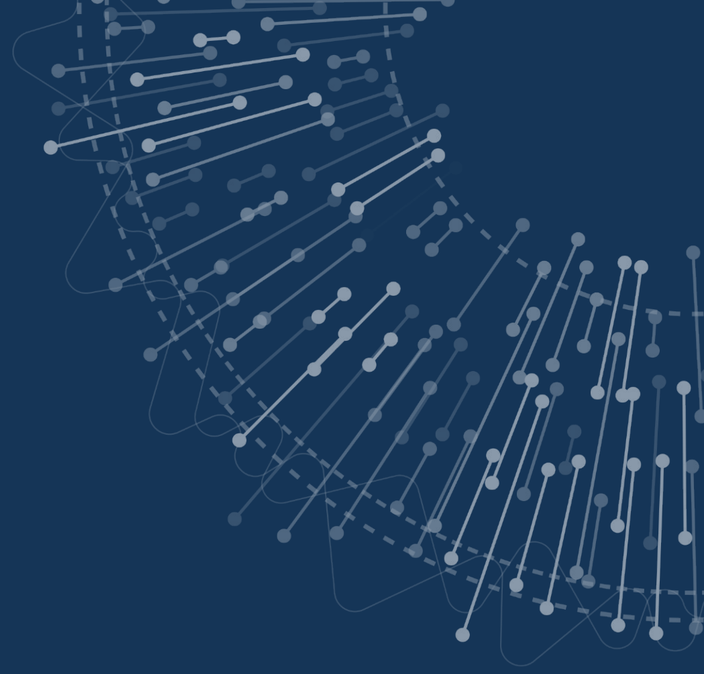


All In A Context!

