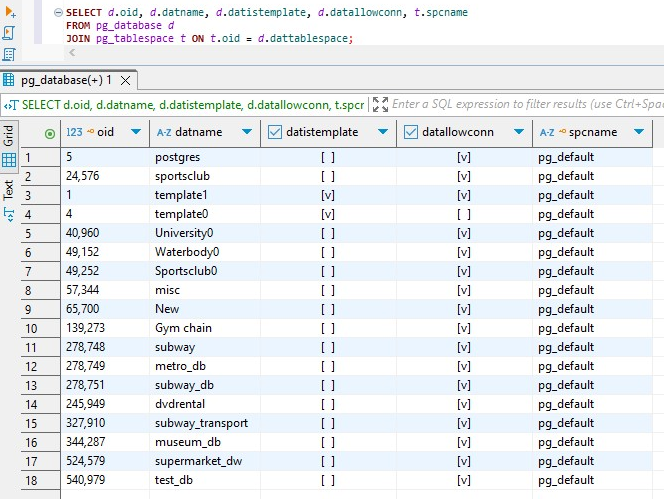
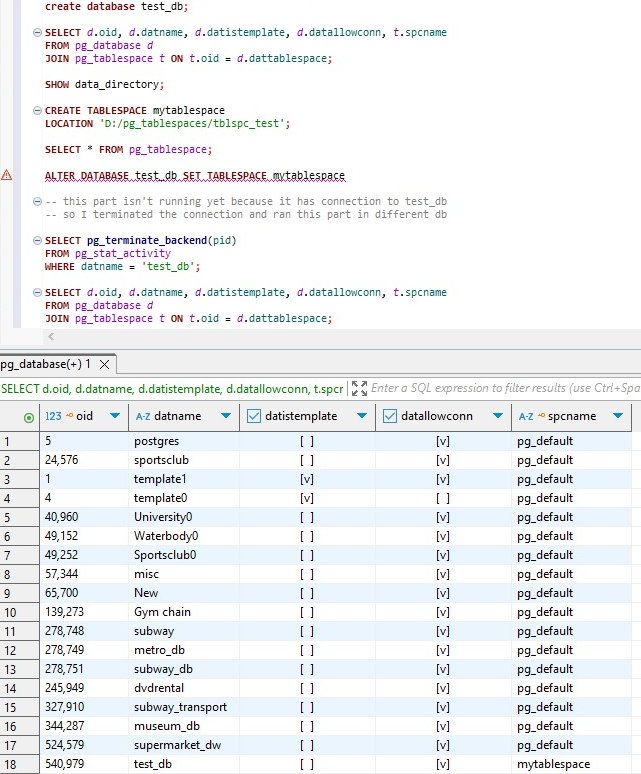
Task 2.1



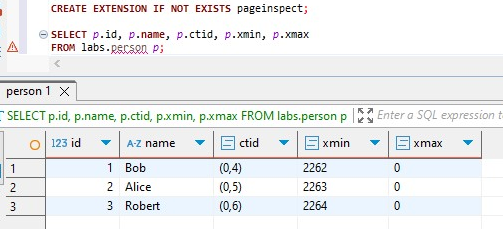
This query showed **all 18 databases** currently present in your PostgreSQL instance and that all of them are using the **default tablespace** (pg\_default), including the newly created one: test\_db. The [v] in datallowconn shows all these databases are **open for connections**. Some databases (like template1, template0, University0, etc.) have [v] in datistemplate, meaning they can be used as a **source template** when creating new databases.

Task 2.2



Task 3.1

Before running queries:



This shows each row's:

ctid: physical tuple location (like (page, row))

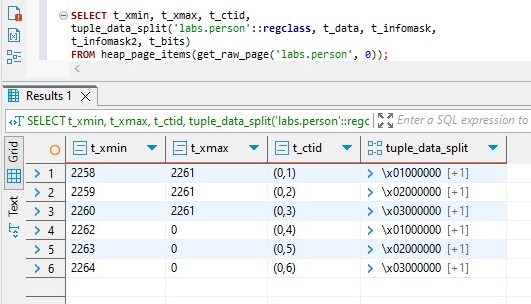
xmin: transaction that created the row

xmax: transaction that deleted the row

**At this point**, all visible rows are active, meaning:

xmax = 0 (not deleted), xmin reflects the transaction ID that inserted them (2262, 2263, 2264)

At first I inserted rows by calling the schema and table and I deleted it with 2261st transaction.



