HailDB

2.3.2

Generated by Doxygen 1.7.1

Fri Dec 24 2010 14:35:34

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Module Documentation

4.1 Startup/Shutdown functions

Functions

- HAILDB_API ib_err_t ib_init (void) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_startup (const char *format) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_shutdown (ib_shutdown_t flag) UNIV_NO_IGNORE

4.1.1 Detailed Description

Define the Doxygen groups:

4.1.2 Function Documentation

4.1.2.1 HAILDB_API ib_err_t ib_init (void)

Initialize the InnoDB engine. This must be called prior to calling any other InnoDB API function. You can call only the ib_cfg_*() functions between calls to ib_init() and ib_startup(). No other HailDB functions should be called.

Returns

DB_SUCCESS or error code

4.1.2.2 HAILDB_API ib_err_t ib_shutdown (ib_shutdown_t *flag*)

Shutdown the InnoDB engine. Call this function when they are no active transactions. It will close all files and release all memory on successful completion. All internal variables will be reset to their default values.

Parameters

flag is the shutdown flag

Returns

DB_SUCCESS or error code

4.1.2.3 HAILDB_API ib_err_t ib_startup (const char * *format*)

Startup the InnoDB engine. If this function is called on a non-existent database then based on the default or user specified configuration settings it will create all the necessary files. If the database was shutdown cleanly but the user deleted the REDO log files then it will recreate the REDO log files.

Parameters

format is the max file format name that the engine supports. Currently this is either Antelope or Barracuda although more may be added in the future without API changes.

Returns

DB_SUCCESS or error code

See also

DB_SUCCESS

4.2 Cursor functions

Functions

- HAILDB_API ib_err_t ib_cursor_open_table_using_id (ib_id_t table_id, ib_trx_t ib_trx, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_open_index_using_id (ib_id_t index_id, ib_trx_t ib_trx, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_open_index_using_name (ib_crsr_t ib_open_crsr, const char *index_name, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_open_table (const char *name, ib_trx_t ib_trx, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_reset (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_close (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_prev (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_next (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_first (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_last (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_moveto (ib_crsr_t ib_crsr, ib_tpl_t ib_tpl, ib_srch_mode_t ib_srch_mode, int *result) UNIV_NO_IGNORE
- HAILDB_API void ib_cursor_attach_trx (ib_crsr_t ib_crsr, ib_trx_t ib_trx)
- HAILDB_API void ib_cursor_set_match_mode (ib_crsr_t ib_crsr, ib_match_mode_t match_mode)
- HAILDB_API ib_bool_t ib_cursor_is_positioned (const ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API void ib_cursor_stmt_begin (ib_crsr_t ib_crsr)

4.2.1 Function Documentation

4.2.1.1 HAILDB_API void ib_cursor_attach_trx (ib_crsr_t *ib_crsr,* ib_trx_t *ib_trx*)

Attach the cursor to the transaction. The cursor must not already be attached to another transaction.

Parameters

ib_crsr is the cursor instance

ib_trx is the transaction to attach to the cursor

4.2.1.2 HAILDB_API ib_err_t ib_cursor_close (ib_crsr_t *ib_crsr*)

Close an InnoDB table and free the cursor.

Parameters

ib_crsr is an open cursor

Returns

DB_SUCCESS or err code

4.2.1.3 HAILDB_API ib_err_t ib_cursor_first (ib_crsr_t *ib_crsr*)

Move cursor to the first record in the table.

Parameters

ib_crsr is the cursor instance

Returns

DB_SUCCESS or err code

4.2.1.4 HAILDB_API ib_bool_t ib_cursor_is_positioned (const ib_crsr_t *ib_crsr*)

Check if cursor is positioned.

Parameters

ib_crsr is the cursor instance to check

Returns

IB_TRUE if positioned

4.2.1.5 HAILDB_API ib_err_t ib_cursor_last (ib_crsr_t *ib_crsr*)

Move cursor to the last record in the table.

Parameters

ib_crsr is the cursor instance

Returns

DB_SUCCESS or err code

4.2.1.6 HAILDB_API ib_err_t ib_cursor_moveto (ib_crsr_t *ib_crsr,* ib_tpl_t *ib_tpl,* ib_srch_mode_t *ib_srch_mode,* int * *result*)

Search for key.

Parameters

ib_crsr is an open cursor instance

ib_tpl is a key to search for

ib_srch_mode is the search mode

[out] ***result*** is -1, 0 or 1 depending on tuple eq or gt than the current row

Returns

DB_SUCCESS or err code

4.2.1.7 HAILDB_API ib_err_t ib_cursor_next (ib_crsr_t *ib_crsr*)

Move cursor to the next user record in the table.

