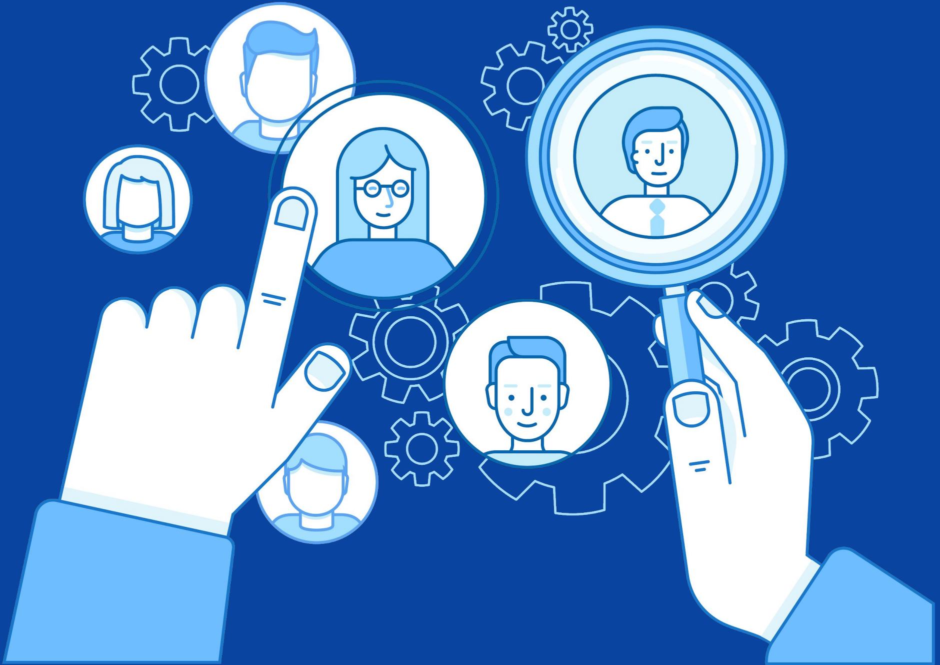
Digital Career Institute

Python Course - Database - Basic Usage



Digital Career Institute

SQL Categories & Commands

Digital Career Institute

DDL	CREATE DATABASE, DROP DATABASE, CREATE TABLE, ALTER TABLE, DROP TABLE	**DQL**	SELECT
DML	INSERT, UPDATE, DELETE, TRUNCATE	**DCL**	GRANT, REVOKE

Data Definition Language

DDL Commands

Digital Career Institute

The most common DDL commands are used to:

- `CREATE` databases and tables.
- `ALTER` the `TABLE` definition.
- `DROP` databases and tables.

Create a Database

```sql
CREATE DATABASE personal;
```

```
                                    List of databases
        Name                 |   Owner    | Encoding |   Collate    |    Ctype     |         Access privileges
-----------------------------+------------+----------+--------------+--------------+-----------------------------------
 DCI                         | postgres   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 uber_eats                   | postgres   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 course_project              | postgres   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 my_notes                    | postgres   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 personal                    | postgres   | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
```

Connect to a Database

```
postgres=# \c personal
```

The server may hold multiple databases.

Two tables with the same name can be defined in two different databases.

To know which of the two tables is being accessed, an active connection to its database must be established before.

Connecting to a database is one of the few operations that cannot be done with SQL in PostgreSQL.

Create a Schema

```sql
CREATE SCHEMA private;
```

```
personal=# \dn
  List of schemas
  Name    |   Owner
---------+----------
 private | postgres
 public  | postgres
(2 rows)
```

Create a Table

```sql
CREATE TABLE private.friends (
  -- The columns will
  -- be defined here.
);
```

The most basic definition of a table consists of:

- a table name. May be preceded by the schema name. If not, the default schema is used.
- a list of columns, wrapped in parentheses.

Create a Table: Columns

```
CREATE TABLE private.friends (
  first_name   varchar(20),
  last_name    varchar(50),
  phone        varchar(12),
  age          integer
);
```

Column definitions must be separated using commas.

`varchar` indicates a character string of varying length. The length is indicated in parentheses.

Each column is defined with a name and a type, separated by a whitespace. The column name must not include whitespaces or special keywords or characters.

