Ciências
ULisboa

Concurrency Anomalies in PostgreSQL

ADVANCED DATABASES

GROUP 1: DANIELA VIEIRA, JOÃO RAIMUNDO, JOÃO RATO, MARIA VIEIRA
PROFESSOR: CÁTIA PESQUITA

22 November 2021

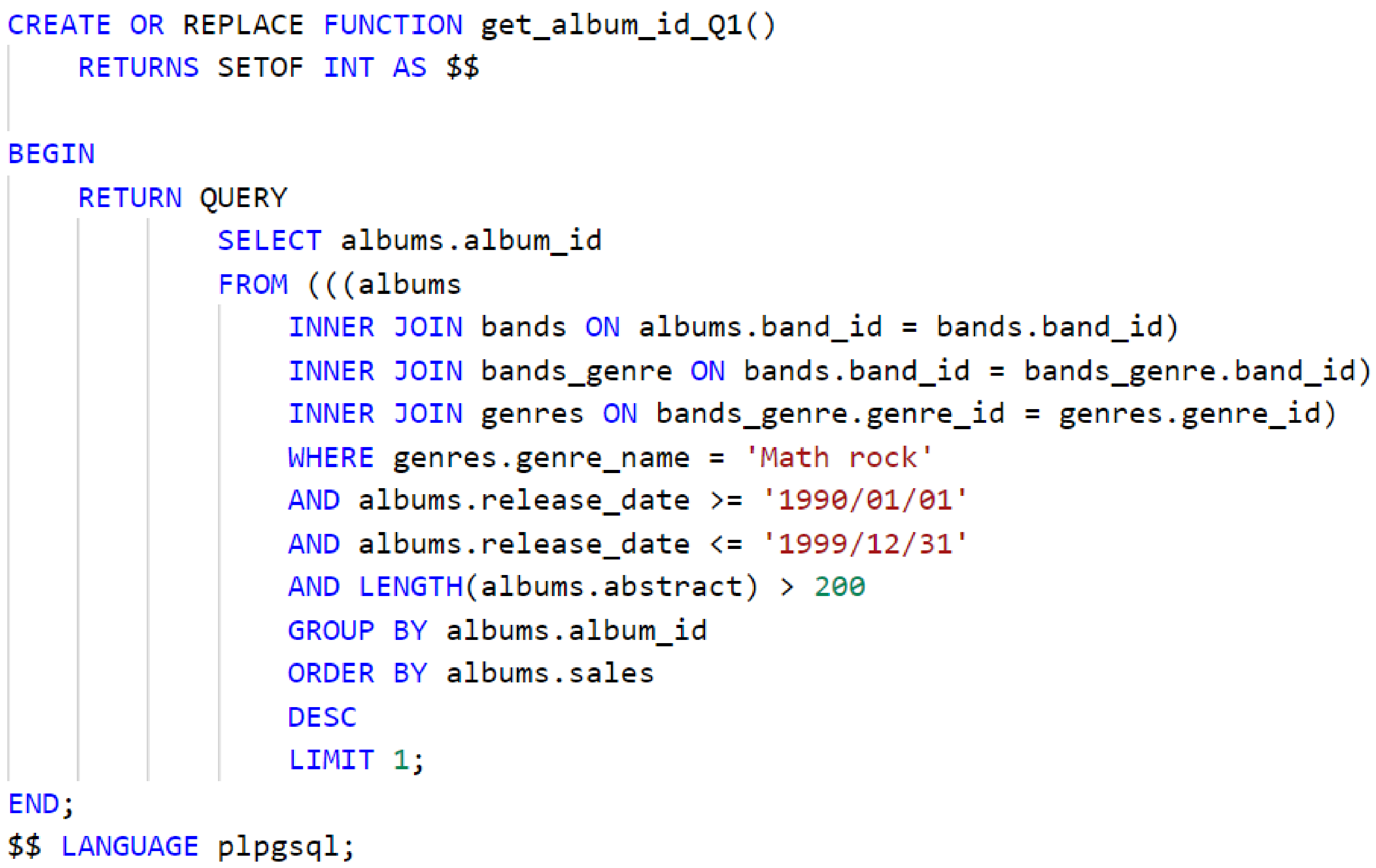
From *PostgreSQL* to *PL/pgSQL*

- ‘Translate’ the queries
- Procedural Language that facilitate concurrency testing

QUERY 1

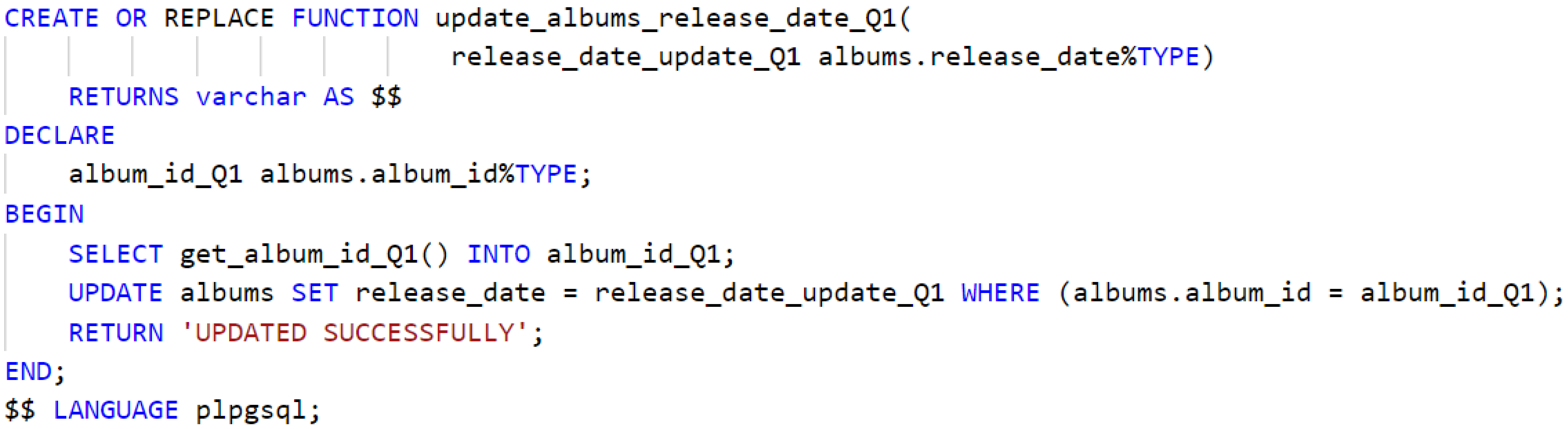
Update to 1980/01/01 the release date of all albums for the genre Math rock, which were released in the 90’s with an abstract over 200 characters, and that had most sales.

SELECT FUNCTION – QUERY 1



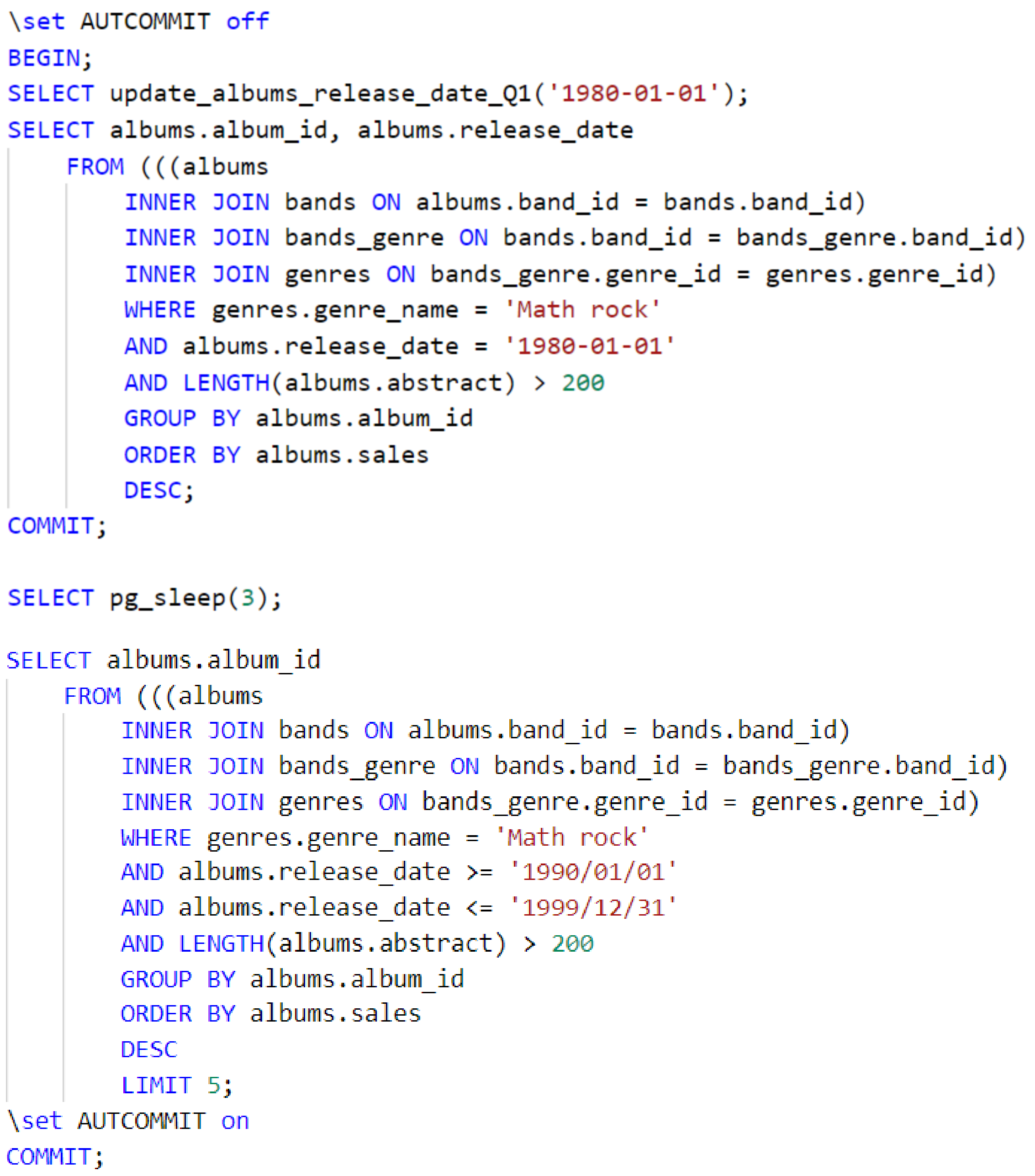
```
CREATE OR REPLACE FUNCTION get_album_id_Q1()
    RETURNS SETOF INT AS $$

BEGIN
    RETURN QUERY
            SELECT albums.album_id
            FROM (((albums
                INNER JOIN bands ON albums.band_id = bands.band_id)
                INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
                INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
                WHERE genres.genre_name = 'Math rock'
                AND albums.release_date >= '1990/01/01'
                AND albums.release_date <= '1999/12/31'
                AND LENGTH(albums.abstract) > 200
                GROUP BY albums.album_id
                ORDER BY albums.sales
                DESC
                LIMIT 1;
END;
$$ LANGUAGE plpgsql;
```

UPDATE FUNCTION – QUERY 1



```
CREATE OR REPLACE FUNCTION update_albums_release_date_Q1(
                            release_date_update_Q1 albums.release_date%TYPE)
    RETURNS varchar AS $$
DECLARE
    album_id_Q1 albums.album_id%TYPE;
BEGIN
    SELECT get_album_id_Q1() INTO album_id_Q1;
    UPDATE albums SET release_date = release_date_update_Q1 WHERE (albums.album_id = album_id_Q1);
    RETURN 'UPDATED SUCCESSFULLY';
END;
$$ LANGUAGE plpgsql;
```

RUN TRANSACTION TWICE,
IN DISTINCT SHELLS,
AT THE SAME TIME



```
\set AUTCOMMIT off
BEGIN;
SELECT update_albums_release_date_Q1('1980-01-01');
SELECT albums.album_id, albums.release_date
    FROM (((albums
        INNER JOIN bands ON albums.band_id = bands.band_id)
        INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
        INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
        WHERE genres.genre_name = 'Math rock'
        AND albums.release_date = '1980-01-01'
        AND LENGTH(albums.abstract) > 200
        GROUP BY albums.album_id
        ORDER BY albums.sales
        DESC;
COMMIT;

SELECT pg_sleep(3);

SELECT albums.album_id
    FROM (((albums
        INNER JOIN bands ON albums.band_id = bands.band_id)
        INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
        INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
        WHERE genres.genre_name = 'Math rock'
        AND albums.release_date >= '1990/01/01'
        AND albums.release_date <= '1999/12/31'
        AND LENGTH(albums.abstract) > 200
        GROUP BY albums.album_id
        ORDER BY albums.sales
        DESC
        LIMIT 5;
\set AUTCOMMIT on
COMMIT;
```

OUTPUTS

- EQUAL OUTPUTS
- UPDATING THE SAME PIECE OF DATA
- CONSISTENCY ANOMALY OCCURRED !
  - **DIRTY READ !**

- Manifests when a transaction can read uncommitted changes of some other concurrent transaction.

