In Postgres, when we create a table, it is stored as an array of pages of a fixed size (usually 8Kb). The structure used to store the table in a page is called heap. To alleviate some well-known problems of heap structure, we've come up with a new structure, called 'zheap'.

Why is it called 'zheap'?
The nomenclature doesn't have anything to do with what it does, it's because 'z' looks cool as a prefix and we usually do cool stuffs.

Why do we need a new structure?
Postgres uses MVCC for its concurrency control management. So, when we delete a row, it is not actually deleted, only marked as unavailable to all future transactions. Same is true for updates, the existing row is marked as unavailable to all future transactions and a new row is inserted. Postgres comes along with another operation, called vacuum, which reclaims these unavailable spaces for future usage. But, it doesn't/can't reclaim the space unconditionally. When it fails to reclaim the space, we've a problem and that problem is called ***bloat***. Postgres users start getting nightmares when the *bloat* problem goes out of hand. There are several online documentation [2] explaining how bloats can hurt the database performance.

The overall goal for inventing a new structure backed by undo logging (it is different from redo WAL logging) is to achieve better control over bloat [1]. For an example, when we update a row, we can update the row in the same place (conditionally) in the page and store the old tuple in undo. Any old transaction can still get the old tuple from undo. By storing old tuple versions in undo logs rather than in the page itself, it becomes possible to discard old row versions without reorganizing the page.

How should we test 'zheap'?
From a user's perspective, the only thing he should be doing is to create a table in zheap storage. Syntax for the same:
CREATE TABLE …….WITH (..,storage_engine='zheap',...)
Once that's done, all user level behavior should be same.

Can we test everything now?
Sadly, NO. We're working on getting everything to work and developing cool things take time. Here is a list of developed features and a way to test the same.

Modules	Features	Testing	Comments
Relation as zheap storage	Create Table		Unlogged, Toast tables don’t work yet
	Constraints		foreign keys doesn't work yet
	Insert		INSERT..ON CONFLICT don’t work yet
	Inplace Update		
	Non-inplace* update		
	Delete		
	SELECT		SELECT FOR UPDATE/FOR SHARE, Table Sample scans not yet implemented
	CTE		
	Views (Simple & Materialized)		INSERT...ON CONFLICT and IndexOnly Scan is not yet implemented.
	TRIGGERS		
	VALUES		
	RETURNING		
	COPY command	Check whether COPY FROM/TO zheap tables working	
	Parallelism		
Transactions	MVCC	Check whether snapshots are working properly	Only READ-COMMITTED isolation level is implemented
	EvalPlanQual	Check whether different transactions can update the same row at once	

	ROLLBACK	Check Rollback and Rollback to savepoint.	It doesn't work in case of any errors. Subtransactions not yet implemented.
Index	Create Index	Check all indexes are working	Concurrent Index creation is not yet implemented
	Index Scan	Check Index Scans and Bitmap scans	Index only scans are yet to be implemented
Vacuum		It should skip all zheap relations	Analyze is not yet implemented
Recovery		Check standby and crash recovery work	
Functions and Procedures		All test cases should be passed	
Other test frameworks	Make Check		Find zheap related test cases in regress/sql/zheap.sql
	Make Check-World		
	Isolation Tests		regress/isolation/zheap_mvv.spec
	pgbench	Single client/Multi Client for 30 minutes ● TPS ● Latency ● Table sizes	
	Table sizes	Comparison of table sizes between heap and zheap as storage	

*Non-inplace updates - Sizes of old and new values should not differ, update only non-key columns.

1. You can create zheap table by mentioning storage_engine='zheap' as reloption/ set storage_engine='zheap' in postgresql.conf.

REFERENCES

[1] https://docs.google.com/document/d/1EXDzmQqKe6ZmPhfQ3mcu49sNdXXfzszevZGsnMroFa4/edit
[2] https://www.keithf4.com/checking-for-postgresql-bloat/
[3] http://rachbelaid.com/introduction-to-postgres-physical-storage/