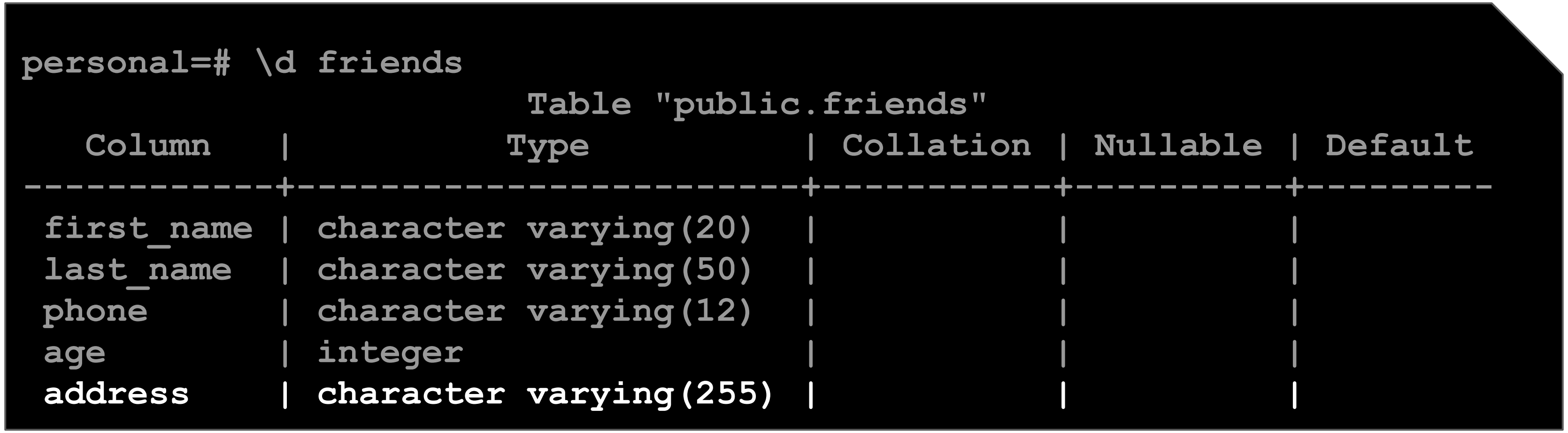
Create a Table: Proper Styling

```sql
CREATE TABLE private.friends(first_name varchar(20),last_name varchar(50));
```

```sql
CREATE TABLE private.friends (
  first_name     varchar(20),
  last_name      varchar(50),
);
```

Change a Table: Add a Column

```
ALTER TABLE friends
ADD [COLUMN] address varchar(255);
```



```
personal=# \d friends
                        Table "public.friends"
   Column    |          Type          | Collation | Nullable | Default
-------------+------------------------+-----------+----------+---------
 first_name  | character varying(20)  |           |          |
 last_name   | character varying(50)  |           |          |
 phone       | character varying(12)  |           |          |
 age         | integer                |           |          |
 address     | character varying(255) |           |          |
```

Change a Table: Rename a Column

```
ALTER TABLE friends
RENAME [COLUMN] address TO location;
```

```
personal=# \d friends
                      Table "public.friends"
   Column    |          Type          | Collation | Nullable | Default
-------------+------------------------+-----------+----------+---------
 first_name  | character varying(20)  |           |          |
 last_name   | character varying(50)  |           |          |
 phone       | character varying(12)  |           |          |
 age         | integer                |           |          |
 location    | character varying(255) |           |          |
```

Change a Table: Change a Column’s Type

```
ALTER TABLE friends
ALTER [COLUMN] location TYPE int;
```

```
personal=# ALTER TABLE friends ALTER location TYPE int;
ERROR:  column "location" cannot be cast automatically to type integer
HINT:  You might need to specify "USING location::integer".
```

Changing the type will require changing the type of the values that may be stored in that column.

Change a Table: Change a Column’s Type

```
ALTER TABLE friends
ALTER [COLUMN] location TYPE int
USING location::integer;
```

```
personal=# \d friends
                        Table "public.friends"
   Column    |          Type           | Collation | Nullable | Default
-------------+-------------------------+-----------+----------+---------
 first_name  | character varying(20)   |           |          |
 last_name   | character varying(50)   |           |          |
 phone       | character varying(12)   |           |          |
 age         | integer                 |           |          |
 location    | integer                 |           |          |
```

Change a Table: Remove a Column

```
ALTER TABLE friends
DROP [COLUMN] location;
```

```
personal=# \d friends
                        Table "public.friends"
   Column   |           Type           | Collation | Nullable | Default
------------+--------------------------+-----------+----------+---------
 first_name | character varying(20)    |           |          |
 last_name  | character varying(50)    |           |          |
 phone      | character varying(12)    |           |          |
 age        | integer                  |           |          |
```

Remove a Table

```
DROP TABLE friends;
```

Remove a Database

```
DROP DATABASE personal;
```

```
personal=# DROP DATABASE personal;
ERROR:  cannot drop the currently open database
personal=# \c postgres
postgres=# DROP DATABASE personal;
DROP DATABASE
```

Connecting to another database will release the lock on the database requiring deletion.

Remove Nonexistent Objects

```
ALTER TABLE friends DROP location;
DROP TABLE friends;
DROP DATABASE personal;
```

```
postgres=# ALTER TABLE friends DROP location;
ERROR:  column "location" of relation "friends" does not exist
postgres=# DROP TABLE friends;
ERROR:  table "friends" does not exist
postgres=# DROP DATABASE personal;
ERROR:  database "personal" does not exist
```

This is not a problem in this case, when using the statements once. But if this is part of a script, it will break the execution.