SHELL 1

Transaction 1

```
 update_albums_release_date_q1
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 album_id | release_date
----------+--------------
    23907 | 1980-01-01

(1 row)
```

```
 album_id
----------
    21300
     2379
     2381
    34011
     2377

(5 rows)
```

SHELL 2

Transaction 2

```
 update_albums_release_date_q1
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 album_id | release_date
----------+--------------
    23907 | 1980-01-01

(1 row)
```

```
 album_id
----------
    21300
     2379
     2381
    34011
     2377

(5 rows)
```

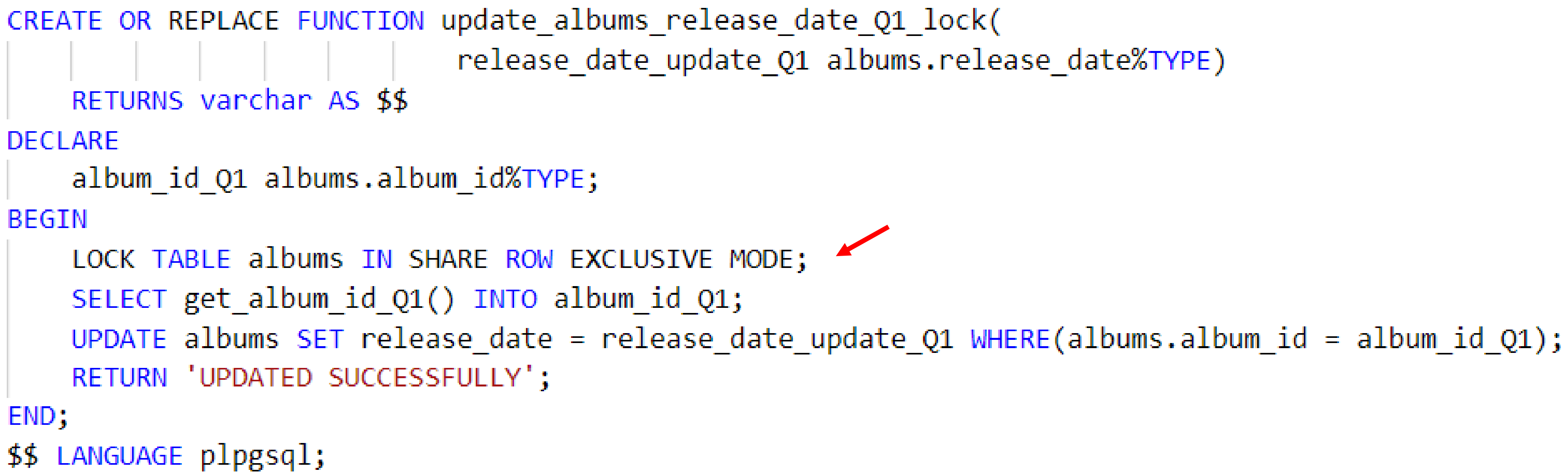
SOLVING DIRTY READ PHENOMENA

Approaches that we used:

- Lock-Based Concurrency Control
- Isolation Levels

SOLVING DIRTY READ PHENOMENA - LOCKS

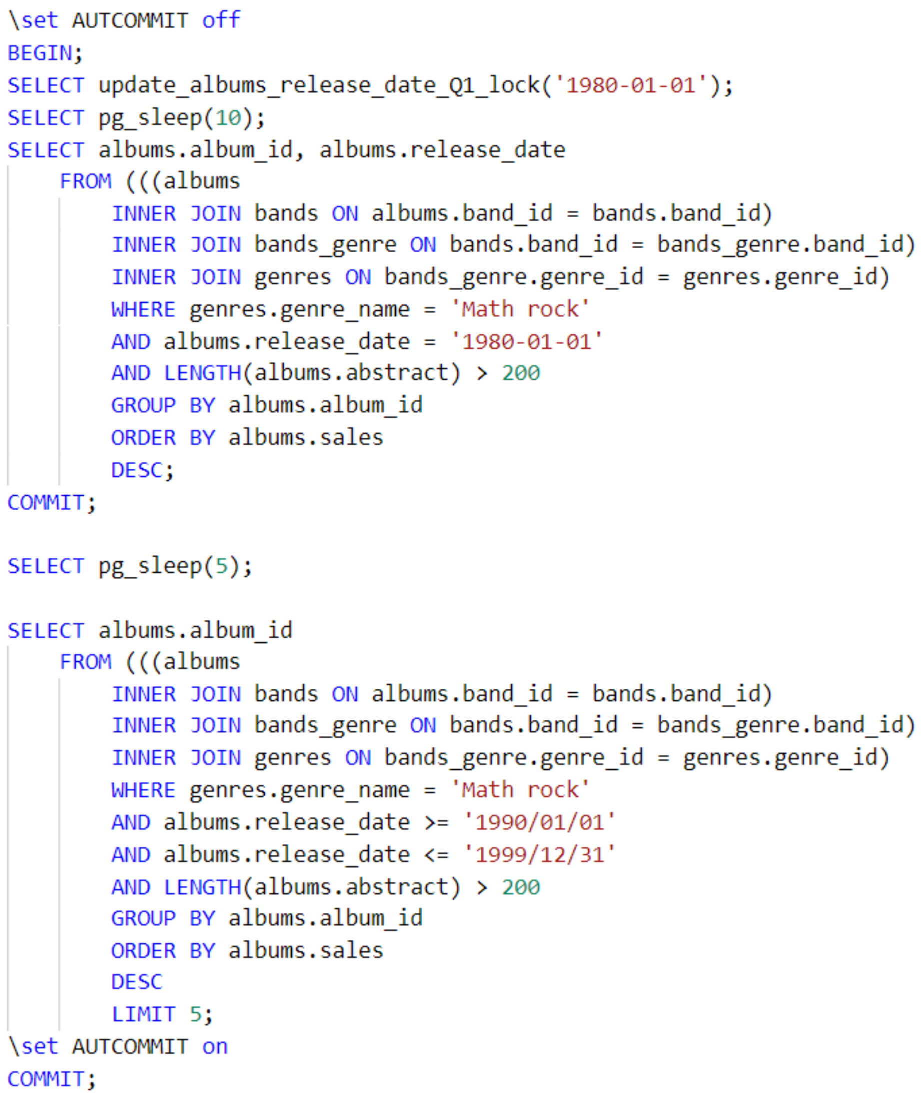
- Implementing **LOCKS** into the Update Function -> **NEW FUNCTION** update_albums_release_date_Q1_lock()
- **LOCK TABLE IN SHARE ROW EXCLUSIVE MODE**
- Protects a table **against concurrent data changes** (**reading & writing**). Only one session can hold it at time



```
CREATE OR REPLACE FUNCTION update_albums_release_date_Q1_lock(
                            release_date_update_Q1 albums.release_date%TYPE)
    RETURNS varchar AS $$
DECLARE
    album_id_Q1 albums.album_id%TYPE;
BEGIN
    LOCK TABLE albums IN SHARE ROW EXCLUSIVE MODE;
    SELECT get_album_id_Q1() INTO album_id_Q1;
    UPDATE albums SET release_date = release_date_update_Q1 WHERE(albums.album_id = album_id_Q1);
    RETURN 'UPDATED SUCCESSFULLY';
END;
$$ LANGUAGE plpgsql;
```

SOLVING DIRTY READ PHENOMENA - LOCKS - TRANSACTIONS

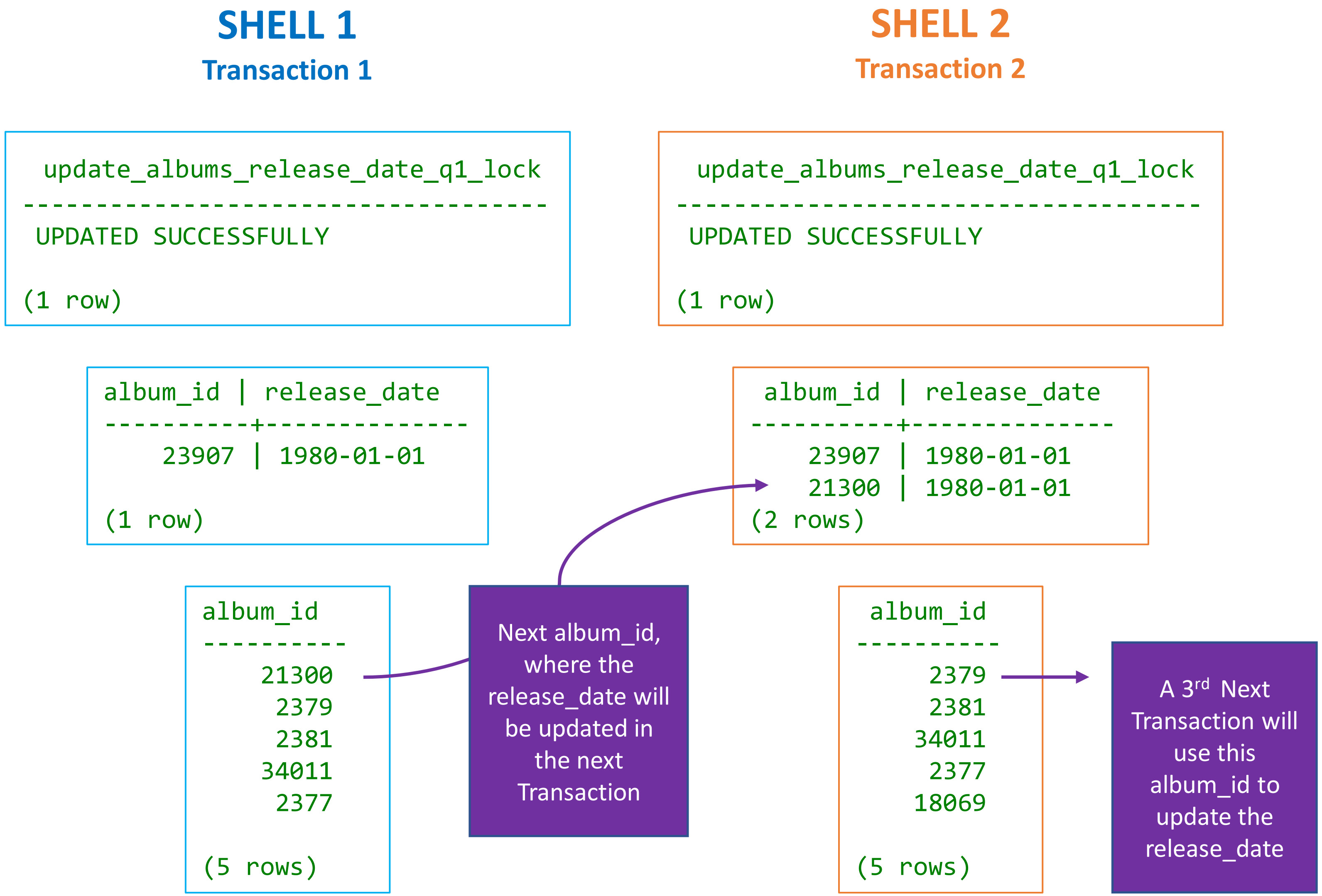
RUN TRANSACTIONS
AGAIN TWICE,
IN DISTINCT SHELLS
AT THE SAME TIME,
BUT THIS TIME WITH
LOCKS IMPLEMENTED.



```
\set AUTCOMMIT off
BEGIN;
SELECT update_albums_release_date_Q1_lock('1980-01-01');
SELECT pg_sleep(10);
SELECT albums.album_id, albums.release_date
    FROM (((albums
        INNER JOIN bands ON albums.band_id = bands.band_id)
        INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
        INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
        WHERE genres.genre_name = 'Math rock'
        AND albums.release_date = '1980-01-01'
        AND LENGTH(albums.abstract) > 200
        GROUP BY albums.album_id
        ORDER BY albums.sales
        DESC;
COMMIT;

SELECT pg_sleep(5);

SELECT albums.album_id
    FROM (((albums
        INNER JOIN bands ON albums.band_id = bands.band_id)
        INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
        INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
        WHERE genres.genre_name = 'Math rock'
        AND albums.release_date >= '1990/01/01'
        AND albums.release_date <= '1999/12/31'
        AND LENGTH(albums.abstract) > 200
        GROUP BY albums.album_id
        ORDER BY albums.sales
        DESC
        LIMIT 5;
\set AUTCOMMIT on
COMMIT;
```

SOLVING DIRTY READ PHENOMENA - LOCKS - OUTPUTS

- DIRTY READ PHENOMENA **SOLVED**
- The **1st Transaction**, occurred in **SHELL 1**, Updating the release_date of the **album_id = ‘23907’**
- The **2nd Transaction**, occurred in **SHELL 2**, Updating the release_date oh the **album_id = ‘21300’**
- If a 3rd Transaction runs, it will Update the release_date for the **album_id = ‘2379’.** The **next** album with the **most sales**.



