4 gender CHAR(1),
5 birthdate DATE,
6 email_id VARCHAR(100) UNIQUE,
7 country_of_birth VARCHAR(50) );
8
```

Below the query editor, the 'Messages' tab is selected, showing the output: 'CREATE TABLE' and 'Query returned successfully in 68 msec.'



## CREATE TABLE COMMAND (Cont..)

**Create table as command:** Use this statement to create a new table with data from the specified SELECT query.

```
CREATE TABLE [IF NOT EXISTS] <table_name> [(<column1>, <column2>, ...)] AS
<Query>;
```

The following command will create the Employee table with the same column names and data types as SELECT query that selects data from the person table.

```
CREATE TABLE employee AS SELECT * FROM person;
```



MyDatabase/postgres@PostgreSQL 14

Query Query History

```
1 create table employee as select * from person
2 select * from employee
```

Scratch Pad

Data Output Messages Notifications

	id	first_name	last_name	gender	birthdate	email_id	country_of_birth
	integer	character varying (50)	character varying (50)	character (1)	date	character varying (100)	character varying (50)
1	6	Kapil	Dev.	M	1959-03-26	[null]	[null]
2	7	Joe	Root	M	1988-11-29	[null]	[null]
3	8	Moeen	All	M	1999-04-16	[null]	[null]
4	1	Annie	Smith	F	1988-01-09	annie.smith@myemail.com	Germany

## CREATE TABLE COMMAND (Cont..)

If you want to create a table with a selected column list then, you can select only required columns in the SELECT clause.

```
CREATE TABLE employee AS SELECT ID, FIRST_NAME, LAST_NAME FROM person;
```

If you want column names to be different from SELECT query columns you can specify new columns list.

```
CREATE TABLE employee1(Id, FirstName, LastName, Gender) AS SELECT ID, FIRST_NAME, LAST_NAME, GENDER FROM person;
```



The screenshot shows a PostgreSQL query editor interface. The title bar indicates the connection is 'MyDatabase/postgres@PostgreSQL 14'. The 'Query' tab is active, displaying the following SQL script:

```
1 CREATE TABLE employee1(Id, FirstName, LastName, Gender)
2 AS SELECT ID, FIRST_NAME, LAST_NAME, GENDER FROM person;
3 select * from employee1
4 |
```

Below the query editor, the 'Data Output' tab is active, showing the results of the query. The output is a table with 4 rows and 4 columns: 'id', 'firstname', 'lastname', and 'gender'. The data is as follows:

	id integer	firstname character varying (50)	lastname character varying (50)	gender character (1)
1	6	Kapil	Dev	M
2	7	Joe	Root	M
3	8	Moeen	Ali	M
4	1	Annie	Smith	F

## CREATE TABLE COMMAND (Cont..)

**Define NOT NULL Constraint while Creating a Table:** The following command declares NOT NULL columns in the CREATE TABLE statement. It will create the employee table with NOT NULL constraints on the first\_name and last\_name columns. If a column has a NOT NULL constraint defined on it then any attempt to insert or update the NULL value to that column will not be allowed and will raise an error.

```
CREATE TABLE employee( emp_id INT, first_name VARCHAR(50) NOT NULL, last_name  
VARCHAR(50) NOT NULL, gender CHAR(1), birthdate DATE, email VARCHAR(100),  
salary INT);
```



The screenshot shows a PostgreSQL database interface with the following components:

- Database:** MyDatabase/postgres@PostgreSQL 14
- Query Editor:** Contains the SQL command:

```
1 CREATE TABLE employee( emp_id INT,  
2 first_name VARCHAR(50) NOT NULL,  
3 last_name VARCHAR(50) NOT NULL,  
4 gender CHAR(1),  
5 birthdate DATE,  
6 email VARCHAR(100),  
7 salary INT);  
8 select * from employee  
9
```
- Data Output:** Displays the table structure with columns and their data types:

emp_id	first_name	last_name	gender	birthdate	email	salary
integer	character varying (50)	character varying (50)	character (1)	date	character varying (100)	integer

## CREATE TABLE COMMAND (Cont..)

**Define UNIQUE Constraint while Creating a Table:** Unique constraints can be defined at the table level or at the column level, as shown below. The following command creates a unique constraint on the email column of the employee table.