Rafia Sabih

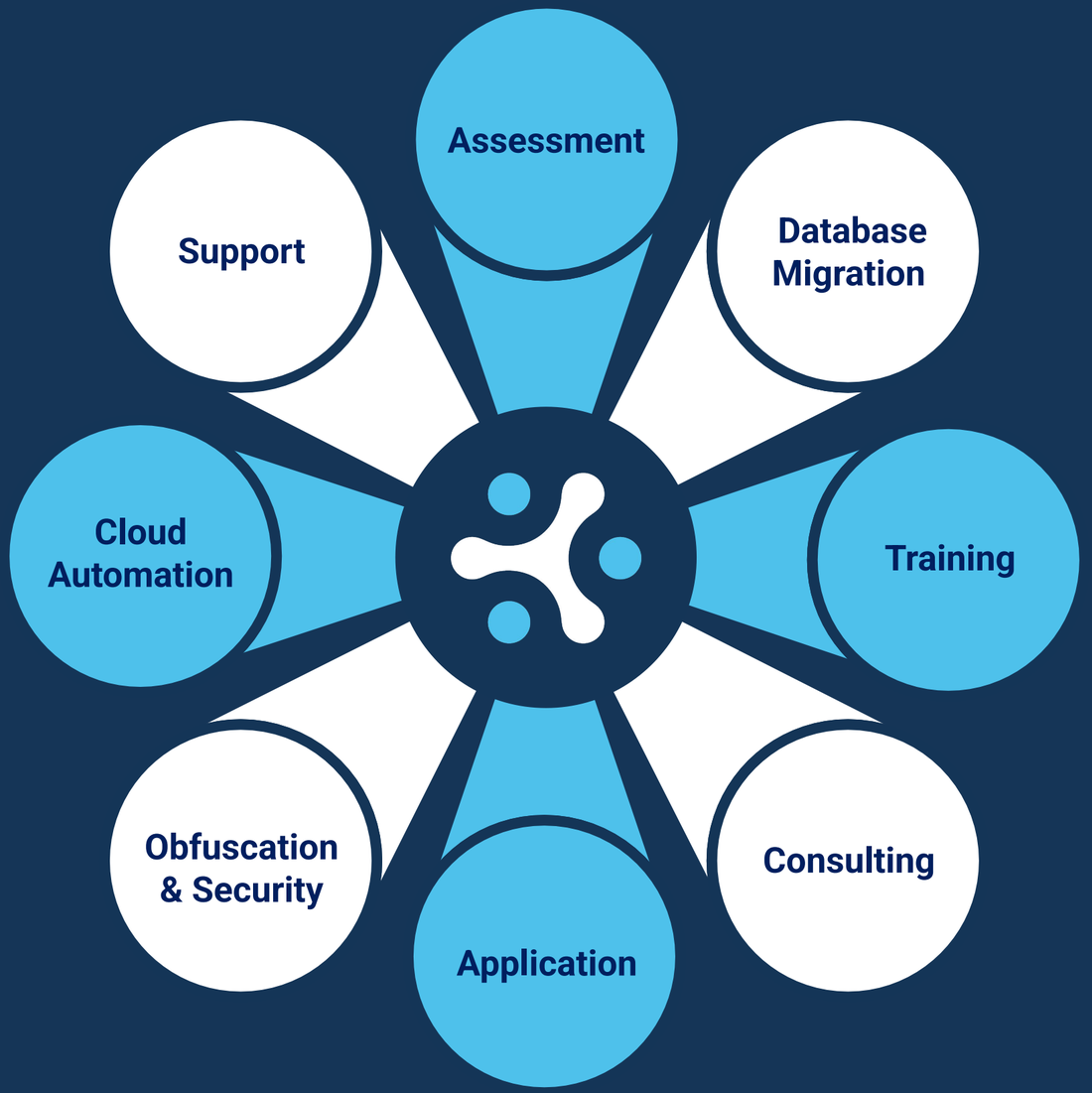
Sr Software Engineer

CYBERTEC

POSTGRESQL SERVICES & SUPPORT

Database Services

- 24/7 Support
- High Availability
- Consulting
- Performance Tuning
- Clustering
- Migration
- Compliance