SHELL 1

Transaction 1

```
 update_albums_release_date_q1_lock
--------------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
album_id | release_date
----------+--------------
    23907 | 1980-01-01

(1 row)
```

```
album_id
----------
    21300
     2379
     2381
    34011
     2377

(5 rows)
```

Next album_id, where the release_date will be updated in the next Transaction

SHELL 2

Transaction 2

```
 update_albums_release_date_q1_lock
--------------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 album_id | release_date
----------+--------------
    23907 | 1980-01-01
    21300 | 1980-01-01
(2 rows)
```

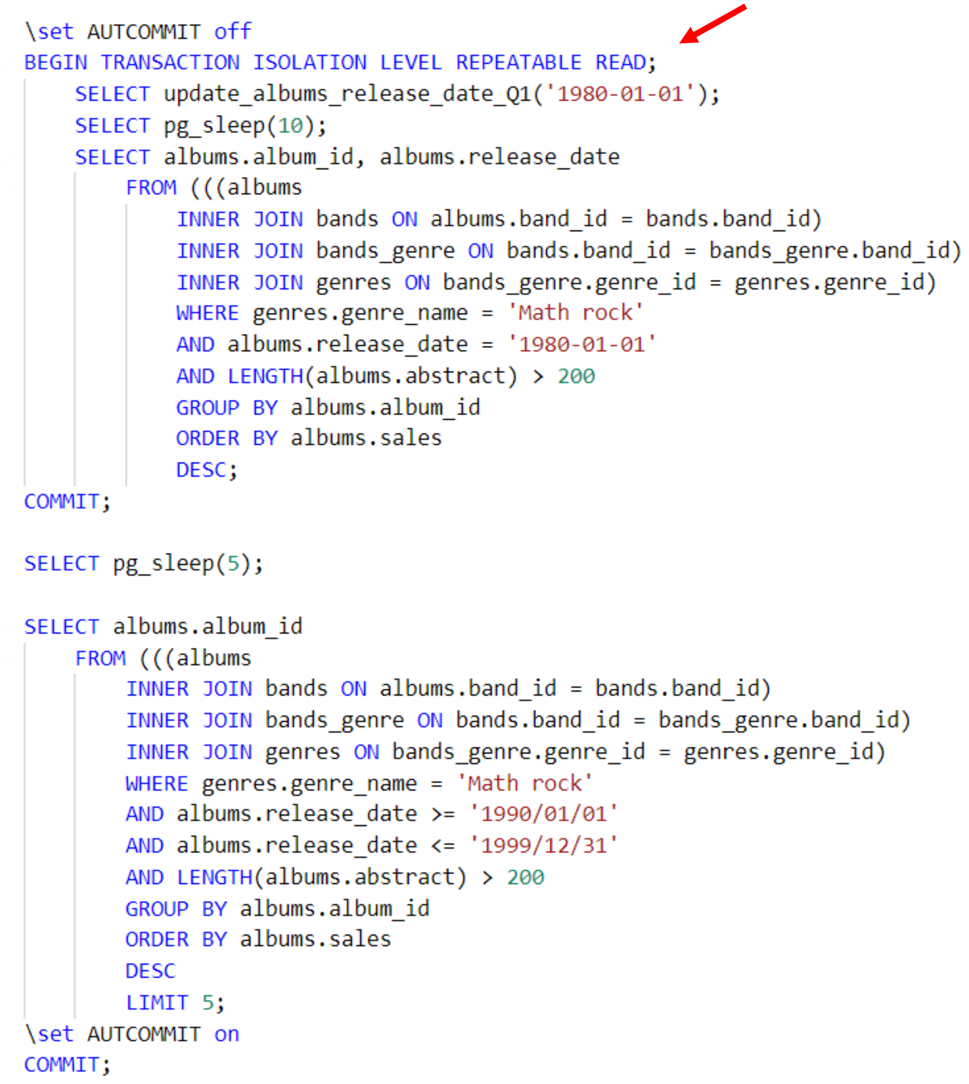
```
 album_id
----------
     2379
     2381
    34011
     2377
    18069

(5 rows)
```

A 3rd Next Transaction will use this album_id to update the release_date

SOLVING DIRTY READ PHENOMENA - ISOLATION LEVELS

TRANSACTIONS

- TRANSACTION ISOLATION LEVEL USED: **REPEATABLE READ**
- Even though PG documentation refers that levels less restricted, like committed reads, does not allow Dirty Reads Phenomena – In our case it still happened, so we used a more restricted level.



```
\set AUTCOMMIT off
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
    SELECT update_albums_release_date_Q1('1980-01-01');
    SELECT pg_sleep(10);
    SELECT albums.album_id, albums.release_date
        FROM (((albums
            INNER JOIN bands ON albums.band_id = bands.band_id)
            INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
            INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
            WHERE genres.genre_name = 'Math rock'
            AND albums.release_date = '1980-01-01'
            AND LENGTH(albums.abstract) > 200
            GROUP BY albums.album_id
            ORDER BY albums.sales
            DESC;
COMMIT;

SELECT pg_sleep(5);

SELECT albums.album_id
    FROM (((albums
        INNER JOIN bands ON albums.band_id = bands.band_id)
        INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
        INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
        WHERE genres.genre_name = 'Math rock'
        AND albums.release_date >= '1990/01/01'
        AND albums.release_date <= '1999/12/31'
        AND LENGTH(albums.abstract) > 200
        GROUP BY albums.album_id
        ORDER BY albums.sales
        DESC
        LIMIT 5;
\set AUTCOMMIT on
COMMIT;
```

SOLVING DIRTY READ PHENOMENA - ISOLATION LEVELS

OUPUTS

SHELL 1

Transaction 1

```
 update_albums_release_date_q1_lock
------------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 album_id | release_date
----------+--------------
    23907 | 1980-01-01
    21300 | 1980-01-01
     2379 | 1980-01-01

(3 rows)
```

```
 album_id
----------
     2381
    34011
     2377
    18069
    18067
(5 rows)
```

SHELL 2

Transaction 2

```
ERROR:  could not serialize access due to concurrent update
CONTEXT:  SQL statement "UPDATE albums SET release_date = release_date_update_Q1 WHERE
(albums.album_id = album_id_Q1)"
PL/pgSQL function update_albums_release_date_q1(date) line 7 at SQL statement
bands_db=!# SELECT pg_sleep(10);
ERROR:  current transaction is aborted, commands ignored until end of transaction block
```

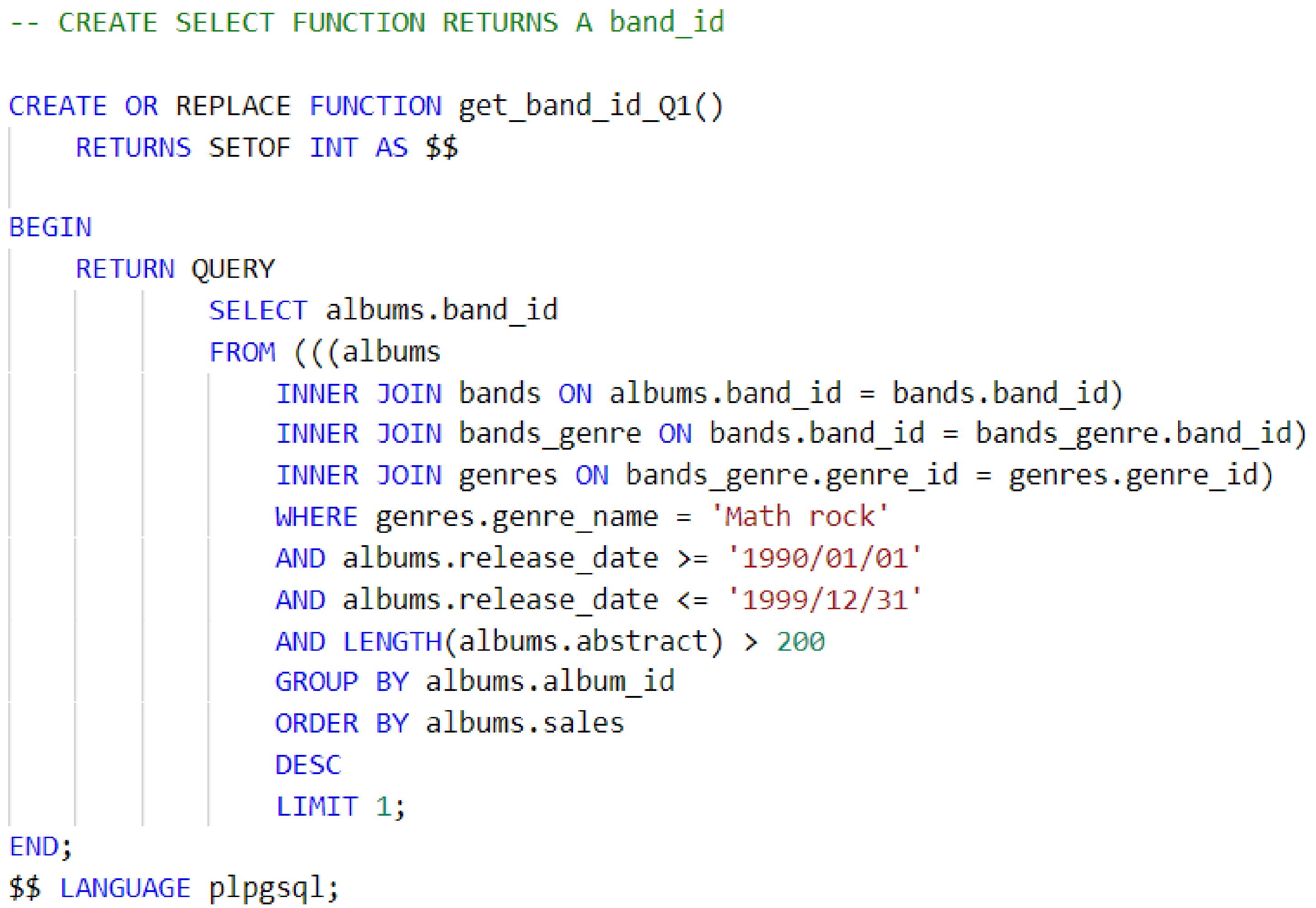
INDUCING PHANTOM READ PHENOMENA

Phantom Read – Occurs when **two queries** ran successively, within the **same transaction**, displays **different** sets of **results** due to the **insertion** or **deletion** of one or more **rows** between read statements.

NEXT STEPS:

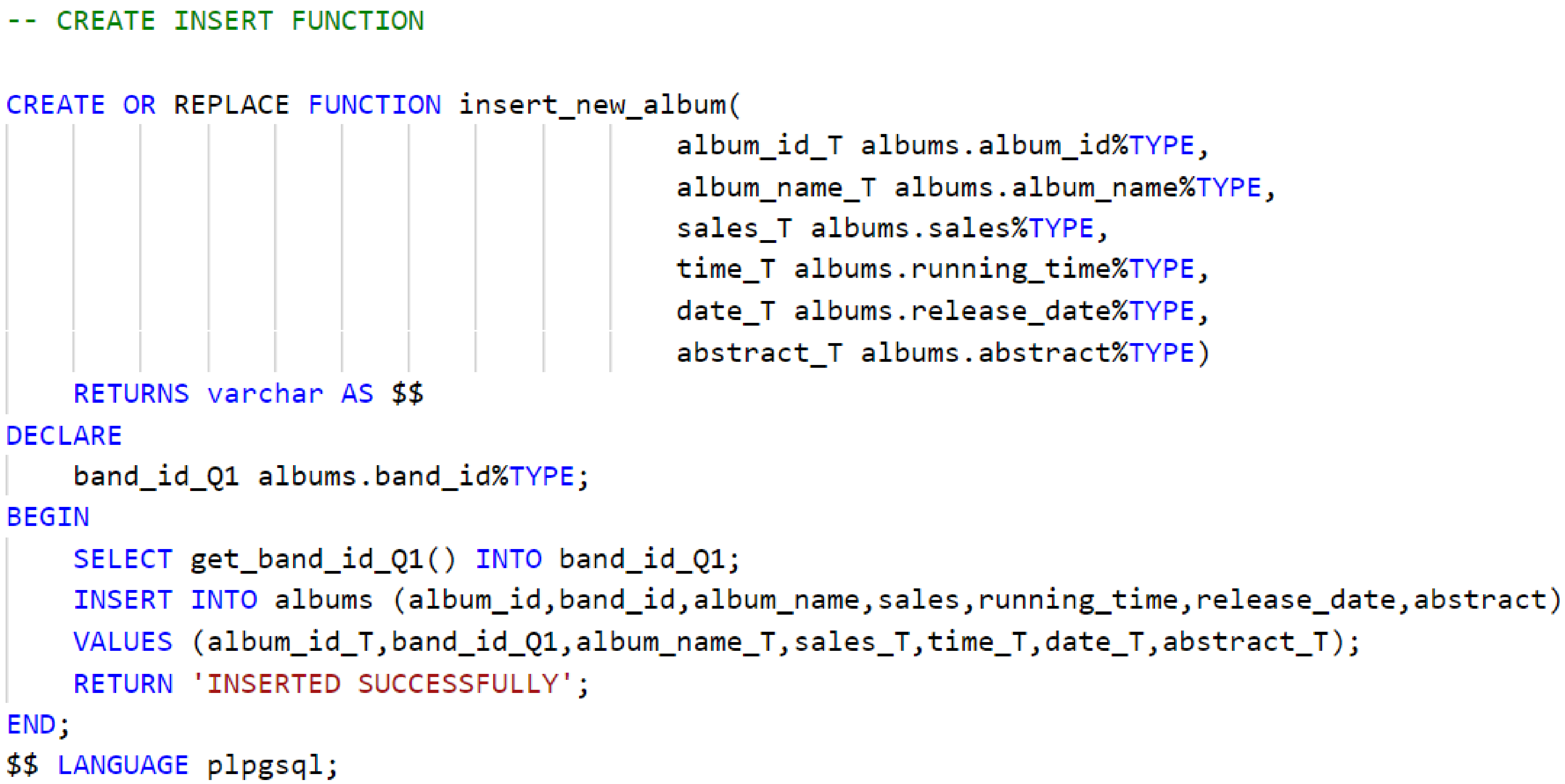
- **CREATE A NEW SELECT FUNCTION**
- **CREATE AN INSERT FUNCTION**

INDUCING PHANTOM READ PHENOMENA - SELECT FUNCTION



```
-- CREATE SELECT FUNCTION RETURNS A band_id

CREATE OR REPLACE FUNCTION get_band_id_Q1()
    RETURNS SETOF INT AS $$

BEGIN
    RETURN QUERY
            SELECT albums.band_id
            FROM (((albums
                INNER JOIN bands ON albums.band_id = bands.band_id)
                INNER JOIN bands_genre ON bands.band_id = bands_genre.band_id)
                INNER JOIN genres ON bands_genre.genre_id = genres.genre_id)
                WHERE genres.genre_name = 'Math rock'
                AND albums.release_date >= '1990/01/01'
                AND albums.release_date <= '1999/12/31'
                AND LENGTH(albums.abstract) > 200
                GROUP BY albums.album_id
                ORDER BY albums.sales
                DESC
                LIMIT 1;
END;
$$ LANGUAGE plpgsql;
```

INDUCING PHANTOM READ PHENOMENA - INSERT FUNCTION

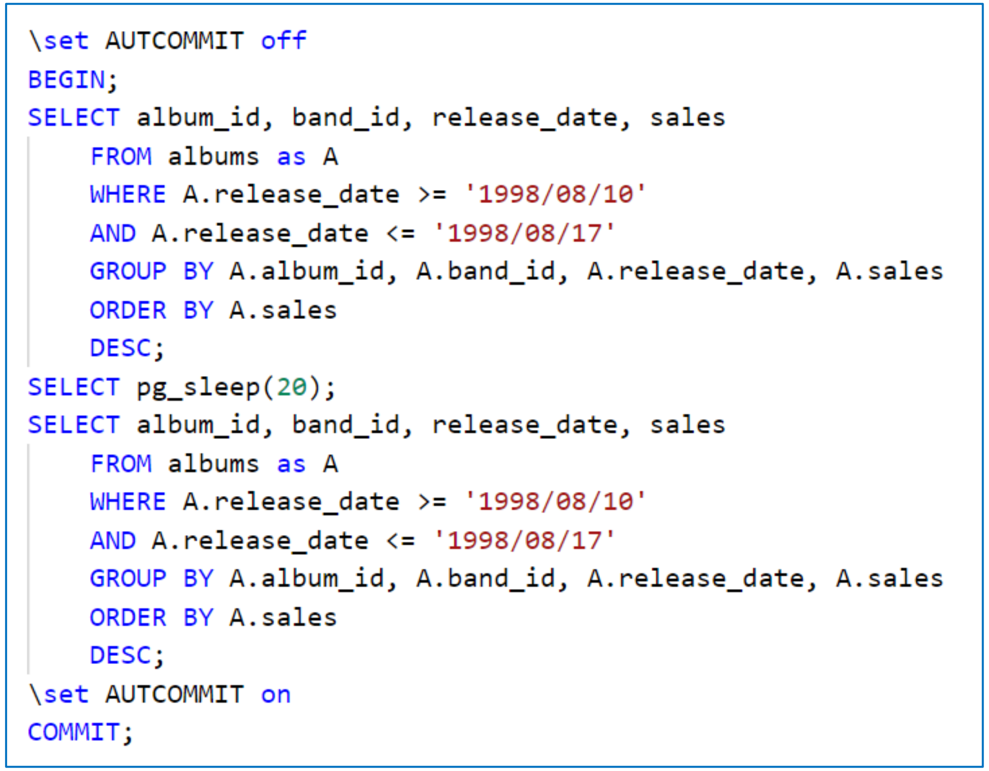


```sql
-- CREATE INSERT FUNCTION

CREATE OR REPLACE FUNCTION insert_new_album(
                                        album_id_T albums.album_id%TYPE,
                                        album_name_T albums.album_name%TYPE,
                                        sales_T albums.sales%TYPE,
                                        time_T albums.running_time%TYPE,
                                        date_T albums.release_date%TYPE,
                                        abstract_T albums.abstract%TYPE)
    RETURNS varchar AS $$
DECLARE
    band_id_Q1 albums.band_id%TYPE;
BEGIN
    SELECT get_band_id_Q1() INTO band_id_Q1;
    INSERT INTO albums (album_id,band_id,album_name,sales,running_time,release_date,abstract)
    VALUES (album_id_T,band_id_Q1,album_name_T,sales_T,time_T,date_T,abstract_T);
    RETURN 'INSERTED SUCCESSFULLY';
END;
$$ LANGUAGE plpgsql;
```

INDUCING PHANTOM READ PHENOMENA - TRANSACTIONS

SHELL 1

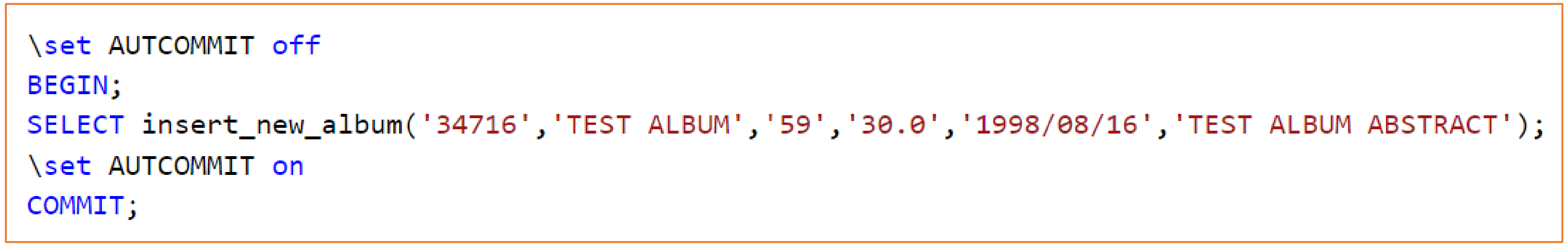
Transaction 1



```
\set AUTCOMMIT off
BEGIN;
SELECT album_id, band_id, release_date, sales
    FROM albums as A
    WHERE A.release_date >= '1998/08/10'
    AND A.release_date <= '1998/08/17'
    GROUP BY A.album_id, A.band_id, A.release_date, A.sales
    ORDER BY A.sales
    DESC;
SELECT pg_sleep(20);
SELECT album_id, band_id, release_date, sales
    FROM albums as A
    WHERE A.release_date >= '1998/08/10'
    AND A.release_date <= '1998/08/17'
    GROUP BY A.album_id, A.band_id, A.release_date, A.sales
    ORDER BY A.sales
    DESC;
\set AUTCOMMIT on
COMMIT;
```

SHELL 2

Transaction 2



```
\set AUTCOMMIT off
BEGIN;
SELECT insert_new_album('34716','TEST ALBUM','59','30.0','1998/08/16','TEST ALBUM ABSTRACT');
\set AUTCOMMIT on
COMMIT;
```

INDUCING PHANTOM READ PHENOMENA - OUTPUTS

SHELL 1
Transaction 1

1st SELECT

```
 album_id | band_id | release_date | sales
----------+---------+--------------+-------
    11958 |    3074 | 1998-08-12   |  9951
     8551 |    2149 | 1998-08-12   |  9880
     9799 |    2540 | 1998-08-10   |  9636
     4994 |    1277 | 1998-08-12   |  9368
    21629 |    5510 | 1998-08-17   |  9176
    11957 |    3074 | 1998-08-12   |  8466
    15571 |    4083 | 1998-08-12   |  8205
     4008 |    1088 | 1998-08-17   |  7657

(8 rows)
```

2nd SELECT

```
 album_id | band_id | release_date | sales
----------+---------+--------------+-------
    11958 |    3074 | 1998-08-12   |  9951
     8551 |    2149 | 1998-08-12   |  9880
     9799 |    2540 | 1998-08-10   |  9636
     4994 |    1277 | 1998-08-12   |  9368
    21629 |    5510 | 1998-08-17   |  9176
    11957 |    3074 | 1998-08-12   |  8466
    15571 |    4083 | 1998-08-12   |  8205
     4008 |    1088 | 1998-08-17   |  7657
    34716 |    4705 | 1998-08-16   |    59
(9 rows)
```

SHELL 2
Transaction 2

```
    insert_new_album
-----------------------
 INSERTED SUCCESSFULLY
```

Transaction 2 influence the result of **Transaction 1**

-> **Phantom READ Phenomena**

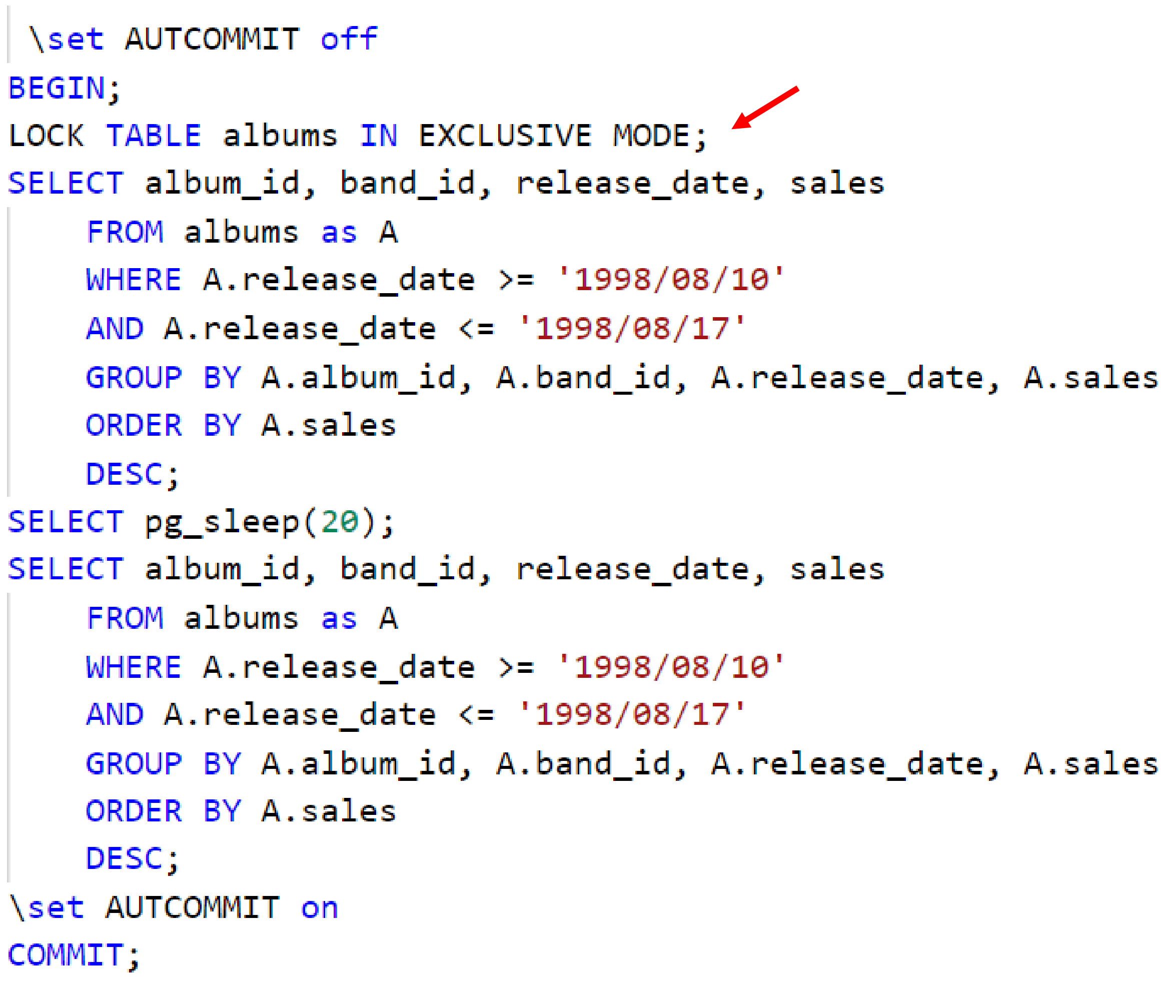
SOLVING PHANTOM READ PHENOMENA

Approaches that we used:

- Lock-Based Concurrency Control
- Isolation Levels

SOLVING PHANTOM READ PHENOMENA - LOCKS

- Implementing **LOCKS** in **Transaction 1 (SHELL 1)**
- **LOCK TABLE** albums **IN EXCLUSIVE MODE**
- Prevents other transactions to **write in table albums** while Transaction 1 runs (until being COMMITTED)



```
 \set AUTCOMMIT off
BEGIN;
LOCK TABLE albums IN EXCLUSIVE MODE;
SELECT album_id, band_id, release_date, sales
    FROM albums as A
    WHERE A.release_date >= '1998/08/10'
    AND A.release_date <= '1998/08/17'
    GROUP BY A.album_id, A.band_id, A.release_date, A.sales
    ORDER BY A.sales
    DESC;
SELECT pg_sleep(20);
SELECT album_id, band_id, release_date, sales
    FROM albums as A
    WHERE A.release_date >= '1998/08/10'
    AND A.release_date <= '1998/08/17'
    GROUP BY A.album_id, A.band_id, A.release_date, A.sales
    ORDER BY A.sales
    DESC;
\set AUTCOMMIT on
COMMIT;
```

SOLVING PHANTOM READ PHENOMENA - LOCKS - OUTPUTS

SHELL 1

Transaction 1

1st SELECT

```
 album_id | band_id | release_date | sales
----------+---------+--------------+-------
    11958 |    3074 | 1998-08-12   |  9951
     8551 |    2149 | 1998-08-12   |  9880
     9799 |    2540 | 1998-08-10   |  9636
     4994 |    1277 | 1998-08-12   |  9368
    21629 |    5510 | 1998-08-17   |  9176
    11957 |    3074 | 1998-08-12   |  8466
    15571 |    4083 | 1998-08-12   |  8205
     4008 |    1088 | 1998-08-17   |  7657

(8 rows)
```

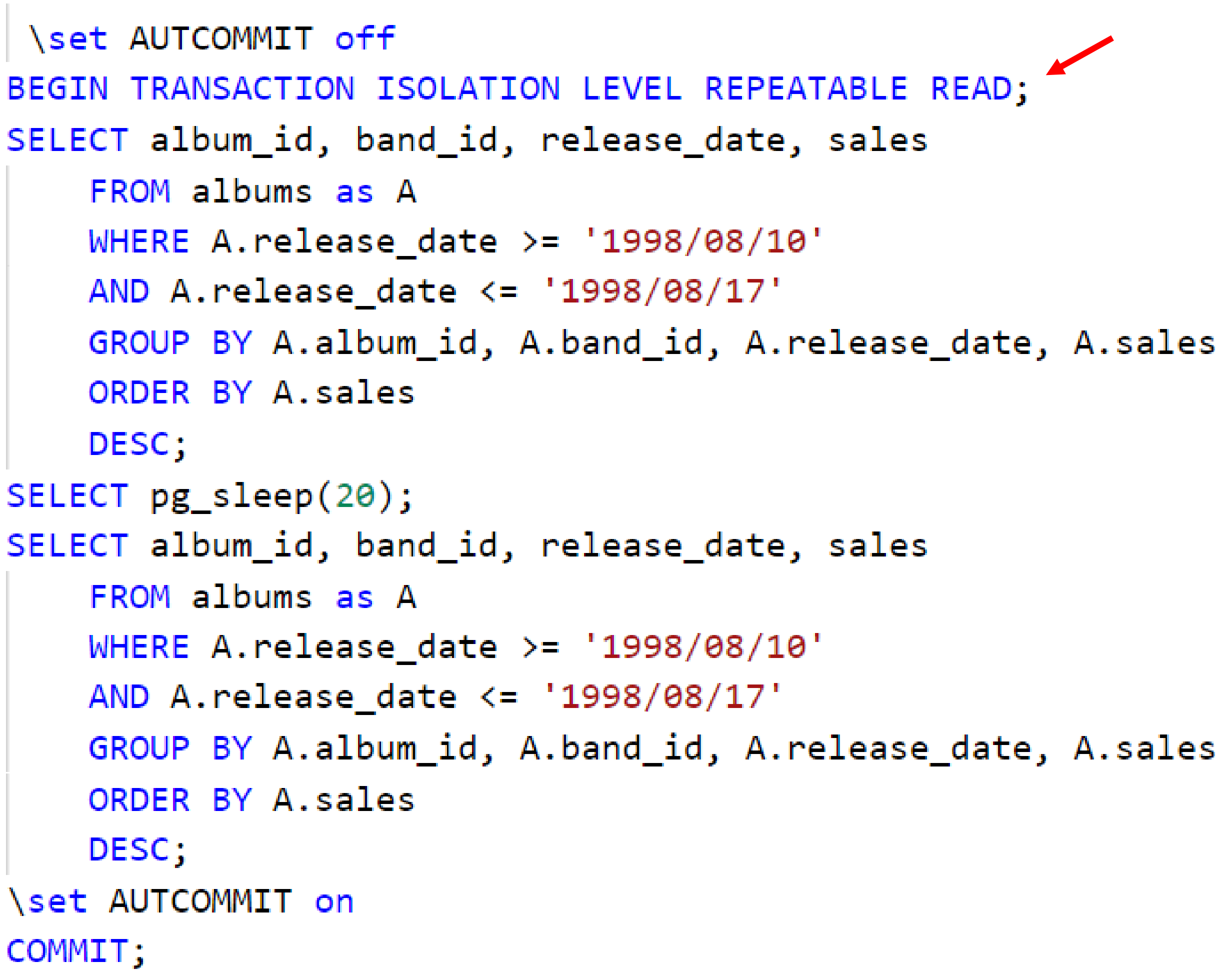
2nd SELECT

```
 album_id | band_id | release_date | sales
----------+---------+--------------+-------
    11958 |    3074 | 1998-08-12   |  9951
     8551 |    2149 | 1998-08-12   |  9880
     9799 |    2540 | 1998-08-10   |  9636
     4994 |    1277 | 1998-08-12   |  9368
    21629 |    5510 | 1998-08-17   |  9176
    11957 |    3074 | 1998-08-12   |  8466
    15571 |    4083 | 1998-08-12   |  8205
     4008 |    1088 | 1998-08-17   |  7657

(8 rows)
```

- **Transaction 1** - Same set of results in both queries
- **Transaction 2** - Waits until Transaction 1 being committed to start the transaction.

SOLVING PHANTOM READ PHENOMENA - ISOLATION LEVELS

- Implemented in **Transaction 1**
- TRANSACTION ISOLATION LEVEL USED: **REPEATABLE READ**



```
 \set AUTCOMMIT off
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT album_id, band_id, release_date, sales
    FROM albums as A
    WHERE A.release_date >= '1998/08/10'
    AND A.release_date <= '1998/08/17'
    GROUP BY A.album_id, A.band_id, A.release_date, A.sales
    ORDER BY A.sales
    DESC;
SELECT pg_sleep(20);
SELECT album_id, band_id, release_date, sales
    FROM albums as A
    WHERE A.release_date >= '1998/08/10'
    AND A.release_date <= '1998/08/17'
    GROUP BY A.album_id, A.band_id, A.release_date, A.sales
    ORDER BY A.sales
    DESC;
\set AUTCOMMIT on
COMMIT;
```

SOLVING PHANTOM READ PHENOMENA - ISOLATION LEVELS

OUPUTS

SHELL 1

Transaction 1

1st SELECT

```
 album_id | band_id | release_date | sales
----------+---------+--------------+-------
    11958 |    3074 | 1998-08-12   |  9951
     8551 |    2149 | 1998-08-12   |  9880
     9799 |    2540 | 1998-08-10   |  9636
     4994 |    1277 | 1998-08-12   |  9368
    21629 |    5510 | 1998-08-17   |  9176
    11957 |    3074 | 1998-08-12   |  8466
    15571 |    4083 | 1998-08-12   |  8205
     4008 |    1088 | 1998-08-17   |  7657

(8 rows)
```

2nd SELECT

```
 album_id | band_id | release_date | sales
----------+---------+--------------+-------
    11958 |    3074 | 1998-08-12   |  9951
     8551 |    2149 | 1998-08-12   |  9880
     9799 |    2540 | 1998-08-10   |  9636
     4994 |    1277 | 1998-08-12   |  9368
    21629 |    5510 | 1998-08-17   |  9176
    11957 |    3074 | 1998-08-12   |  8466
    15571 |    4083 | 1998-08-12   |  8205
     4008 |    1088 | 1998-08-17   |  7657

(8 rows)
```

SHELL 2

Transaction 2

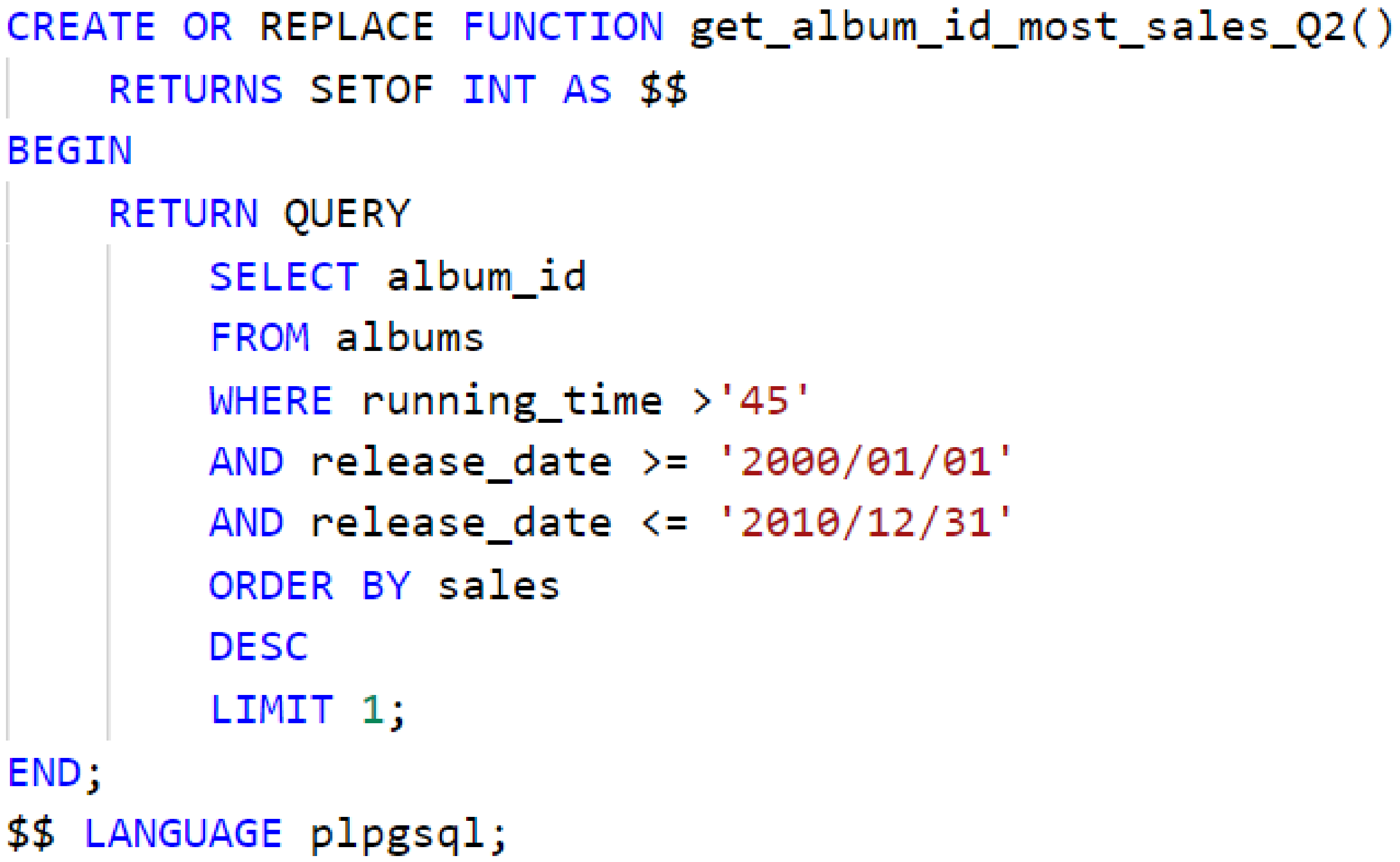
```
   insert_new_album
-----------------------
 INSERTED SUCCESSFULLY
```

- **Transaction 1** - Same set of results in both queries
- **Transaction 2** - It can be committed during Transaction 1 process, without affecting its set of results. Did not report any errors.