Parameters

ib_crsr is the cursor instance

Returns

DB_SUCCESS or err code

4.2.1.8 HAILDB_API ib_err_t ib_cursor_open_index_using_id (ib_id_t *index_id,* ib_trx_t *ib_trx,* ib_crsr_t * *ib_crsr*)

Open an InnoDB index and return a cursor handle to it.

Parameters

index_id is the id of the index to open

ib_trx is the current transaction handlem can be NULL

[out] ***ib_crsr*** is the new cursor

Returns

DB_SUCCESS or err code

4.2.1.9 HAILDB_API ib_err_t ib_cursor_open_index_using_name (ib_crsr_t *ib_open_crsr,* const char * *index_name,* ib_crsr_t * *ib_crsr*)

Open an InnoDB secondary index cursor and return a cursor handle to it.

Parameters

ib_open_crsr is an open cursor

index_name is the name of the index

`[out]` ***ib_crsr*** is the new cursor

Returns

DB_SUCCESS or err code

4.2.1.10 HAILDB_API ib_err_t ib_cursor_open_table (const char * *name,* ib_trx_t *ib_trx,* ib_crsr_t * *ib_crsr*)

Open an InnoDB table by name and return a cursor handle to it.

Parameters

name is the table name to open

ib_trx is the current transactionm, can be NULL

ib_crsr is the new cursor

Returns

DB_SUCCESS or err code

4.2.1.11 HAILDB_API ib_err_t ib_cursor_open_table_using_id (ib_id_t *table_id,* ib_trx_t *ib_trx,* ib_crsr_t * *ib_crsr*)

Open an InnoDB table and return a cursor handle to it.

Parameters

table_id is the id of the table to open

ib_trx is the current transaction handle, can be NULL

`[out]` ***ib_crsr*** is the new cursor

Returns

DB_SUCCESS or err code

4.2.1.12 HAILDB_API ib_err_t ib_cursor_prev (ib_crsr_t *ib_crsr*)

Move cursor to the prev user record in the table.

Parameters

ib_crsr is the cursor instance

Returns

DB_SUCCESS or err code

4.2.1.13 HAILDB_API ib_err_t ib_cursor_reset (ib_crsr_t *ib_crsr*)

Reset the cursor.

Parameters

ib_crsr is an open cursor

Returns

DB_SUCCESS or err code

4.2.1.14 HAILDB_API void ib_cursor_set_match_mode (ib_crsr_t *ib_crsr,* ib_match_mode_t *match_mode*)

Set the match mode for ib_cursor_move().

Parameters

ib_crsr is the cursor instance

match_mode is the match mode to set

4.2.1.15 HAILDB_API void ib_cursor_stmt_begin (ib_crsr_t *ib_crsr*)

Inform the cursor that it's the start of an SQL statement.

Parameters

ib_crsr is the cursor instance

4.3 Transaction functions

Functions

- HAILDB_API ib_err_t ib_trx_start (ib_trx_t ib_trx, ib_trx_level_t ib_trx_level) UNIV_NO_IGNORE
- HAILDB_API ib_trx_t ib_trx_begin (ib_trx_level_t ib_trx_level) UNIV_NO_IGNORE
- HAILDB_API void ib_trx_set_client_data (ib_trx_t ib_trx, void *client_data)
- HAILDB_API ib_trx_state_t ib_trx_state (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_release (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_commit (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_rollback (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_lock (ib_crsr_t ib_crsr, ib_lck_mode_t ib_lck_mode) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_lock (ib_trx_t ib_trx, ib_id_t table_id, ib_lck_mode_t ib_lck_mode) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_set_lock_mode (ib_crsr_t ib_crsr, ib_lck_mode_t ib_lck_mode) UNIV_NO_IGNORE
- HAILDB_API void ib_savepoint_take (ib_trx_t ib_trx, const void *name, ib_ulint_t name_len)

- HAILDB_API ib_err_t ib_savepoint_release (ib_trx_t ib_trx, const void *name, ib_ulint_t name_len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_savepoint_rollback (ib_trx_t ib_trx, const void *name, ib_ulint_t name_len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_get_duplicate_key (ib_trx_t ib_trx, const char **table_name, const char **index_name)

4.3.1 Function Documentation

4.3.1.1 HAILDB_API ib_err_t ib_cursor_lock (ib_crsr_t *ib_crsr*, ib_lck_mode_t *ib_lck_mode*)

Lock an InnoDB cursor/table.