ISO 27001 CERTIFIED

Database Products & Tools

CYBERTEC MIGRATOR

CYPEX

CYBERTEC PGEE

scalefield

PG WATCH

PG SQUEEZE

scalefield secure

CYBERTEC LakeHouse

YAIM

WAL BOUNCER

DATA MASKING

pg_ TIMETABLE

PATRONI ENVIRONMENT SETUP

POSTGRESQL CONFIGURATOR

PG SHOW PLANS

PL/pgSQL_sec CYBERTEC

ISO 27001 CERTIFIED

Our partners at PGConf.EU

readyset

redgate

OpenSource^DB
Your Trusted Data Partner

Microsoft

PROVENTA

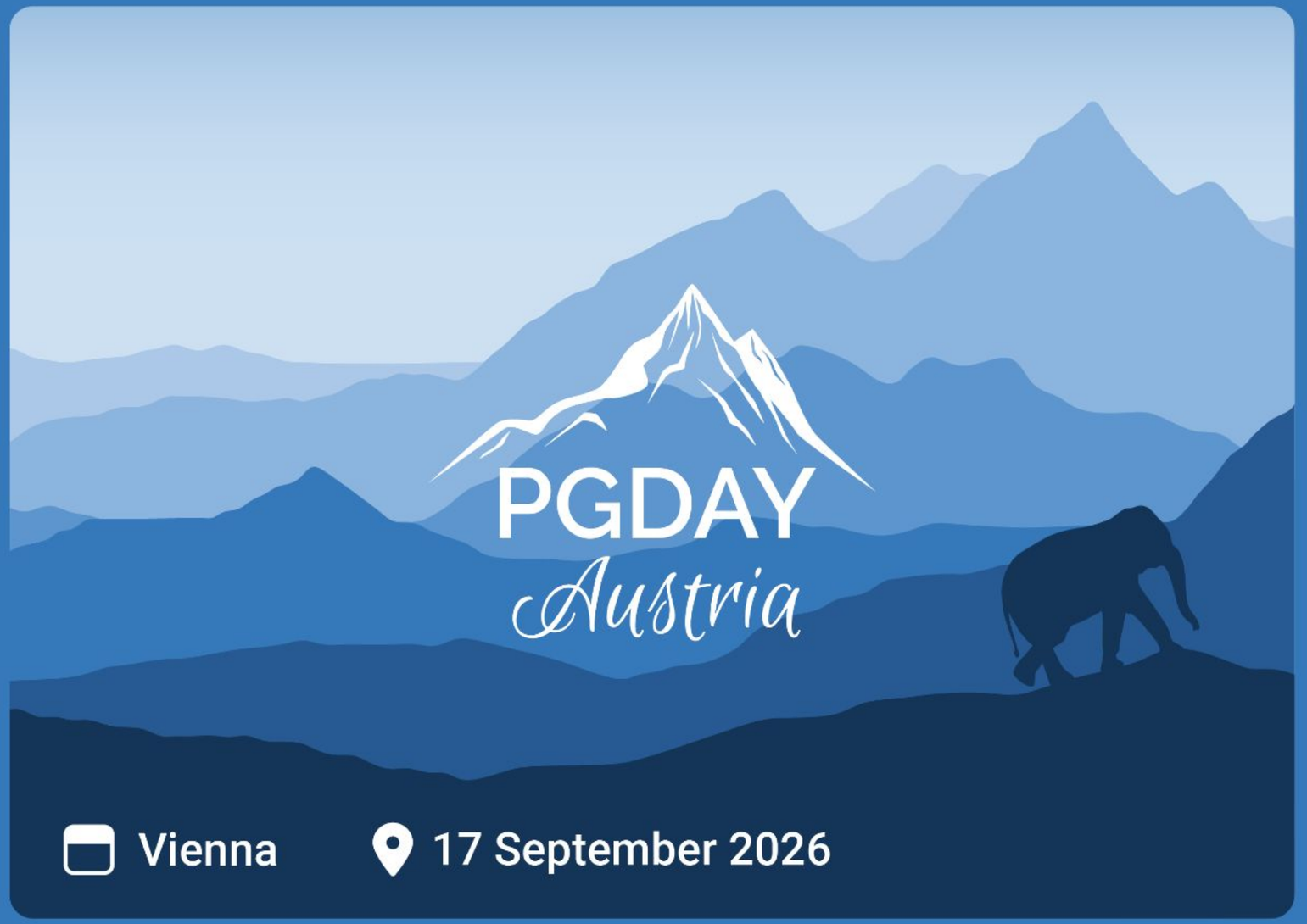
Open Alliance

For PostgreSQL Education

Your Pathway to Verified PostgreSQL Skills

Scan for Updates

oapg-edu.org



PGDay Austria returns in 2026

Scan for updates

pgday.at

Agenda

1. Why Memory Context
2. What Is A Memory Context
3. How To Use Them
4. Caveats

Memory Context

Why

- Tackle the memory management issue of C
  - Memory leaks
  - Absence of garbage collection
  - Heap fragmentation

Memory Context

What

- A logical context within which allocation occurs
- An abstract with multiple implementations
- Alloc sets of 8kb
  - at every realloc the size doubles
- Chunks of memory allocated/deallocated as requested

Memory Context

What

```
typedef struct AllocSetContext
{
   MemoryContextData header;/* Standard memory-context fields */
   /* Info about storage allocated in this context: */
   AllocBlock  blocks;       /* head of list of blocks in this set */
   MemoryChunk *freelist[ALLOCSET_NUM_FREELISTS];/* free chunk lists */
   /* Allocation parameters for this context: */
   uint32    initBlockSize;  /* initial block size */
   uint32    maxBlockSize; /* maximum block size */
   uint32    nextBlockSize;  /* next block size to allocate */
   uint32    allocChunkLimit;/* effective chunk size limit */
   /* freelist this context could be put in, or -1 if not a candidate: */
   int       freeListIndex;  /* index in context_freelists[], or -1 */
} AllocSetContext;
```

Memory Context

What

```
1| typedef struct AllocBlockData
2| {
3|   AllocSet aset;        /* aset that owns this block */
4|   AllocBlock  prev;        /* prev block in aset's blocks list, if any */
5|   AllocBlock  next;        /* next block in aset's blocks list, if any */
6|   char    *freeptr;     /* start of free space in this block */
7|   char    *endptr;      /* end of space in this block */
8| }      AllocBlockData;
```