```
CREATE TABLE IF NOT EXISTS employee (emp_id INT, first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50), gender CHAR(1), birthdate DATE, email VARCHAR(100) UNIQUE,
salary INT);
```



The screenshot shows a PostgreSQL query editor interface. The query window displays the following SQL command:

```
1 CREATE TABLE IF NOT EXISTS employee (emp_id INT,
2 first_name VARCHAR(50) NOT NULL,
3 last_name VARCHAR(50),
4 gender CHAR(1),
5 birthdate DATE,
6 email VARCHAR(100) UNIQUE,
7 salary INT);
8 select * from employee
9
```

The bottom of the interface shows the 'Data Output' tab, which displays the table structure for the 'employee' table:

emp_id	first_name	last_name	gender	birthdate	email	salary
integer	character varying (50)	character varying (50)	character (1)	date	character varying (100)	integer



## CREATE TABLE COMMAND (Cont..)

When a unique constraint is defined on a group of columns, then the combination of those column values needs to be unique across the table. A Unique constraint can be defined on multiple columns by specifying it at table level.

```
CREATE TABLE IF NOT EXISTS employee (emp_id INT, first_name VARCHAR(50) NOT NULL,  
last_name VARCHAR(50), gender CHAR(1), birthdate DATE, email VARCHAR(100), salary INT,  
UNIQUE (first_name, last_name));
```



MyDatabase/postgres@PostgreSQL 14

Query Query History Scratch Pad

```
1 CREATE TABLE IF NOT EXISTS employee (emp_id INT,  
2 first_name VARCHAR(50) NOT NULL,  
3 last_name VARCHAR(50),  
4 gender CHAR(1),  
5 birthdate DATE,  
6 email VARCHAR(100),  
7 salary INT,  
8 UNIQUE (first_name, last_name));  
9  
10 select * from employee  
11
```

Data Output Messages Graph Visualiser Notifications

emp_id	first_name	last_name	gender	birthdate	email	salary
integer	character varying (50)	character varying (50)	character (1)	date	character varying (100)	integer

**Define PRIMARY KEY Constraint while Creating a Table:** you can define a primary key on a single column by writing "primary key" after the column name in the CREATE TABLE statement. For example, the following CREATE TABLE statement will create the employee table with a primary key defined on emp\_id column. This is called defining a primary key at the column level.

```
CREATE TABLE IF NOT EXISTS employee (emp_id INT PRIMARY KEY, first_name  
VARCHAR(50), last_name VARCHAR(50), gender CHAR(1), birthdate DATE, email  
VARCHAR (100), salary INT);
```

The Primary key can be defined on more than one column also. For example, the following command defines a primary key on two columns:

```
CREATE TABLE IF NOT EXISTS employee_department (emp_id INT, dept_id  
INT, CONSTRAINT empid_deptid_pkey PRIMARY KEY (emp_id, dept_id));
```

# CREATE TABLE COMMAND (Cont..)

MyDatabase/postgres@PostgreSQL 14

Query Query History Scratch Pad