Parameters

ib_crsr is the cursor instance

ib_lck_mode is the lock mode

Returns

DB_SUCCESS or error code

4.3.1.2 HAILDB_API ib_err_t ib_cursor_set_lock_mode (ib_crsr_t *ib_crsr*, ib_lck_mode_t *ib_lck_mode*)

Set the Lock mode of the cursor.

Parameters

ib_crsr is the cursor instance for which we want to set the lock mode

ib_lck_mode is the lock mode

Returns

DB_SUCCESS or error code

4.3.1.3 HAILDB_API ib_err_t ib_get_duplicate_key (ib_trx_t *ib_trx*, const char ** *table_name*, const char ** *index_name*)

Get which key caused a duplicate key error.

In the event of DB_DUPLICATE_KEY error, you can call this function immediately after to get the name of the table and index that caused the error.

Parameters

ib_trx Transaction where error occured

table_name pointer to be set to table_name. Valid until next ib_ function call. If you would like to keep it, make a copy.

index_name pointer to be set to the index name. Valid until next ib_ function call. If you would like to keep it, make a copy.

4.3.1.4 HAILDB_API ib_err_t ib_savepoint_release (ib_trx_t *ib_trx*, const void * *name*, ib_ulint_t *name_len*)

Releases only the named savepoint. Savepoints which were set after this savepoint are left as is.

Parameters

ib_trx is the active transaction

name is the name of the savepoint

name_len is the length of name in bytes

Returns

if no savepoint of the name found then DB_NO_SAVEPOINT, otherwise DB_SUCCESS

4.3.1.5 HAILDB_API ib_err_t ib_savepoint_rollback (ib_trx_t *ib_trx*, const void * *name*, ib_ulint_t *name_len*)

Rolls back a transaction back to a named savepoint. Modifications after the savepoint are undone but InnoDB does NOT release the corresponding locks which are stored in memory. If a lock is 'implicit', that is, a new inserted row holds a lock where the lock information is carried by the trx id stored in the row, these locks are naturally released in the rollback. Savepoints which were set after this savepoint are deleted. If name equals NULL then all the savepoints are rolled back.

Parameters

ib_trx is the active transaction

name is the savepoint name can be NULL

name_len is the length of name in bytes

Returns

if no savepoint of the name found then DB_NO_SAVEPOINT, otherwise DB_SUCCESS

4.3.1.6 HAILDB_API void ib_savepoint_take (ib_trx_t *ib_trx*, const void * *name*, ib_ulint_t *name_len*)

Creates a named savepoint. The transaction must be started. If there is already a savepoint of the same name, this call erases that old savepoint and replaces it with a new. Savepoints are deleted in a transaction commit or rollback.

Parameters

ib_trx is the transaction instance

name is the name of the savepoint

name_len is the length of name in bytes

4.3.1.7 HAILDB_API ib_err_t ib_table_lock (ib_trx_t *ib_trx,* ib_id_t *table_id,* ib_lck_mode_t *ib_lck_mode*)

Set the Lock an InnoDB table using the table id.

Parameters

ib_trx is a transaction instance

table_id is the table to lock

ib_lck_mode is the lock mode

Returns

DB_SUCCESS or error code

4.3.1.8 HAILDB_API ib_trx_t ib_trx_begin (ib_trx_level_t *ib_trx_level*)

Begin a transaction. This will allocate a new transaction handle and put the transaction in the active state.

Parameters

ib_trx_level is the transaction isolation level

Returns

innobase txn handle

4.3.1.9 HAILDB_API ib_err_t ib_trx_commit (ib_trx_t *ib_trx*)

Commit a transaction. This function will release the schema latches too. It will also free the transaction handle.

Parameters

ib_trx is thr transaction handle

Returns

DB_SUCCESS or err code

4.3.1.10 HAILDB_API ib_err_t ib_trx_release (ib_trx_t *ib_trx*)

Release the resources of the transaction. If the transaction was selected as a victim by InnoDB and rolled back then use this function to free the transaction handle.

Parameters

ib_trx is the transaction handle

Returns

DB_SUCCESS or err code

4.3.1.11 HAILDB_API ib_err_t ib_trx_rollback (ib_trx_t *ib_trx*)

Rollback a transaction. This function will release the schema latches too. It will also free the transaction handle.

Parameters

ib_trx is the transaction handle

Returns

DB_SUCCESS or err code

4.3.1.12 HAILDB_API void ib_trx_set_client_data (ib_trx_t *ib_trx*, void * *client_data*)

Set client data for a transaction. This is passed back to the client in the trx_is_interrupted callback. HailDB will only ever pass this around, it will never dereference it.