QUERY 2

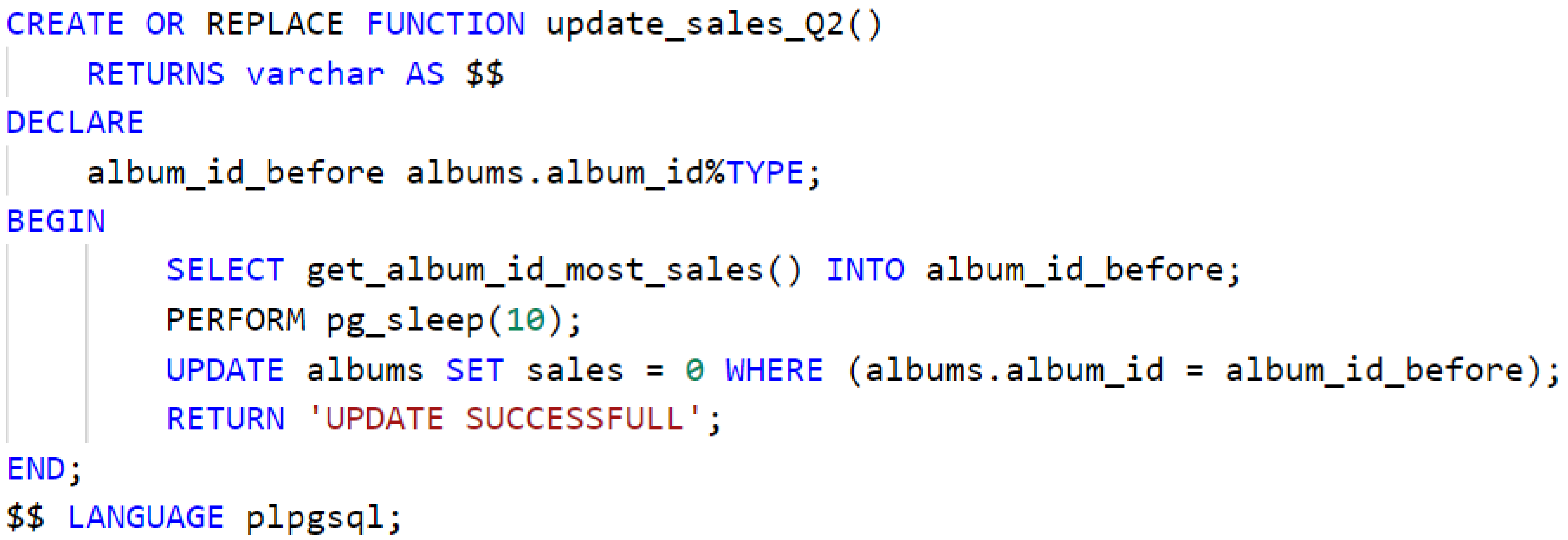
Update to 0 the sales from the album with the most sales in the first decade of the year 2000, and which the running time is longer than 45 minutes.

SELECT FUNCTION – QUERY 2



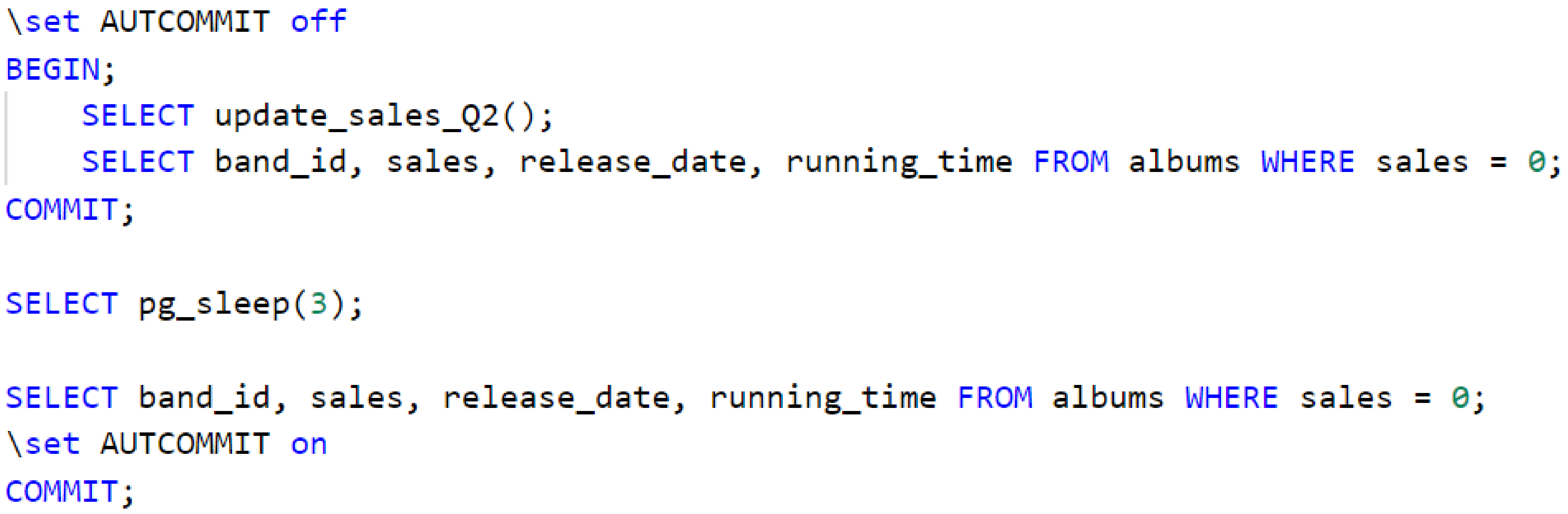
```
CREATE OR REPLACE FUNCTION get_album_id_most_sales_Q2()
    RETURNS SETOF INT AS $$
BEGIN
    RETURN QUERY
        SELECT album_id
        FROM albums
        WHERE running_time >'45'
        AND release_date >= '2000/01/01'
        AND release_date <= '2010/12/31'
        ORDER BY sales
        DESC
        LIMIT 1;
END;
$$ LANGUAGE plpgsql;
```

UPDATE FUNCTION – QUERY 2



```
CREATE OR REPLACE FUNCTION update_sales_Q2()
    RETURNS varchar AS $$
DECLARE
    album_id_before albums.album_id%TYPE;
BEGIN
        SELECT get_album_id_most_sales() INTO album_id_before;
        PERFORM pg_sleep(10);
        UPDATE albums SET sales = 0 WHERE (albums.album_id = album_id_before);
        RETURN 'UPDATE SUCCESSFULL';
END;
$$ LANGUAGE plpgsql;
```

RUN TRANSACTION, IN DISTINCT SHELLS, AT THE SAME TIME



```
\set AUTCOMMIT off
BEGIN;
    SELECT update_sales_Q2();
    SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0;
COMMIT;

SELECT pg_sleep(3);

SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0;
\set AUTCOMMIT on
COMMIT;
```

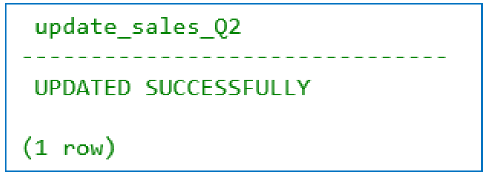
OUTPUTS

- EQUAL OUTPUTS
- UPDATING THE SAME PIECE OF DATA
- CONSISTENCY ANOMALY OCCURRED !
    - **DIRTY READ !**

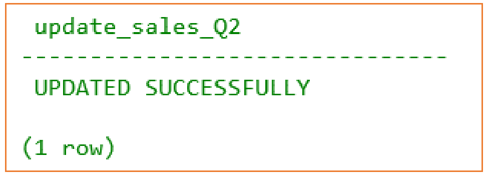
SHELL 1
Transaction 1

SHELL 2
Transaction 2

Update function
output

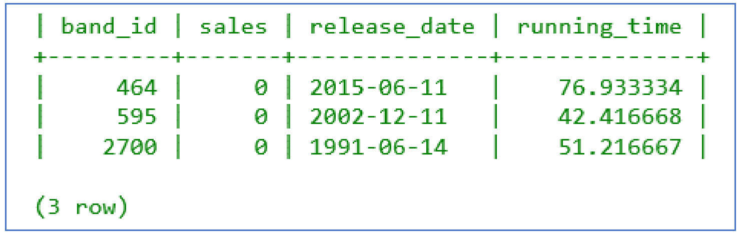


```
 update_sales_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

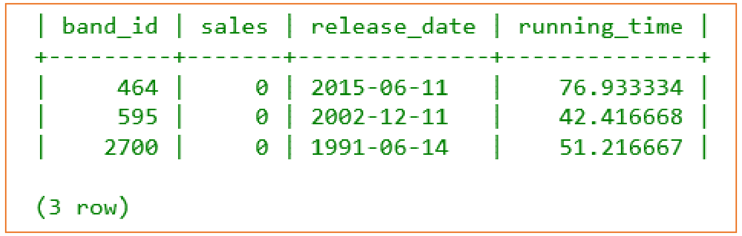


```
 update_sales_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

First SELECT query
output

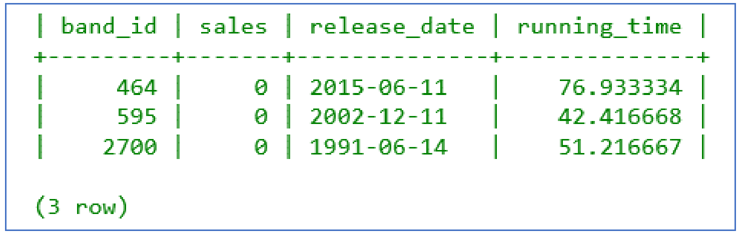


```
| band_id | sales | release_date | running_time |
+---------+-------+--------------+--------------+
|     464 |     0 | 2015-06-11   |    76.933334 |
|     595 |     0 | 2002-12-11   |    42.416668 |
|    2700 |     0 | 1991-06-14   |    51.216667 |

(3 row)
```

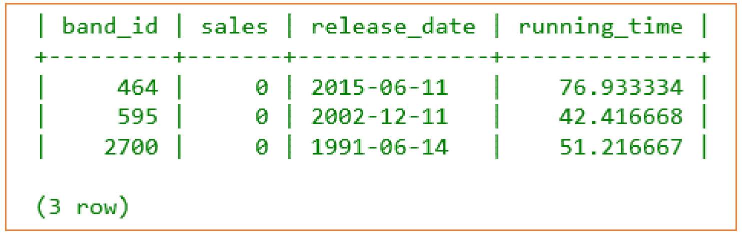


```
| band_id | sales | release_date | running_time |
+---------+-------+--------------+--------------+
|     464 |     0 | 2015-06-11   |    76.933334 |
|     595 |     0 | 2002-12-11   |    42.416668 |
|    2700 |     0 | 1991-06-14   |    51.216667 |

(3 row)
```

Second SELECT query
output

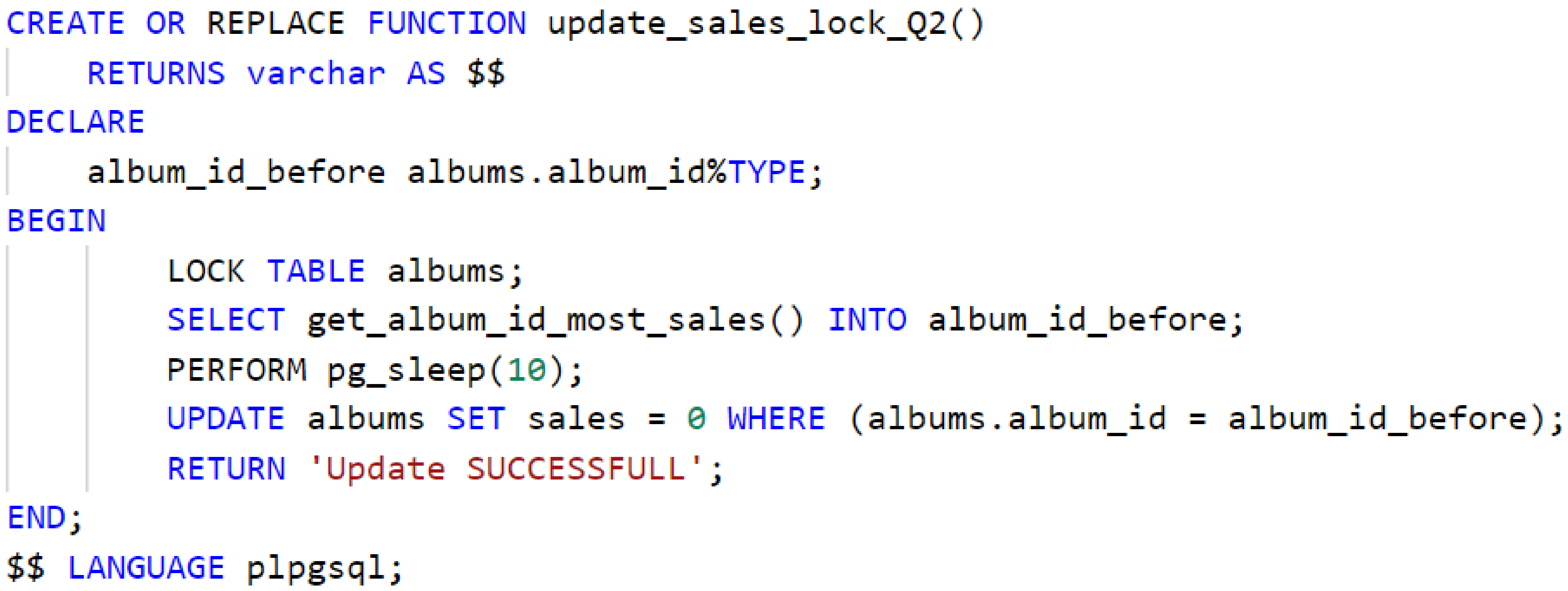


```
| band_id | sales | release_date | running_time |
+---------+-------+--------------+--------------+
|     464 |     0 | 2015-06-11   |    76.933334 |
|     595 |     0 | 2002-12-11   |    42.416668 |
|    2700 |     0 | 1991-06-14   |    51.216667 |

(3 row)
```



```
| band_id | sales | release_date | running_time |
+---------+-------+--------------+--------------+
|     464 |     0 | 2015-06-11   |    76.933334 |
|     595 |     0 | 2002-12-11   |    42.416668 |
|    2700 |     0 | 1991-06-14   |    51.216667 |

(3 row)
```