Remove Objects Only if they Exist

```
ALTER TABLE friends DROP IF EXISTS location;
DROP TABLE IF EXISTS friends;
DROP DATABASE IF EXISTS personal;
```

```
personal=# ALTER TABLE friends DROP IF EXISTS location;
NOTICE:  column "location" of relation "friends" does not exist, skipping
ALTER TABLE
personal=# DROP TABLE IF EXISTS friends;
NOTICE:  table "friends" does not exist, skipping
DROP TABLE
postgres=# DROP DATABASE IF EXISTS personal;
NOTICE:  database "personal" does not exist, skipping
DROP DATABASE
```

Data Manipulation Language

DML Commands

The most common DML commands are:

- **`INSERT`** to add data (DML).
- **`UPDATE`** to change data (DML).
- **`DELETE`** to remove rows of data (DML).
- **`TRUNCATE`** to clear the table (DML).

Add Rows

Digital Career Institute

Insert data in all fields.

```
INSERT INTO <table>
VALUES (<value1>, <value2>, <value3>, <value4>);
```

The values must be written in the same order as they were defined in the `CREATE TABLE` statement.

Add Rows

```
personal=# INSERT INTO friends
personal-# VALUES ('Lisa', 'Klepp', '916736453', 32);
INSERT 0 1
```

The values must be written in the same order as they were defined in the `CREATE TABLE` statement.

Add Rows

Insert data in some fields.

```
INSERT INTO <table>(<column2>, <column1>)
VALUES (<value2>, <value1>);
```

A different order may be specified in the first part of the statement.

If some fields allow NULL values, these can also be left out of the statement.

Add Rows

```
personal=# INSERT INTO friends(last_name, first_name)
personal-# VALUES ('Strum', 'Peter');
INSERT 0 1
```

The `phone` and `age` columns allow NULL values,
so we can skip them.

Add Rows

Insert multiple rows.

```
INSERT INTO <table>(<column2>, <column1>)
VALUES (<value2.1>, <value1.1>),
       (<value2.2>, <value1.2>);
```

Multiple rows can be inserted in one statement, by adding more data in the `VALUES` clause and separating them with commas.

Add Rows

Insert multiple rows.

```
personal=# INSERT INTO friends(last_name, first_name)
personal-# VALUES ('Strum', 'Peter'), ('Sullivan', 'Regina');
INSERT 0 2
```

The output of the insert statement will indicate how many rows have been inserted.

Update Data

Digital Career Institute

Update all rows.

```
UPDATE <table>
SET <column1> = <value1>, <column2> = <value2>;
```

The `UPDATE` command uses the `SET` clause to identify what data has to be changed.

Multiple columns can be updated at the same time, separating them with commas.

Update Data

Update all rows.

```
personal=# UPDATE friends SET age = 33;
UPDATE 2
personal=# SELECT * FROM friends;
 first_name | last_name |   phone    | age
------------+-----------+------------+-----
 Lisa       | Klepp     | 916736453 |  33
 Peter      | Strum     |            |  33
(2 rows)
```

Update Data

Digital Career Institute

Update only some rows.

```
UPDATE <table> SET <column1> = <new_value>
WHERE <condition>;
```

Just as with the **`SELECT`** command, the **`UPDATE`** also allows for row selection using the **`WHERE`** clause and a **`<condition>`**.

Update Data

Digital Career Institute

Update some rows.

```
personal=# UPDATE friends
personal=# SET phone = 923451762, first_name = 'Pete'
personal=# WHERE first_name = 'Peter';
UPDATE 1
personal=# SELECT * FROM friends;
 first_name | last_name |   phone    | age
------------+-----------+------------+-----
 Lisa       | Klepp     | 916736453 |  33
 Pete       | Strum     | 923451762 |  33
(2 rows)
```

Delete Data

Digital Career Institute

Delete all rows.

```
DELETE FROM <table>;
```

The `DELETE FROM` command removes rows from a table.

Delete Data

Clear table data.

```
TRUNCATE <tables>;
```

The `TRUNCATE` command is similar to the command in the previous slide.

It can only clear entire tables, but it can clear multiple tables at once, separated by commas.

When removing all rows from a table, this is the preferred method.

Delete Data

Delete some rows.

```
DELETE FROM <table>
WHERE <condition>;
```

The **`TRUNCATE`** command does not allow removing specific rows in a table.

The **`<condition>`** in the **`WHERE`** clause of the **`DELETE FROM`** command can be used to do so.

Delete Data

```
personal=# DELETE FROM friends
personal=# WHERE first_name = 'Pete';
DELETE 1
personal=# SELECT * FROM friends;
 first_name | last_name |   phone    | age
------------+-----------+------------+-----
 Lisa       | Klepp     | 916736453 |  33
(1 row)
```

THANK YOU

Contact Details
DCI Digital Career Institute gGmbH

Digital Career Institute
DCI