Transaction

Standard form

```
BEGIN;-- or BEGIN TRANSACTION
UPDATE accounts SET balance = balance - 100.00
 WHERE name = 'zhangsan';
-- etc etc
COMMIT;--or COMMIT TRANSACTION or ROLLBACK
```

PostgreSQL actually treats every SQL statement as being executed within a transaction. If you do not issue a `BEGIN` command, then each individual statement has an implicit `BEGIN` and (if successful) `COMMIT` wrapped around it. A group of statements surrounded by `BEGIN` and `COMMIT` is sometimes called a *transaction block*.

Notice: Some client libraries issue `BEGIN` and `COMMIT` commands automatically, so that you might get the effect of transaction blocks without asking. Check the documentation for the interface you are using.

we use follow table

```
create table emp(
    id integer primary key ,
    name varchar(30),
    age integer,
    dept_code integer,
    office_loc varchar(100),
    salary integer default 0
);

insert into emp values (1,'张三',22,1,'宝安',6000),
                       (2,'李四',31,1,'宝安',7000),
                       (3,'王五',25,2,'福田',6000),
                       (4,'赵六',24,1,'宝安',5000),
                       (5,'庄七',22,3,'光明',8000),
                       (6,'康八',45,2,'福田',15000),
                       (7,'聂九',34,3,'光明',7500),
                       (8,'刘二麻子',56,4,'光明',17000),
                       (9,'孙小毛',17,1,'宝安',3000),
                       (10,'陈老大',37,1,'宝安',7000);
```

Feature ACID（Atomicity, Consistency, Isolation, Durability)

- **Atomicity**: In a transaction involving two or more discrete pieces of information, either **all** of the pieces are committed or **none** are.
- **Consistency**: A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.

不同事务之间相互独立 相互隔离地并发工作

- **Isolation**: A transaction in process and not yet committed must remain isolated from any other transaction.

  The SQL standard defines four levels of transaction isolation:

  连续两次查询结果不一致：
  1、不可重复读
  2、幻读
  某个行内信息被改变 多半来自于update

  一个事务读取了另一个事务未提交的修改

  `dirty read`: A transaction reads data written by a concurrent uncommitted transaction.

  一个数据在连续两次读取时结果不同

  `nonrepeatable read`: A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).

  `phantom read`: A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.

  某个集合的整体性质被改变
  多半来自于insert delete
  产生的行上的修改

  `serialization anomaly`: The result of successfully committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

  You can set above level in Postgres using **SET TRANSACTION**

  https://www.postgresql.org/docs/12/sql-set-transaction.html

- **Durability**: Committed data is saved by the system such that, even in the event of a failure and system restart, the data is available in its correct state.

Isolation Level

越高的隔离级别 越高地保证数据有效性 但同时越会降低并发事务的执行速率

Isolation Level	Dirty Read	Nonrepeatable Read	Phantom Read	Serialization Anomaly
Read uncommitted	Allowed, but not in PG	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible	Possible
Repeatable read	Not possible	Not possible	Allowed, but not in PG	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

不能读取尚未提交的事务

会在尝试修改已经读取过的数据时产生报错 保证了连续读取同一个数据时该数据不会被修改

In PostgreSQL, you can request any of the four standard transaction isolation levels, but internally only three distinct isolation levels are implemented, i.e., PostgreSQL's Read Uncommitted mode behaves like Read Committed. This is because it is the only sensible way to map the standard isolation levels to PostgreSQL's multi-version concurrency control architecture.

- Read Committed Isolation Level

  Read Committed is the **default** isolation level in PostgreSQL.. When a transaction uses this isolation level, a `SELECT` query (without a `FOR UPDATE/SHARE` clause) sees only data committed before the query began; it never sees either uncommitted data or changes committed during query execution by concurrent transactions. In effect, a `SELECT` query sees a snapshot of the database as of the instant the query begins to run. However, `SELECT` does see the effects of previous updates executed within its own transaction, even though they are not yet committed. Also note that two successive `SELECT` commands can see different data, even though they are within a single transaction, if other transactions commit changes after the first `SELECT` starts and before the second `SELECT` starts.

- Repeatable Read Isolation level

The *Repeatable Read* isolation level only sees **data committed before the transaction began**; it never sees either uncommitted data or changes committed during transaction execution by concurrent transactions.

This level is different from Read Committed in that a query in a repeatable read transaction sees a snapshot as of the start of the first non-transaction-control statement in the *transaction*, not as of the start of the current statement within the transaction. Thus, successive `SELECT` commands within a *single* transaction see the same data, i.e., they do not see changes made by other transactions that committed after their own transaction started.

Applications using this level must be prepared to retry transactions due to serialization failures.

Maybe...

```
ERROR:  could not serialize access due to concurrent update
```

because a repeatable read transaction cannot modify or lock rows changed by other transactions after the repeatable read transaction began.

When an application receives this error message, it should abort the current transaction and retry the whole transaction from the beginning. The second time through, the transaction will see the previously-committed change as part of its initial view of the database, so there is no logical conflict in using the new version of the row as the starting point for the new transaction's update.

Note that only updating transactions might need to be retried; read-only transactions will never have serialization conflicts.