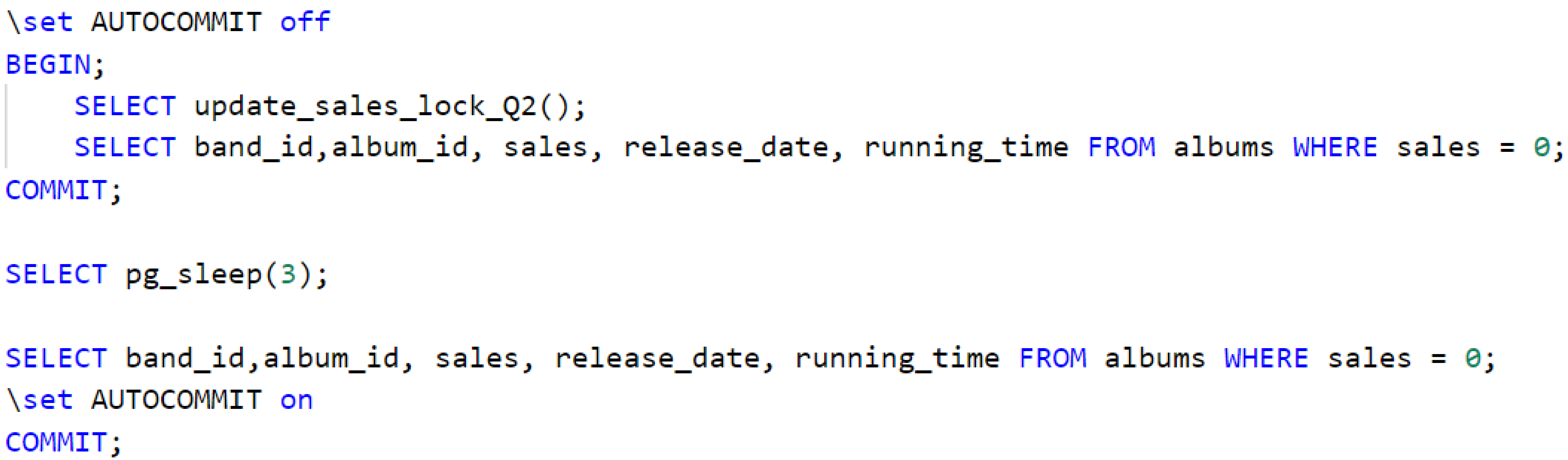
SOLVING DIRTY READ PHENOMENA - LOCKS



```sql
CREATE OR REPLACE FUNCTION update_sales_lock_Q2()
    RETURNS varchar AS $$
DECLARE
    album_id_before albums.album_id%TYPE;
BEGIN
        LOCK TABLE albums;
        SELECT get_album_id_most_sales() INTO album_id_before;
        PERFORM pg_sleep(10);
        UPDATE albums SET sales = 0 WHERE (albums.album_id = album_id_before);
        RETURN 'Update SUCCESSFULL';
END;
$$ LANGUAGE plpgsql;
```

SOLVING DIRTY READ PHENOMENA - LOCKS - TRANSACTIONS

RUN TRANSACTIONS IN DISTINCT SHELLS AT THE SAME TIME,
BUT THIS TIME WITH **LOCKS IMPLEMENTED**.



```
\set AUTOCOMMIT off
BEGIN;
    SELECT update_sales_lock_Q2();
    SELECT band_id,album_id, sales, release_date, running_time FROM albums WHERE sales = 0;
COMMIT;

SELECT pg_sleep(3);

SELECT band_id,album_id, sales, release_date, running_time FROM albums WHERE sales = 0;
\set AUTOCOMMIT on
COMMIT;
```

SOLVING DIRTY READ PHENOMENA - LOCKS - OUTPUTS

SHELL 1
Transaction 1

SHELL 2
Transaction 2

Update function output

```
 update_sales_lock_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 update_sales_lock_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

First SELECT query output

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

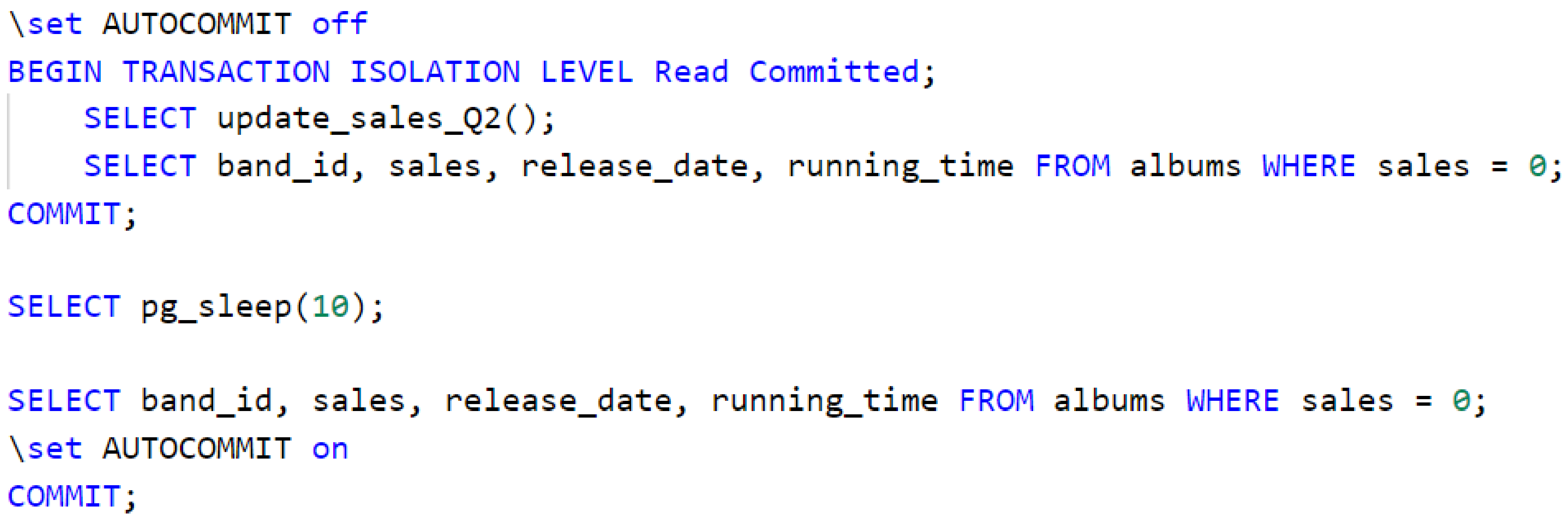
Second SELECT query output

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334
    464 |     1338 |     0 | 2015-06-11   |     79.13333
(2 row)
```

SOLVING DIRTY READ PHENOMENA - ISOLATION LEVELS

TRANSACTION ISOLATION LEVEL USED: **REPEATABLE READ**



```
\set AUTOCOMMIT off
BEGIN TRANSACTION ISOLATION LEVEL Read Committed;
    SELECT update_sales_Q2();
    SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0;
COMMIT;

SELECT pg_sleep(10);

SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0;
\set AUTOCOMMIT on
COMMIT;
```

SOLVING DIRTY READ PHENOMENA - ISOLATION LEVELS

OUPUTS

SHELL 1
Transaction 1

SHELL 2
Transaction 2

Update function output

Shell 1:

```
 update_sales_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

Shell 2:

```
 update_sales_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

First SELECT query output

Shell 1:

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

Shell 2:

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

Second SELECT query output

Shell 1:

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

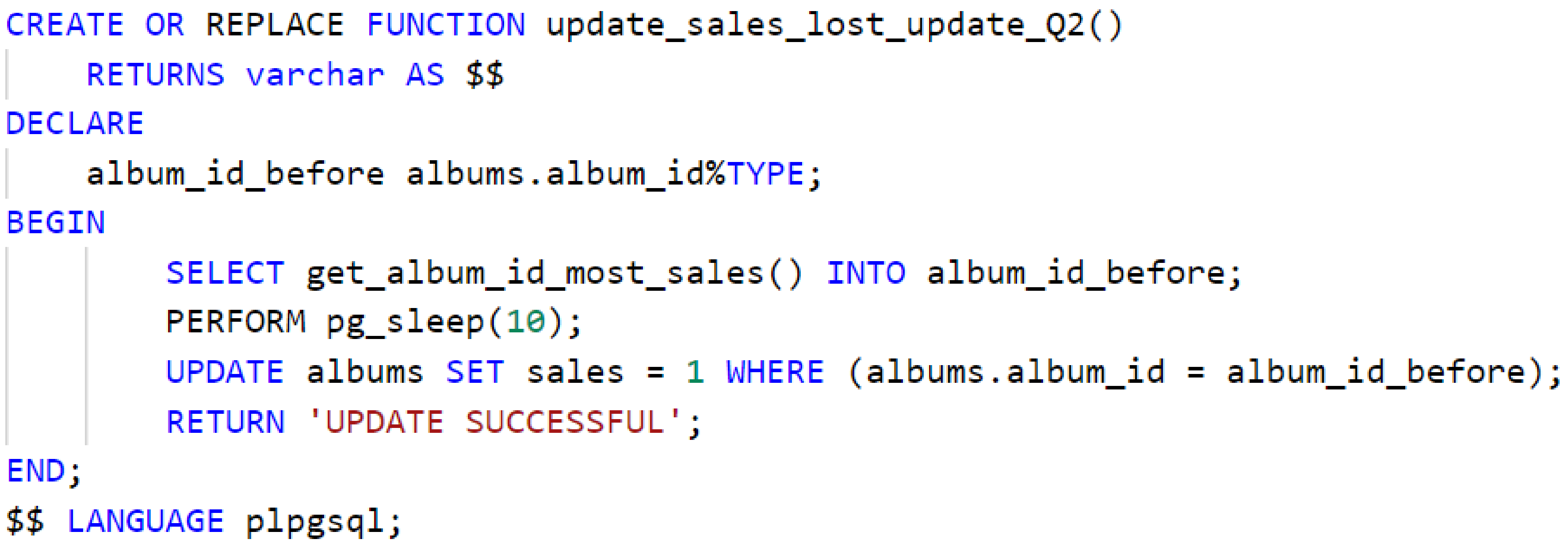
Shell 2:

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334
    464 |     1338 |     0 | 2015-06-11   |     79.13333
(2 row)
```

INDUCING LOST UPDATE PHENOMENA

Lost update – occur when the second update overwrites the first one.

UPDATE FUNCTION



```
CREATE OR REPLACE FUNCTION update_sales_lost_update_Q2()
    RETURNS varchar AS $$
DECLARE
    album_id_before albums.album_id%TYPE;
BEGIN
        SELECT get_album_id_most_sales() INTO album_id_before;
        PERFORM pg_sleep(10);
        UPDATE albums SET sales = 1 WHERE (albums.album_id = album_id_before);
        RETURN 'UPDATE SUCCESSFUL';
END;
$$ LANGUAGE plpgsql;
```

INDUCING LOST UPDATE PHENOMENA - OUTPUTS

SHELL 1
Transaction 1

SHELL 2
Transaction 2

Update function output

```
 update_sales_Q2
---------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 update_sales_Q2
---------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

