- PG Practise tips:

> *Dont do it*
>
> *PG Internal Post*

- check current database's isolation level

```
  select current_setting('transaction_isolation');
  -- or
  show transaction_isolation;
```

- Acquire a Snapshot

```
  -- get output like:  xmin:xmax:list of active transaction ids at
the time of the snapshot
  -- e.g. 100:104:100,102
  select pg_current_snapshot();
```

- set isolation level within a transaction

```
 begin;
 set transaction isolation level repeatable read ;
 show transaction_isolation;
 commit;
```

- How to check pages a table contains

```
  -- pg_relpages is in the built-in extension of pgstattuple
  create extension pgstattuple ;
  SELECT pg_relpages('x');
```

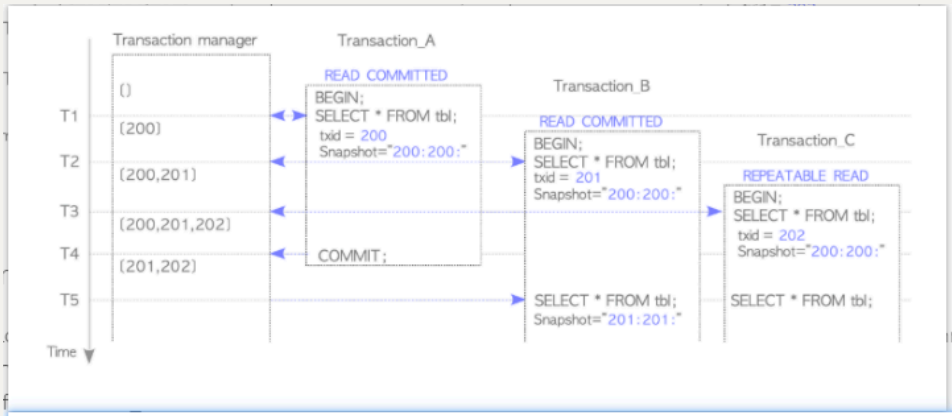
- Table Logical Architecture
    - pages
    - tuples (heap tuple)

- mvcc

> *Due to PostgreSQL's MVCC, when you update a row, PostgreSQL doesn't overwrite the existing tuple.*
>
> *Instead, it creates a new tuple with the updated data*
>
> *so a single logical "row" can have multiple physical "tuples" associated with it, representing its different versions over time*

- SI: snapshot isolation
- Transaction manager: within each transaction, the manager only focus on active transaction
- invisible check: dependent on active transaction and the SI level .
- **almost all tuples within a PostgreSQL table's pages are related to a transaction ID (XID)**

> *PostgreSQL's Multi-Version Concurrency Control (MVCC) relies heavily on transaction IDs to manage concurrent data access.*



ree
COMMITTED,

The transaction manager always holds information about currently running transactions. Suppose that three transactions start one after another, and the isolation level of Transaction_A and Transaction_B are READ COMMITTED, and that of Transaction_C is REPEATABLE READ.

- **T1:**
  Transaction_A starts and executes the first SELECT command. When executing the first command, Transaction_A requests the txid and snapshot of this moment.
  In this scenario, the transaction manager assigns txid 200, and returns the transaction snapshot '200:200:'.
- **T2:**
  Transaction_B starts and executes the first SELECT command. The transaction manager assigns txid 201, and returns the transaction snapshot '200:200:' because Transaction_A (txid 200) is in progress. Thus, Transaction_A cannot be seen from Transaction_B.
- **T3:**
  Transaction_C starts and executes the first SELECT command. The transaction manager assigns txid 202, and returns the transaction snapshot '200:200:', thus, Transaction_A and Transaction_B cannot be seen from Transaction_C.
- **T4:**
  Transaction_A has been committed. The transaction manager removes the information about this transaction.
- **T5:**
  Transaction_B and Transaction_C execute their respective SELECT commands.
  Transaction_B requires a transaction snapshot because it is in the READ COMMITTED level.
  In this scenario, Transaction_B obtains a new snapshot '201:201:' because Transaction_A (txid 200) is committed. Thus, Transaction_A is no longer invisible from Transaction_B.
  In contrast, Transaction_C does not require a transaction snapshot because it is in the REPEATABLE READ level and uses the obtained snapshot, i.e. '200:200:'. Thus, Transaction_A is still invisible from Transaction_C.