```

1 CREATE TABLE IF NOT EXISTS employee_department
2 (emp_id INT,
3  dept_id INT,
4  CONSTRAINT empid_deptid_pkey PRIMARY KEY (emp_id, dept_id));
5
6
7 select * from employee
8 |
  
```

Data Output Messages Graph Visualiser Notifications

emp_id	first_name	last_name	gender	birthdate	email	salary
integer	character varying (50)	character varying (50)	character (1)	date	character varying (100)	integer



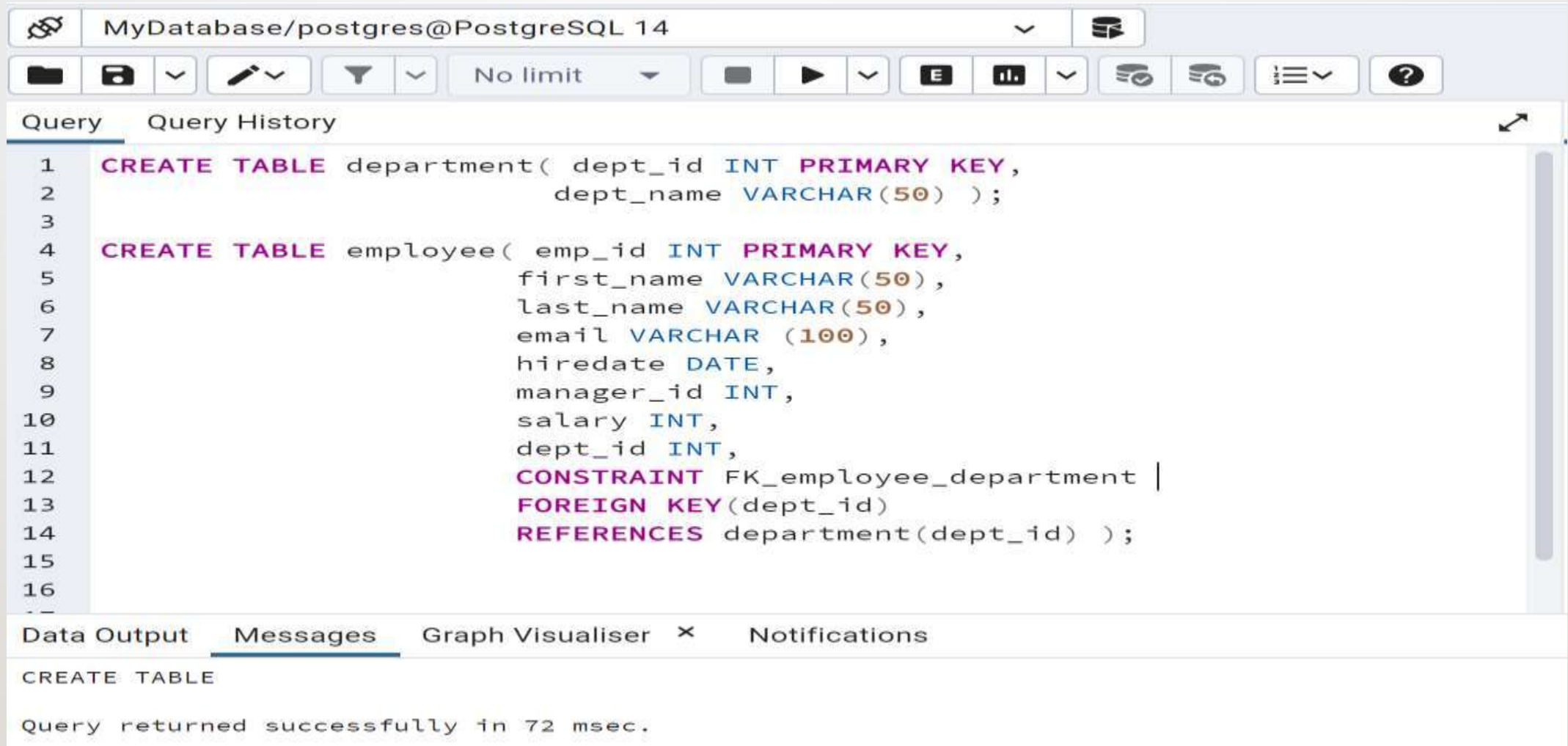
## CREATE TABLE COMMAND (Cont..)

**Define FOREIGN KEY Constraint while Creating a Table:** The foreign key is a column(s) in a table that points to a primary key or unique key column in the same or another table. You can define a foreign key when you create a table using CREATE TABLE statement. The following example demonstrates creating a foreign key in the employee table that points to the department table.

