# 9.14. UUID Functions

PostgreSQL includes several functions to generate UUIDs (Universally Unique IDentifiers) as specified in [RFC 9562](https://datatracker.ietf.org/doc/html/rfc9562). This section describes those included in the core distribution.

The [uuid-ossp](https://www.postgresql.org/docs/devel/uuid-ossp.html) module provides additional functions that implement other standard algorithms for generating UUIDs.

PostgreSQL also provides the usual comparison operators shown in [Table 9.1](https://www.postgresql.org/docs/devel/functions-comparison.html#FUNCTIONS-COMPARISON-OP-TABLE) for UUIDs.

## 9.14.1. Generating Version 7 UUIDs

uuidv7 ([ *offset* interval ]) → uuid

Function uuidv7() is designed to be the default choice for generation of primary keys instead of integer data types, backed by a sequence generator.

The function returns a version 7 UUID, which includes a UNIX timestamp with millisecond precision, a 12-bit sub-millisecond timestamp, and a random component. This function can accept optional *offset* parameter of type interval which is added to the internal timestamp.

Monotonically increasing identifiers are generated even if the system clock jumps backward, or access to the system clock is unavailable, or UUIDs are generated at very high frequency, as the internal timestamp functions as a counter to maintain order.

If using the *offset* parameter results in timestamp overflow or a negative timestamp, an adjusted timestamp value is automatically used. The timestamp behaves like a ring buffer: when the maximum value is exceeded, it wraps around to the minimum value. Similarly, if the absolute value of the negative *offset* exceeds the time elapsed since 00:00:00 UTC on 1 January, 1970, the timestamp wraps around to the maximum value.

## 9.14.2. Generating Version 4 UUIDs

gen\_random\_uuid () → uuid

uuidv4 () → uuid

These functions return a version 4 (random) UUID.

They are not recommended for generation of primary keys.

## 9.14.3. Extracting Data from UUIDs

There are also two functions to extract data from UUIDs:

uuid\_extract\_timestamp (uuid) → timestamp with time zone

This function extracts a timestamp with time zone from UUID version 1 and 7. For other versions, the function returns null. Note that the extracted timestamp is not necessarily exactly equal to the date and time the UUID was generated; this depends on the implementation that generated the UUID.

uuid\_extract\_version (uuid) → smallint

This function extracts the version from a UUID of the variant described by RFC 9562. For other variants, the function returns null. For example, for a UUID generated by gen\_random\_uuid(), the function will return 4.

## 9.14.4. Deciding Whether and Which UUID to Use

UUIDs serve as unique identifiers. Alternatives include integer data types backed by a sequence generator. When choosing between them for primary keys, consider the following information.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Disadvantages or limitations of identifier types | uuidv4() | uuidv7() | uuidv7(*offset*) | IDENTITY or BIGSERIAL |
| Need to generate new primary keys when merging data | NO | NO | NO | YES |
| Need for synchronization in distributed generation across multiple processes (microservices) | NO | NO | NO | YES |
| Lock contention arises when multiple processes (microservices) write to the same table simultaneously | NO | YES | NO(with several offsets) | YES |
| Absence of identifier locality reduces performance and increases index size | YES | NO | NO | NO |
| Absence of identifier locality results in inefficient partitioning | YES | NO | NO | NO |
| The order of creation of records is unknown for logging systems, time-series databases, debugging, and auditing | YES | NO | NO(with nondecreasing offsets) | NO |
| The number of records in the table is disclosed | NO | NO | NO | YES |
| The creation date and time of the record are disclosed | NO | YES | NO | NO |
| Takes up more memory and disk space | YES | YES | YES | NO |
| Ambiguous full-text search for documents, files, and web pages by identifiers | NO | NO | NO | YES |

When generating identifiers simultaneously in several client sessions, the uuidv7() function does not guarantee monotonicity, although monotonicity is usually preserved in such a situation.

In real-world scenarios, the performance of keys generated by uuidv7() function is nearly equivalent to that of identity or bigserial type, significantly outperforming uuidv4().

It is advisable to assess the performance of keys generated by different methods using the [pgbench](https://www.postgresql.org/docs/devel/pgbench.html) benchmarking utility, along with custom scenarios and script files tailored to your specific requirements.