First SELECT query output

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

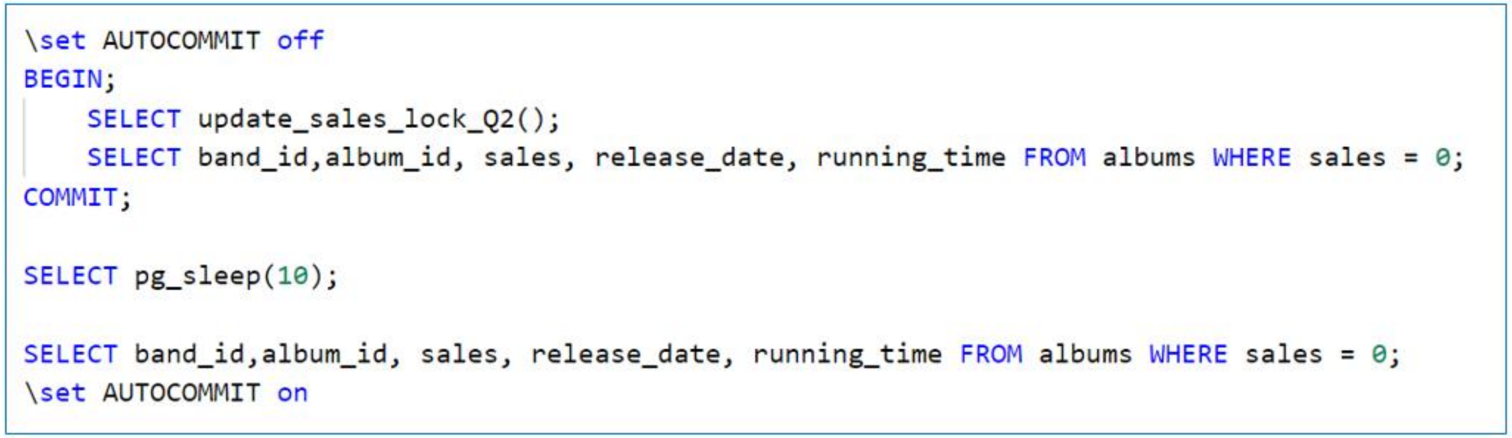
Second SELECT query output

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334
    464 |     1338 |     0 | 2015-06-11   |     79.13333
(2 row)
```

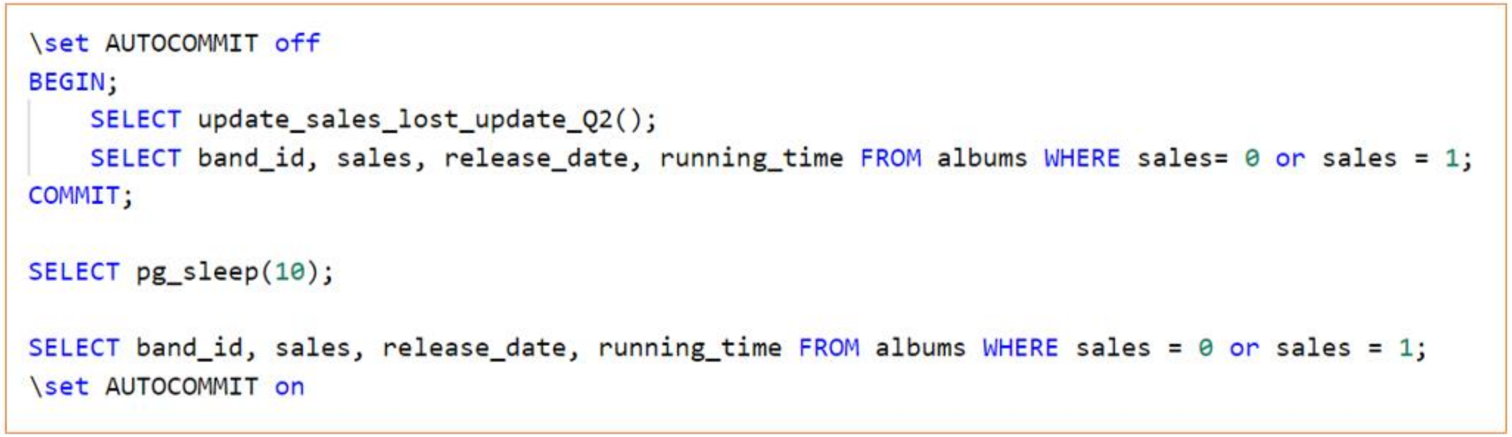
SOLVING LOST UPDATE PHENOMENA - LOCKS

SHELL 1
Transaction 1



```
\set AUTOCOMMIT off
BEGIN;
    SELECT update_sales_lock_Q2();
    SELECT band_id,album_id, sales, release_date, running_time FROM albums WHERE sales = 0;
COMMIT;

SELECT pg_sleep(10);

SELECT band_id,album_id, sales, release_date, running_time FROM albums WHERE sales = 0;
\set AUTOCOMMIT on
```

SHELL 2
Transaction 2



```
\set AUTOCOMMIT off
BEGIN;
    SELECT update_sales_lost_update_Q2();
    SELECT band_id, sales, release_date, running_time FROM albums WHERE sales= 0 or sales = 1;
COMMIT;

SELECT pg_sleep(10);

SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0 or sales = 1;
\set AUTOCOMMIT on
```

SOLVING LOST UPDATE PHENOMENA - LOCKS - OUTPUTS

SHELL 1
Transaction 1

SHELL 2
Transaction 2

Update function output

```
 update_sales_lock_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

```
 update_sales_lost_update_Q2
--------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

First SELECT query output

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

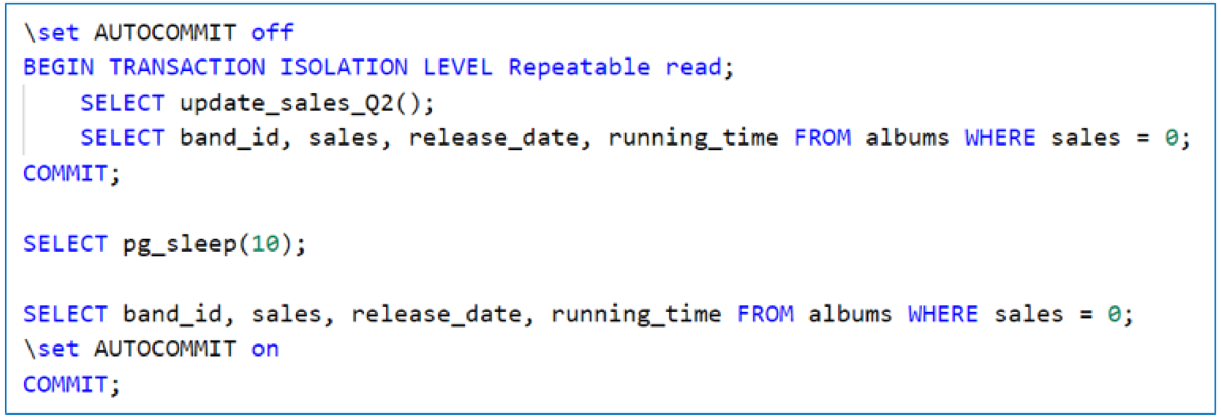
Second SELECT query output

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334

(1 row)
```

```
band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+-------------
    464 |     1280 |     0 | 2015-06-11   |    76.933334
    464 |     1338 |     1 | 2015-06-11   |     79.13333
(2 row)
```

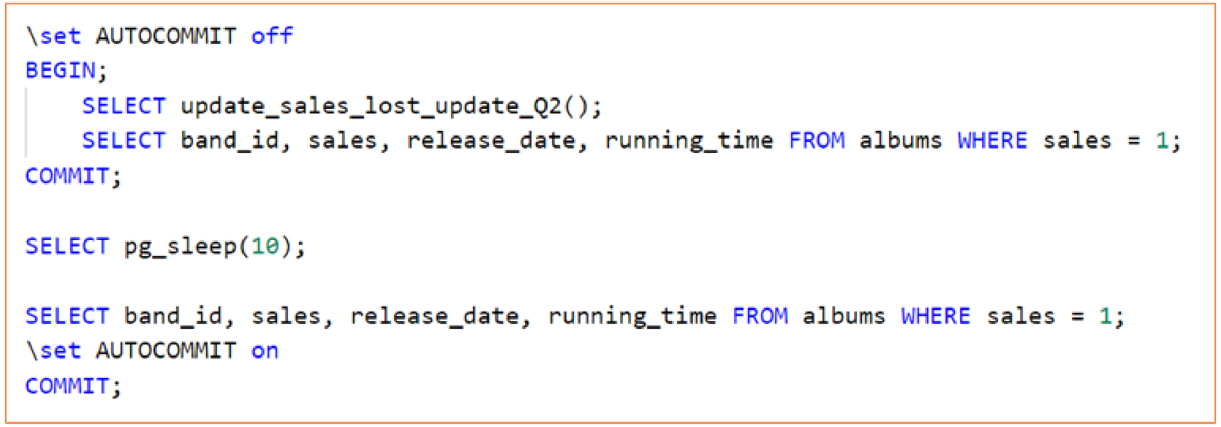
SOLVING LOST UPDATE PHENOMENA - ISOLATION LEVELS

SHELL 1
Transaction 1



```
\set AUTOCOMMIT off
BEGIN TRANSACTION ISOLATION LEVEL Repeatable read;
    SELECT update_sales_Q2();
    SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0;
COMMIT;

SELECT pg_sleep(10);

SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 0;
\set AUTOCOMMIT on
COMMIT;
```

SHELL 2
Transaction 2

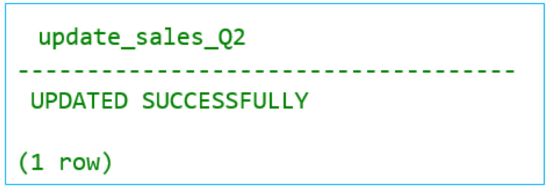


```
\set AUTOCOMMIT off
BEGIN;
    SELECT update_sales_lost_update_Q2();
    SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 1;
COMMIT;

SELECT pg_sleep(10);

SELECT band_id, sales, release_date, running_time FROM albums WHERE sales = 1;
\set AUTOCOMMIT on
COMMIT;
```

SOLVING LOST UPDATE PHENOMENA - ISOLATION LEVELS

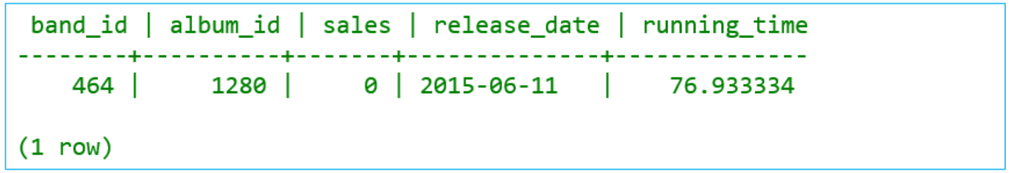
OUPUTS

SHELL 1
Transaction 1



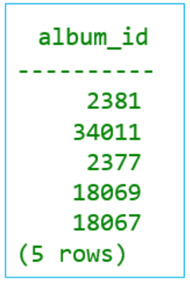
```
 update_sales_Q2
------------------------------------
 UPDATED SUCCESSFULLY

(1 row)
```

Update function output



```
 band_id | album_id | sales | release_date | running_time
--------+----------+-------+--------------+--------------
    464 |     1280 |     0 | 2015-06-11    |    76.933334

(1 row)
```

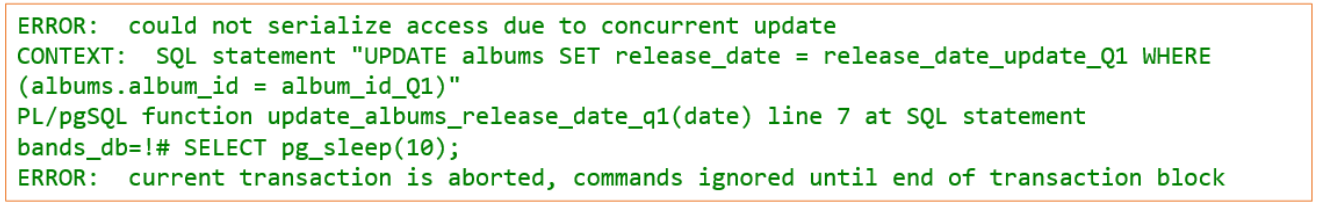
First SELECT query output



```
 album_id
----------
     2381
    34011
     2377
    18069
    18067
(5 rows)
```

Second SELECT query output

SHELL 2
Transaction 2



```
ERROR:  could not serialize access due to concurrent update
CONTEXT:  SQL statement "UPDATE albums SET release_date = release_date_update_Q1 WHERE
(albums.album_id = album_id_Q1)"
PL/pgSQL function update_albums_release_date_q1(date) line 7 at SQL statement
bands_db=!# SELECT pg_sleep(10);
ERROR:  current transaction is aborted, commands ignored until end of transaction block
```

FINAL REMARKS

- *Lock-Based Concurrency Control or Isolation Levels* implementations can sometimes have a negative impact on the database performance

- The most coherent approach to use in our case is the explicit locks, as it does not demonstrate any implementation concerns in contrast to the isolation levels, and proved to be the best option to ensure a better availability of the database

- It is necessary to evaluate the trade-offs between data consistency and concurrency, considering what use the database will be given.