1)After running

**INSERT INTO labs.person VALUES (4, 'John');**

PostgreSQL **appends** a new tuple to the labs.person heap file.

The new row has: fresh ctid, xmin = 2265, xmax = 0. Now the row is **visible to all future transactions.**

2) After running

**UPDATE labs.person SET name = 'Alex' WHERE id = 2;**

**UPDATE is implemented as DELETE + INSERT**. Old version (id=2, name='Alice') is marked with:

xmin = 2263 and xmax = 2266 as the previous row was deleted by 2266th transaction.

New version (id=2, name='Alex') is created with: new ctid (e.g., (0,6)) xmin = 2266, xmax = 0

3) After running

**DELETE FROM labs.person WHERE id = 3;**

In this process third row where xmin = 2264 is deleted to xmax is updated to 2267 and we have new live row.

4) After running

**INSERT INTO labs.person VALUES (999, 'Test');**

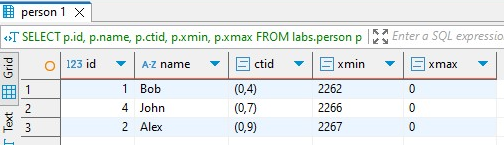
**DELETE FROM labs.person WHERE id = 999;**

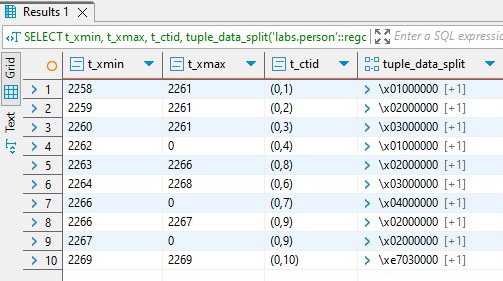
PostgreSQL creates a tuple with:

xmin = 2269 xmax = 2269.

The row is **created and deleted in the same transaction**. As a result, this tuple **is never visible** to any other transaction — not even the one that reads committed data after commit.

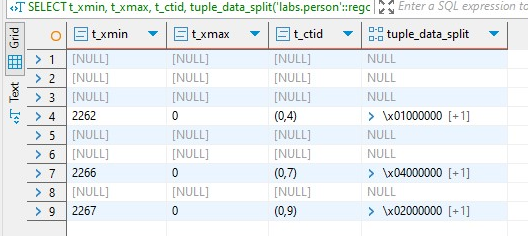
After running all of them we get this result:





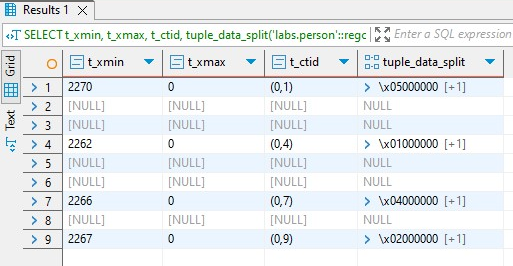
Task 3.2

1) After vacuum:



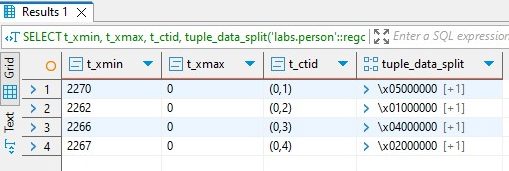
PostgreSQL **marks dead tuples as reusable**, but **does not physically remove them** or **shrink the table file**. It only **updates visibility maps and frees up dead space for reuse** in future inserts. Because of this, dead rows still appear in heap\_page\_items(...) inspection. PostgreSQL can now **reuse** space occupied by dead tuples in future writes.

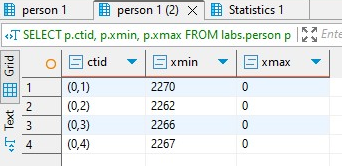
2) After insertion:



PostgreSQL checks the free space from the **visibility map**. If there is a slot where a dead tuple (like id = 3 or 999) was stored, it will **reuse that same physical location** (ctid) for the new row. Because of it new row gets the ctid of a previously deleted one.

3) After vacuum full:





**VACUUM FULL** performs a full table **rewrite**, copies all live rows to **a new physical file**, deletes the old file with dead/obsolete rows. CTIDs of all rows are changed (because their physical positions change) and disk space is actually **reclaimed** (file size shrinks)After all, all dead tuples are gone. Inspection with heap\_page\_items shows **only four live rows**.