- Serializable Isolation Level

  The *Serializable* isolation level provides the strictest transaction isolation. This level emulates serial transaction execution for all committed transactions; as if **transactions had been executed one after another**, serially, rather than concurrently. However, like the Repeatable Read level, applications using this level must be prepared to retry transactions due to serialization failures. In fact, this isolation level works exactly the same as Repeatable Read except that it monitors for conditions which could make execution of a concurrent set of serializable transactions behave in a manner inconsistent with all possible serial (one at a time) executions of those transactions. This monitoring does not introduce any blocking beyond that present in repeatable read, but there is some overhead to the monitoring, and detection of the conditions which could cause a *serialization anomaly* will trigger a *serialization failure*.

```
ERROR:  could not serialize access due to read/write dependencies among
transactions
```

Set Transaction 修改当前使用的隔离级别

```
SET TRANSACTION transaction_mode [, ...]
```

transaction_mode:

READ COMMITTED--A statement can only see rows committed before it began. This is the default.

REPEATABLE READ--All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

SERIALIZABLE--All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If a pattern of reads and writes among concurrent serializable transactions would create a situation which could not have occurred for any serial (one-at-a-time) execution of those transactions, one of them will be rolled back with a `serialization_failure` error.

Show the level of transaction isolation

```
show transaction_isolation;
```

Set default isolation level

```
set default_transaction_isolation = '[isolation level]';
```

Experiment 1: Atomicity

```
BEGIN;
DELETE from emp where age <17;
select * from emp where age <17;
COMMIT ;
select * from emp where age <17;

BEGIN;
DELETE from emp where name = '张三';
select * from emp where name = '张三';
ROLLBACK;
select * from emp where name = '张三';

BEGIN TRANSACTION ;
UPDATE emp SET salary=salary+1000 where id=5;
SAVEPOINT sp01;
UPDATE emp SET salary=salary-1000 where id=6;
select * from emp;
ROLLBACK TO sp01;
select * from emp;
UPDATE emp SET salary=salary-1000 where id=7;
select * from emp;
COMMIT ;
select * from emp;
```

默认的隔离级别是read commited
仅delete之后 在本控制台查询不到 但其他的控制台还能查询到
但commit之后 其他控制台不能在查询到
（不能在一个控制台里开两个transaction）

回滚回事务开始的状态
rollback之后 能够查询到‘ 张三 ’

建立存档点 但并未提交

Experiment 2: Consistency

```
select * from emp;

BEGIN TRANSACTION ;
UPDATE emp SET salary=salary+1000 where id=5;
SAVEPOINT sp01;
UPDATE emp SET salary=salary-1000 where id=6;
select * from emp;
ROLLBACK TO sp01;
select * from emp;

-- close this console, and check the table in database
select * from emp;
```

事务退出 尚未commit 回自动回滚到开始的时候
但并未结束 后续进行其他操作会发现处于占用状态

Experiment 3: Durability

```
select * from emp;

BEGIN TRANSACTION ;
TRUNCATE emp;
COMMIT;
select * from emp;--empty table

ROLLBACK;

select * from emp;--still empty
```

清空表

Experiment 4: Isolation(read committed)

Step 1: Open two console

Step 2: Execute SQL as following order, and check the table emp

order	Console 1	Console 2
1	BEGIN TRANSACTION ;	BEGIN TRANSACTION ;
2	insert into emp values (1,'张三',22,1,'宝安',6000);	
3	select * from emp; 查得到	select * from emp; 查不到
4	COMMIT;	
5	select * from emp; 查得到	select * from emp; 查得到
6		insert into emp values (2,'李四',31,1,'宝安',7000);
7		COMMIT;
8	select * from emp; 查得到*2	~~select * from emp where id=5;~~

select * from emp; 查得到*2

Experiment 5: Isolation(repeatable read)

Step 1: Query current isolation level

```
show default_transaction_isolation;
```

Step 2:Set isolation level

```
set default_transaction_isolation='repeatable read';
```

Step 3: Open two console

Step 4: Execute SQL as following order, and check the table emp

order	Console 1	Console 2
1	set default_transaction_isolation='repeatable read';	set default_transaction_isolation='repeatable read';
2	BEGIN TRANSACTION ;	BEGIN TRANSACTION ;
3	select * from emp;	select * from emp;
4	insert into emp values (3,'王五',25,2,'福田',6000);	
5	select * from emp; 查得到‘王五’	select * from emp; 查不到‘王五’
6	COMMIT;	
7	select * from emp; 查得到‘王五’	select * from emp; 依然查不到‘王五’：可重复读 在console2中连续两次查询结果应该相同
8		COMMIT;
9	select * from emp;	select * from emp; 查得到‘王五’

Experiment 6: Isolation(SERIALIZABLE)

Step 1: Query current isolation level

```
show default_transaction_isolation;
```

Step 2:Set isolation level

```
set default_transaction_isolation='serializable';
```

Step 3: Open two console

Step 4: Execute SQL as following order, and check the table emp

order	Console 1	Console 2
1	set default_transaction_isolation='serializable';	set default_transaction_isolation='serializable';
2	BEGIN TRANSACTION ;	BEGIN TRANSACTION ;
3	select * from emp;	select * from emp;
4	insert into emp values (4,'赵六',24,1,'宝安',5000);	
5	select * from emp;	select * from emp;
6	COMMIT;	
7		insert into emp values (5,'庄七',22,3,'光明',8000); 产生报错 无法执行
8		COMMIT;
9	select * from emp;	select * from emp;

如果在这里select* console2中不会查询到‘赵六’

能查到‘赵六’ 不会查到‘庄七’

*Here will be Error after Step 7

[40001] ERROR: could not serialize access due to read/write dependencies among transactions
Detail: Reason code: Canceled on identification as a pivot, during write. Hint: The transaction might succeed if retried.

会形成一个①<==>②的环

*Tips: Do not forget to recover to default level