- Listening on a topic requires its own connection and this connection cannot be used for anything else.
- NOTIFY is asynchronous so it is much faster to use within a trigger. It just sends a notification to a daemon which records them.
- If the listener is stopped then we will lose notifications.
- If you want a reliable system along those lines than you need to use SKIP LOCKED to SELECT one row to lock, then process it, and *then* DELETE the row. If your process dies then the lock will be release. You still have a new flavor of the same problem: you might process a message twice because the process might die in between completing processing and deleting the row. You could add complexity: first use SKIP LOCKED to SELECT one row to UPDATE to mark in-progress and LOCK the row, then later if the process dies another can go check if the job was performed (then clean the garbage) or not (pick and perform the job) -- a two-phase commit, essentially.
- `SKIP LOCKED` apply only to the row-level lock(s) — the required `ROW SHARE` table-level lock is still taken in the ordinary way

What ROW SHARE Does

When you execute `SELECT ... FOR UPDATE SKIP LOCKED`, PostgreSQL acquires:

1. **Table-level lock**: ROW SHARE on the entire table
2. **Row-level locks**: Exclusive locks on individual rows (unless skipped)

The ROW SHARE table lock:

- **Allows**: Other transactions to read the table and acquire their own ROW SHARE locks
- **Allows**: Other transactions to lock individual rows (SELECT FOR UPDATE/SHARE)
- **Blocks**: Only `EXCLUSIVE` and `ACCESS EXCLUSIVE` table-level locks (like `LOCK TABLE ... IN EXCLUSIVE MODE` or `DROP TABLE`)

- The same channel name is signaled multiple times from the same transaction with identical payload strings, the database server can decide to deliver a single notification only. On the other hand, notifications with distinct payload strings will always be delivered as distinct notifications. Similarly, notifications from different transactions will never get folded into one notification. Except for dropping later instances of duplicate notifications, NOTIFY guarantees that notifications from the same transaction get delivered in the order they were sent. It is also guaranteed that messages from different transactions are delivered in the order in which the transactions committed.

- It is crucial that an application using the PostgreSQL notification capabilities are capable of missing events. Notifications are only sent to connected client connections.
-