Memory Context

What

```
 1| typedef struct MemoryContextData
 2| {
 3|    pg_node_attr(abstract)     /* there are no nodes of this type */
 4|
 5|    NodeTag     type;       /* identifies exact kind of context */
 6|    /* these two fields are placed here to minimize alignment wastage: */
 7|    bool     isReset;    /* T = no space allocated since last reset */
 8|    bool     allowInCritSection; /* allow palloc in critical section */
 9|    Size     mem_allocated; /* track memory allocated for this context */
10|    const MemoryContextMethods *methods;/* virtual function table */
11|    MemoryContext parent;   /* NULL if no parent (toplevel context) */
12|    MemoryContext firstchild;  /* head of linked list of children */
13|    MemoryContext prevchild;/* previous child of same parent */
14|    MemoryContext nextchild;/* next child of same parent */
15|    const char *name;       /* context name (just for debugging) */
16|    const char *ident;      /* context ID if any (just for debugging) */
17|    MemoryContextCallback *reset_cbs;/* list of reset/delete callbacks */
18| } MemoryContextData;
```

Memory Context

What

```
typedef struct MemoryContextMethods
{
    void     *(*alloc) (MemoryContext context, Size size);
    /* call this free_p in case someone #define's free() */
    void      (*free_p) (void *pointer);
    void     *(*realloc) (void *pointer, Size size);
    void      (*reset) (MemoryContext context);
    void      (*delete_context) (MemoryContext context);
    MemoryContext (*get_chunk_context) (void *pointer);
    Size      (*get_chunk_space) (void *pointer);
    bool      (*is_empty) (MemoryContext context);
    void      (*stats) (MemoryContext context,
                        MemoryStatsPrintFunc printfunc, void *passthru,
                        MemoryContextCounters *totals,
                        bool print_to_stderr);
#ifdef MEMORY_CONTEXT_CHECKING
    void      (*check) (MemoryContext context);
#endif
} MemoryContextMethods;
```

Memory Context

What

```
1| /*
2|  * Flags for MemoryContextAllocExtended.
3|  */
4| #define MCXT_ALLOC_HUGE     0x01 /* allow huge allocation (> 1 GB) */
5| #define MCXT_ALLOC_NO_OOM   0x02 /* no failure if out-of-memory */
6| #define MCXT_ALLOC_ZERO     0x04 /* zero allocated memory */
```

Memory Context

Freelists

- Instead of multiple creation and deletion
  - Some freed contexts are added to freelists
  - Works in LIFO
  - These are reset before added to freelist
  - Are already chained via their nextchild pointers
- Next invocation of palloc can get a chunk from the one in freelist

Memory Context

MemoryContext Callbacks

- to manage resources other than just palloc-ed block
  - closing files with tuplesort objects
  - releasing reference counts on long-lived cache objects that are held by some object within the context being reset
- done by registering a reset callback function for a context
- callbacks called in reverse order of registration
- callbacks to child context are called before parent context

Memory Context

Basic operations

- Create a context
- Allocate a chunk within a context
- Reallocate a chunk to increase/decrease its size
- Delete a context and free its memory
- Reset a context
- Inquire about total memory allocated in a context

Memory Context

Organisation

- Arranged in a forest like structure
- Child context may have a different lifetime than parent but not longer
- Freeing a context, recursively frees child contexts
- Every context (except TopMemoryContext) has a parent

Memory Context

Organisation

- TopMemoryContext – for things which are never cleared
  - PostmasterContext
  - CacheMemoryContext
  - MessageContext
  - TopTransactionContext
    - PortalContext
    - ErrorContext

Memory Context

Organisation

- Other important contexts
  - per query context
  - per tuple context
  - per plan node
  - aggregation context
  - ...

