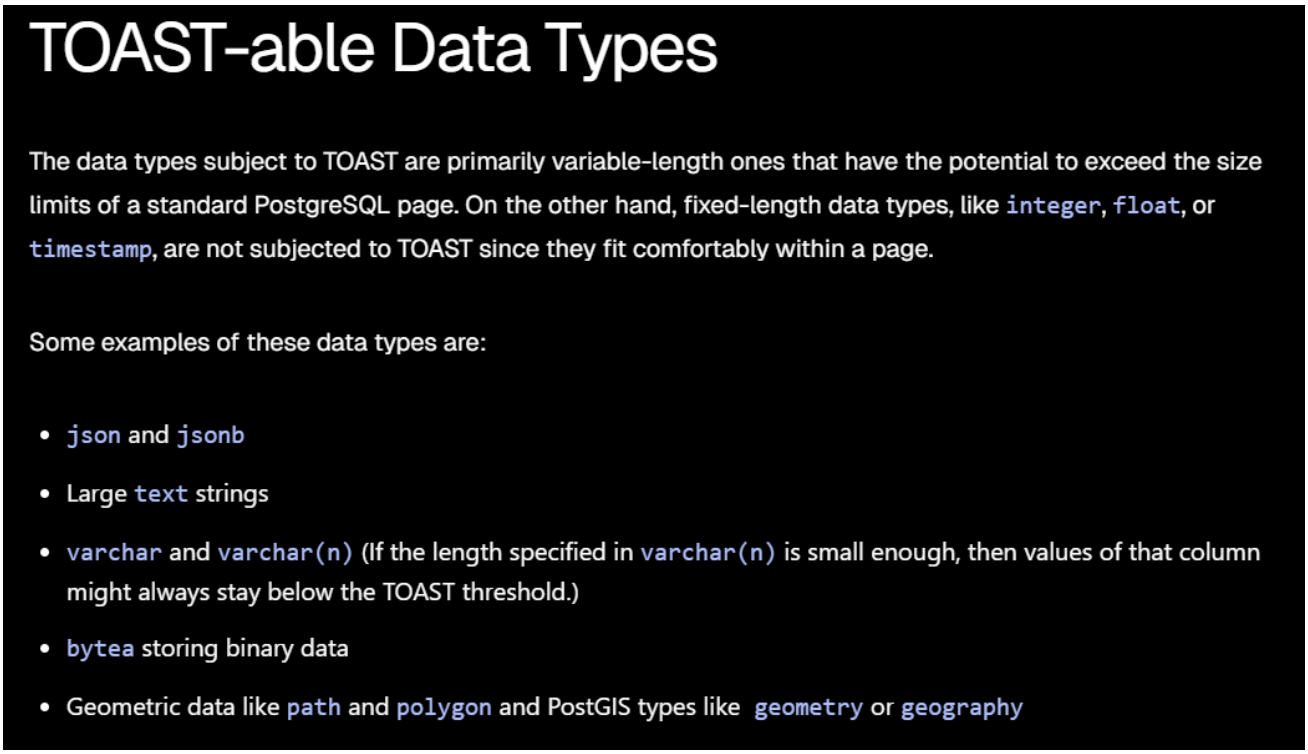
- TOAST is the automatic mechanism that PostgreSQL uses to store and manage large values that do not fit within individual database pages.
- Postgres' storage units are called pages, and pages have a fixed size (8 kB by default). Having a fixed page size gives Postgres many advantages: data management simplicity, efficiency, and consistency. But there is a downside: some data values might not fit within that page
- TOAST refers to the automatic mechanism that PostgreSQL uses to efficiently store and manage values in Postgres that do not fit within a page. To handle such values, Postgres TOAST will, by default, compress them using an internal algorithm. TOAST will attempt to squeeze the user-table row down to 2KB in this way.
- The `pglz` (PostgreSQL Lempel-Ziv) is the default internal compression algorithm used by PostgreSQL specifically tailored for TOAST. The compression rate effectiveness of `pglz` will largely depend on the nature of the data. For example, highly repetitive data will compress much better than high entropy data (like random data).
- If, after compression, the values are still too large, Postgres will move them to a separate table (called the TOAST table), leaving pointers in the original table.
- 

TOAST strategies

By default, PostgreSQL will go through the TOAST mechanism according to the procedure explained earlier (compression first and out-of-line storage next, if compression is not enough). Still, there might be scenarios where you might want to fine-tune this behavior on a per-column basis. PostgreSQL allows you to do this using the TOAST strategies `EXTENDED`, `EXTERNAL`, `MAIN`, and `PLAIN`.

- **`EXTENDED`:** This is the default strategy. Data will be stored out of line in a separate TOAST table if it's too large for a regular table page. Data will be compressed to save space before being moved to the TOAST table.
- **`EXTERNAL`:** This strategy tells PostgreSQL to store the data for this column out of line if the data is too large to fit in a regular table page, and we're asking PostgreSQL not to compress the data—the value will be moved to the TOAST table as-is.
- **`MAIN`:** This strategy is a middle ground. It tries to keep data in line in the main table through compression; if the data is definitely too large, it will move the data to the TOAST table to avoid an error, but PostgreSQL won't move the compressed data. Instead, it will store the value in the TOAST table in its original form.
- **`PLAIN`:** Using `PLAIN` in a column tells PostgreSQL to always store the column's data in line in the main table, ensuring it isn't moved to an out-of-line TOAST table. Take into account that if the data grows beyond the page size, the `INSERT` will fail because the data won't fit.

`TOAST_TUPLE_THRESHOLD` is the trigger point. When the size of a tuple's data fields combined exceeds this threshold, PostgreSQL will evaluate how to manage it based on the set TOAST strategy for its columns, considering compression and out-of-line storage. The exact actions taken will also depend on whether column data surpasses the `TOAST_COMPRESSION_THRESHOLD`.

Strategy	Compress if tuple > TOAST_COMPRESSION_THRESHOLD	Store out-of-line if tuple > TOAST_TUPLE_THRESHOLD	Description
EXTENDED	Yes	Yes	Default strategy. Compresses first, then checks if out-of-line storage is needed.
MAIN	Yes	Only in uncompressed form	Compresses first, and if still oversized, moves to TOAST table without compression.
EXTERNAL	No	Yes	Always moves to TOAST if oversized, without compression.
PLAIN	No	No	Data always stays in the main table. If a tuple exceeds the page size, an error occurs.

- While TOAST incorporates compression as one of its techniques to achieve this, TOAST's primary role isn't to optimize storage space across the board.
- TOAST can also introduce significant I/O overhead, especially for large tables with frequently accessed oversized columns. In such cases, PostgreSQL needs to retrieve the out-of-line data from the TOAST table, which is a separate I/O operation from accessing the

main table, as PostgreSQL must follow pointers from the main table to the TOAST table to read the complete data. This typically leads to worse performance.

- TOAST's compression is not designed to provide especially high compression ratios, as it uses one standard algorithm for all data types.