```
CREATE TABLE department( dept_id INT PRIMARY KEY, dept_name
VARCHAR(50) );

CREATE TABLE employee( emp_id INT PRIMARY KEY, first_name
VARCHAR(50), last_name VARCHAR(50), email VARCHAR (100), hiredate
DATE, manager_id INT, salary INT, dept_id INT,
CONSTRAINT FK_employee_department FOREIGN KEY(dept_id) REFERENCES
department(dept_id) );
```

# CREATE TABLE COMMAND (Cont..)



The screenshot shows a PostgreSQL query editor window titled "MyDatabase/postgres@PostgreSQL 14". The interface includes a toolbar with icons for file operations, query execution, and settings. The "Query" tab is active, displaying the following SQL code:

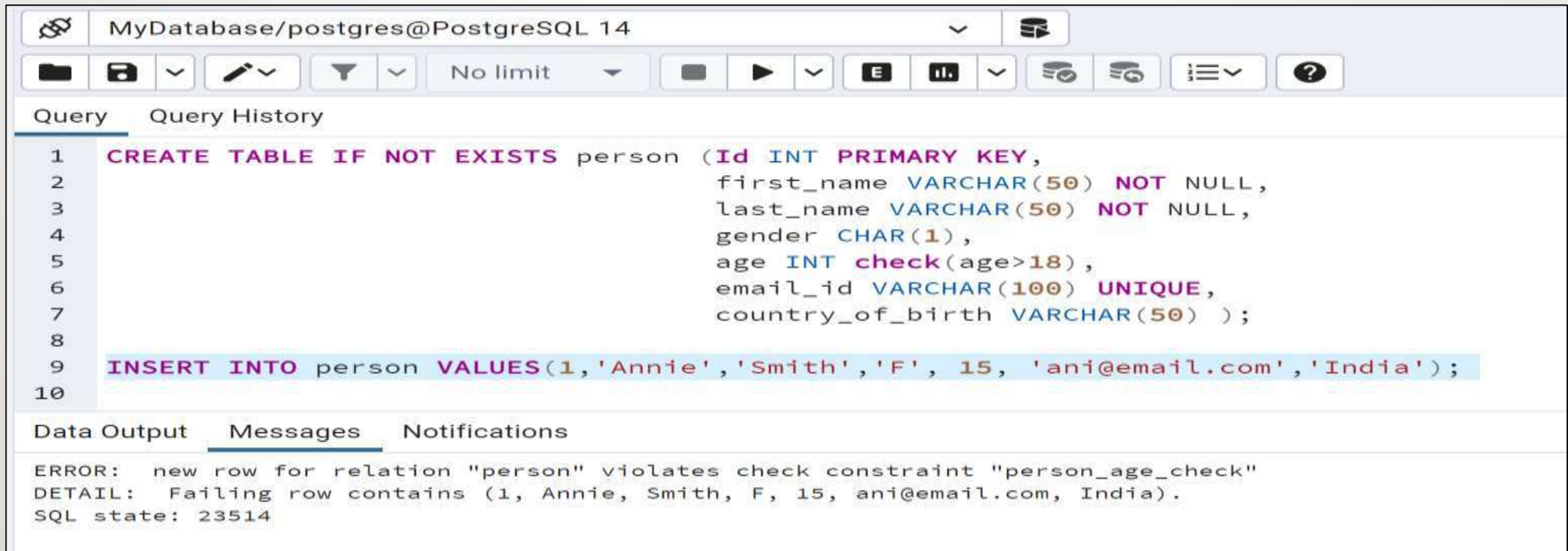
```
1 CREATE TABLE department( dept_id INT PRIMARY KEY,
2     dept_name VARCHAR(50) );
3
4 CREATE TABLE employee( emp_id INT PRIMARY KEY,
5     first_name VARCHAR(50),
6     last_name VARCHAR(50),
7     email VARCHAR (100),
8     hiredate DATE,
9     manager_id INT,
10    salary INT,
11    dept_id INT,
12    CONSTRAINT FK_employee_department |
13    FOREIGN KEY(dept_id)
14    REFERENCES department(dept_id) );
15
16
```

Below the query editor, the "Messages" tab is active, showing the following output:

```
CREATE TABLE
Query returned successfully in 72 msec.
```

## CREATE TABLE COMMAND (Cont..)

**Define CHECK Constraint while Creating a Table:** The CHECK constraint is used to limit the value range that can be placed in a column. The following SQL creates a CHECK constraint on the age column when the "Person" table is created. The CHECK constraint ensures that the age of a person must be greater than 18:



```
MyDatabase/postgres@PostgreSQL 14

Query  Query History

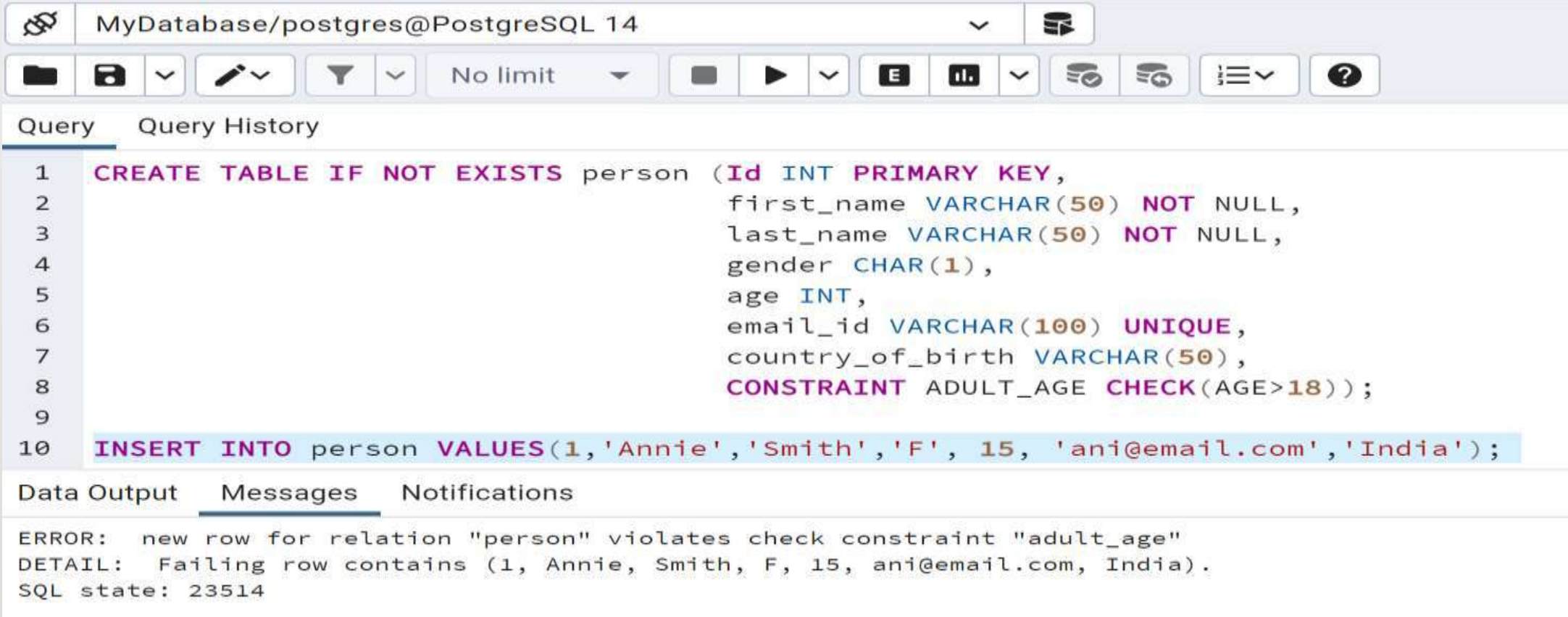
1  CREATE TABLE IF NOT EXISTS person (Id INT PRIMARY KEY,
2                                     first_name VARCHAR(50) NOT NULL,
3                                     last_name VARCHAR(50) NOT NULL,
4                                     gender CHAR(1),
5                                     age INT check(age>18),
6                                     email_id VARCHAR(100) UNIQUE,
7                                     country_of_birth VARCHAR(50) );
8
9  INSERT INTO person VALUES(1,'Annie','Smith','F', 15, 'ani@email.com','India');
10