Memory Context

Current Context

- There is a always a CurrentMemoryContext global variable
- By default palloc allocates in this context
- However, palloc, repalloc, etc. can be called for any chunk irrespective of CurrentMemoryContext
  - Owning context will be invoked

Memory Context

Example: create

- AllocSetContextCreate(parent_context, context_name, ALLOCSET_DEFAULT_SIZES)
  - MemoryContext AllocSetContextCreateInternal(MemoryContext parent, const char *name, Size minContextSize, Size initBlockSize, Size maxBlockSize)
    - MemoryContextCreate(MemoryContext node, NodeTag tag, MemoryContextMethodID method_id, MemoryContext parent, const char *name)

Memory Context

Example: delete

- MemoryContextDelete(context)
- MemoryContextDeleteChildren(context)
  - MemoryContextDelete(context)
- MemoryContextCallResetCallbacks (context)
- MemoryContextSetParent(context, NULL)

Memory Context

Example: reset

- MemoryContextReset(context)
  - MemoryContextDeleteChildren(context)
    - MemoryContextDelete(context)
  - MemoryContextResetOnly (context)
  - MemoryContextCallResetCallbacks(context)

Memory Context

Other operations

- MemoryContextSwitchTo()
  - switch to a desired context
- MemoryContextAlloc()
  - to allocate a chunk in a context

Memory Context

Other operations

- MemoryContextAllocZero()
  - allocates and clears allocated memory
- MemoryContextSetParent()
  - changes the parent after creation
  - useful to change lifespan of context after creation
  - e.g. context in a transient context changed to go under CacheMemoryContext

Memory Context

Inquiry operations

- GetMemoryChunkContext()
- GetMemoryChunkSpace()
- MemoryContextIsEmpty()
  - not empty if children exists
- MemoryContextGetParent()
- MemoryContextMemAllocated()
- MemoryContextMemConsumed()

Memory Context

Debugging utilities

- MemoryContextStats()
  - Total, bytes in blocks, free chunks, used
- MemoryContextCheck()
  - check all the chunks in a context
  - check all the children

Memory Context

Interesting functions

- palloc, palloc0, palloc_extended
  - allocated in CurrentMemoryContext
- palloc_aligned
  - MemoryContextAllocAligned
  - returns aligned memory allocation
- pfree
  - blocks are added to freelists
- repalloc

Memory Context

Related view

- pg_backend_memory_contexts
  - one row for each memory context
  - can be read only by superusers or roles with the privileges of the pg_read_all_stats role

Memory Context

Related view

```
                    View "pg_catalog.pg_backend_memory_contexts"
    Column      |   Type    | Collation | Nullable | Default | Storage  | Description
----------------+-----------+-----------+----------+---------+----------+-------------
 name           | text      |           |          |         | extended |
 ident          | text      |           |          |         | extended |
 type           | text      |           |          |         | extended |
 level          | integer   |           |          |         | plain    |
 path           | integer[] |           |          |         | extended |
 total_bytes    | bigint    |           |          |         | plain    |
 total_nblocks  | bigint    |           |          |         | plain    |
 free_bytes     | bigint    |           |          |         | plain    |
 free_chunks    | bigint    |           |          |         | plain    |
 used_bytes     | bigint    |           |          |         | plain    |
```

Memory Context

Related developer options

- use –enable-cassert when compiling
  - zeroes out all memory context before deallocating
  - enables sanity checks
  - helps in uncovering subtle bugs
- MemoryContextStats(TopMemoryContext)
  - called in any function or gdb can help figure out leaks

Memory Context

Advantages

- Faster and more reliable than per chunk bookkeeping
- Delete all transaction and shorter length context at the end of the transaction
  - transient memory is available to reuse
- Delete per query context at query completion
  - tuple based context gets deleted too

Memory Context

Summary

- Allocate memory in chunks
- Contexts are arranged in forest like hierarchy
- Each context has a lifetime same or smaller as its parent
- Reduces the risk of memory leaks

Keeping It In The Context!

Thank You!

Questions?

Link for feedback

Rafia Sabih

Sr. PostgreSQL Core Developer

Email

rafia.sabih@cybertec.at

www.cybertec-postgresql.com

@cybertec-postgresql

www.youtube.com/@cybertecpostgresql

PROUD CONTRIBUTOR TO PostgreSQL
the world's most advanced open source database