ACID Transactions.
ACID is an acronym that refers to the set of 4 key properties that define

a transaction: Atomicity, Consistency, Isolation,
and Durability.

What is a
transaction?
In the context of databases and data storage systems, a transaction is any
operation that is treated as a single unit of work,
which either completes fully or does not
complete at all, and leaves the storage system in a consistent state.
The classic example of
a transaction is what occurs when you withdraw money from your bank account.
Either the money
has left your bank account, or it has not — there cannot be an in-between
state.

A.C.I.D. properties: Atomicity, Consistency, Isolation, and Durability
ACID is an
acronym that refers to the set of 4 key properties that define a transaction: Atomicity,

Consistency, Isolation, and Durability. If a database operation has these ACID properties,

it can be called an ACID transaction, and data storage systems that apply these operations are

called transactional systems. ACID transactions guarantee that each read, write, or
modification
of a table has the following properties:

Atomicity - each statement in a
transaction (to read, write, update or delete data) is treated as a
single unit. Either the
entire statement is executed, or none of it is executed. This property prevents
data loss and
corruption from occurring if, for example, if your streaming data source fails
mid-stream.
Consistency - ensures that transactions only make changes to tables in predefined,
predictable ways.
Transactional consistency ensures that corruption or errors in your data do
not create unintended
consequences for the integrity of your table.
Isolation - when
multiple users are reading and writing from the same table all at once,
isolation of their
transactions ensures that the concurrent transactions don't interfere
with or affect one
another. Each request can occur as though they were occurring one by
one, even though
they're actually occurring simultaneously.
Durability - ensures that changes to your data made
by successfully executed transactions
will be saved, even in the event of system
failure.

Why are ACID transactions a good thing to have?
ACID transactions ensure the
highest possible data reliability and integrity.
They ensure that your data never falls into
an inconsistent state because of
an operation that only partially completes. For example,
without ACID transactions,
if you were writing some data to a database table, but the power
went out unexpectedly,
it's possible that only some of your data would have been saved,
while some of it would not.
Now your database is in an inconsistent state that is very
difficult and time-consuming to recover from.

Delta Lake: Reliable, consistent data with

the guarantees of ACID transactions
 ACID transactions have long been one of the most enviable
properties of data warehouses,
 but Delta Lake has now brought them to data lakes. They allow
users to see consistent
 views of their data even while new data is being written to the
table in real-time,
 because each write is an isolated transaction that is recorded in an
ordered transaction log.
  [Delta Lake employs the highest level of isolation possible
(serializable isolation),
  ensuring that reads and writes to a single table are consistent
and reliable.]
  By implementing ACID transactions, Delta Lake effectively solves for several
of
   the previously listed criticisms of Lambda architecture: its complexity,
    incorrect
views of data, and the rework and reprocessing needed after Lambda pipelines
    inevitably
break. Users can perform multiple concurrent transactions on their data,
    and in the event
of an error with a data source or a stream, Delta Lake cancels execution
     of the
transaction to ensure that the data is kept clean and intact. The beauty of ACID

transactions is that users can trust the data that is stored in Delta Lake. A data analyst

   making use of Delta Lake tables to perform ETL on his or her data to ready it for

dashboarding can count on the fact that the KPIs he or she is seeing represent
       the
actual state of the data. A machine learning engineer using Delta Lake tables
        to
perform feature engineering can be 100% confident that all of his or her

transformations and aggregations either executed exactly as intended, or
        didn't
execute at all (in which case, he or she would be notified).
        The value of knowing
that the mental model
 you have of your data is actually reflective of its true underlying
state cannot be overstated.