Data Output  Messages  Notifications

ERROR:  new row for relation "person" violates check constraint "person_age_check"
DETAIL:  Failing row contains (1, Annie, Smith, F, 15, ani@email.com, India).
SQL state: 23514
```

## CREATE TABLE COMMAND (Cont..)

You can also give the constraint a separate name. This clarifies error messages and allows you to refer to the constraint when you need to change it. For example:



The screenshot shows a PostgreSQL query editor interface. The top bar displays the connection 'MyDatabase/postgres@PostgreSQL 14'. Below the toolbar, the 'Query' tab is active, showing a SQL script. The script defines a 'person' table with columns: 'Id' (INT PRIMARY KEY), 'first\_name' (VARCHAR(50) NOT NULL), 'last\_name' (VARCHAR(50) NOT NULL), 'gender' (CHAR(1)), 'age' (INT), 'email\_id' (VARCHAR(100) UNIQUE), 'country\_of\_birth' (VARCHAR(50)), and a check constraint named 'ADULT\_AGE' (CHECK(AGE > 18)). An 'INSERT INTO' statement follows, attempting to add a row with age 15. The 'Messages' tab at the bottom shows an error: 'ERROR: new row for relation "person" violates check constraint "adult\_age"'.

```
1 CREATE TABLE IF NOT EXISTS person (Id INT PRIMARY KEY,  
2 first_name VARCHAR(50) NOT NULL,  
3 last_name VARCHAR(50) NOT NULL,  
4 gender CHAR(1),  
5 age INT,  
6 email_id VARCHAR(100) UNIQUE,  
7 country_of_birth VARCHAR(50),  
8 CONSTRAINT ADULT_AGE CHECK(AGE>18));  
9  
10 INSERT INTO person VALUES(1,'Annie','Smith','F', 15, 'ani@email.com','India');
```

ERROR: new row for relation "person" violates check constraint "adult\_age"  
DETAIL: Failing row contains (1, Annie, Smith, F, 15, ani@email.com, India).  
SQL state: 23514



## IMPORTANT FACTS RELATED TO THE SESSION

- Structured Query Language(SQL) is the widely used database language for almost all types of relational databases, by the use of which we can perform certain operations on the existing database and also we can use this language to create a new database.
- Data Definition Language can be used to define the database structure or schema.
- Data Manipulation language can be used to access, store, modify, update and delete the existing records from the database.

## SUMMARY

1. In this section, we discussed the various types of SQL languages and commands of these languages that are used to work on the database and the data stored in the databases.
2. We also discussed the CREATE TABLE command in detail with its syntax and examples for creating the tables.

## SELF-ASSESSMENT QUESTIONS

**1. Commands that comes under DDL is/are –**

- (a) CREATE
- (b) DROP
- (c) TRUCATE
- (d) ALLOF THE ABOVE

**2. Command that comes under DML is/are –**

- (a) ROLLBACK
- (b) GRANT
- (c) UPDATE
- (d) ALL OF THE ABOVE



## SELF-ASSESSMENT QUESTIONS

**3. Command that comes under DCL is/are -**

- (a) GRANT
- (b) REVOKE
- (c) BOTH (a) AND (b)
- (d) NONE OF THE ABOVE

**4. Following the completion of a transaction, it must be executed to save all the operations performed in the transaction. Here we are talking about which command?**

- (a) REVOKE
- (b) COMMIT
- (c) ROLLBACK
- (d) SAVE

1. Describe various types of SQL Languages.
2. List out the commands of Data Definition Language with examples.
3. Analyze the DDL commands in PostgreSQL.
4. Summarize the create table command with its syntax and examples.

## Reference Books:

1. Database System Concepts, Sixth Edition, Abraham Silberschatz, Yale University Henry, F. Korth Lehigh University, S. Sudarshan Indian Institute of Technology, Bombay.
2. An Introduction to Database Systems by Bipin C. Desai
3. Fundamentals of Database Systems, 7<sup>th</sup> Edition, RamezElmasri, University of Texas at Arlington, Shamkant B. Navathe, University of Texasat Arlington.

## Sites and Web links:

1. <https://www.geeksforgeeks.org/postgresql-create-table/>
2. <https://www.tutorialsteacher.com/postgresql>

THANK YOU



Team – DBMS