Parameters

ib_trx is the transaction to set the client data for

client_data is client program's data about this transaction

Parameters

ib_trx in: transaction

4.3.1.13 HAILDB_API ib_err_t ib_trx_start (ib_trx_t *ib_trx*, ib_trx_level_t *ib_trx_level*)

Start a transaction that's been rolled back. This special function exists for the case when InnoDB's deadlock detector has rolledack a transaction. While the transaction has been rolled back the handle is still valid and can be reused by calling this function. If you don't want to reuse the transaction handle then you can free the handle by calling ib_trx_release().

Parameters

ib_trx is the transaction to restart

ib_trx_level is the transaction isolation level

Returns

innobase txn handle

4.3.1.14 HAILDB_API ib_trx_state_t ib_trx_state (ib_trx_t *ib_trx*)

Query the transaction's state. This function can be used to check for the state of the transaction in case it has been rolled back by the InnoDB deadlock detector. Note that when a transaction is selected as a victim for rollback, InnoDB will always return an appropriate error code indicating this.

See also

DB_DEADLOCK,
DB_LOCK_TABLE_FULL and
DB_LOCK_WAIT_TIMEOUT

Parameters

ib_trx is the transaction handle

Returns

transaction state

4.4 Configuration functions

Defines

- #define ib_cfg_set_int(name, value) ib_cfg_set(name, value)
- #define ib_cfg_set_text(name, value) ib_cfg_set(name, value)
- #define ib_cfg_set_bool_on(name) ib_cfg_set(name, IB_TRUE)
- #define ib_cfg_set_bool_off(name) ib_cfg_set(name, IB_FALSE)
- #define ib_cfg_set_callback(name, value) ib_cfg_set(name, value)

Functions

- HAILDB_API ib_err_t ib_cfg_var_get_type (const char *name, ib_cfg_type_t *type) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_set (const char *name,...) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_get (const char *name, void *value) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_get_all (const char ***names, ib_u32_t *names_num) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_status_get_all (const char ***names, ib_u32_t *names_num) UNIV_NO_IGNORE
- HAILDB_API void ib_set_panic_handler (ib_panic_handler_t new_panic_handler)
- HAILDB_API void ib_set_trx_is_interrupted_handler (ib_trx_is_interrupted_handler_t handler)

4.4.1 Define Documentation

4.4.1.1 #define ib_cfg_set_bool_off(*name*) ib_cfg_set(name, IB_FALSE)

Set a boolean configuration variable to IB_FALSE.

Parameters

name is the config variable name

Returns

DB_SUCCESS or error code

4.4.1.2 #define ib_cfg_set_bool_on(*name*) ib_cfg_set(name, IB_TRUE)

Set a boolean configuration variable to IB_TRUE.

Parameters

name is the config variable name

Returns

DB_SUCCESS or error code

4.4.1.3 #define ib_cfg_set_callback(*name, value*) ib_cfg_set(name, value)

Set a generic ib_cb_t callback function.

Parameters

name is the config variable name

value is a pointer to a callback function

Returns

DB_SUCCESS or error code

4.4.1.4 #define ib_cfg_set_int(*name, value*) ib_cfg_set(name, value)

Set an int configuration variable.

Parameters

name is the config variable name

value is the integer value of the variable

Returns

DB_SUCCESS or error code

4.4.1.5 #define ib_cfg_set_text(*name, value*) ib_cfg_set(name, value)

Set a text configuration variable.

Parameters

name is the config variable name

value is the char∗ value of the variable

Returns

DB_SUCCESS or error code

4.4.2 Function Documentation

4.4.2.1 HAILDB_API ib_err_t ib_cfg_get (const char * *name*, void * *value*)

Get the value of a configuration variable. The type of the returned value depends on the type of the configuration variable. DB_SUCCESS is returned if the variable with name "name" was found and "value" was set.

Parameters

name is the variable name whose value needs to be accessed

`[out]` ***value*** is the value of the variable if found

Returns

DB_SUCCESS if retrieved successfully

4.4.2.2 HAILDB_API ib_err_t ib_cfg_get_all (const char *** *names*, ib_u32_t * *names_num*)

Get a list of the names of all configuration variables. The caller is responsible for free(3)ing the returned array of strings when it is not needed anymore and for not modifying the individual strings.

Parameters

`[out]` ***names*** pointer to array of strings

`[out]` ***names_num*** number of strings returned

Returns

DB_SUCCESS or error code

4.4.2.3 HAILDB_API ib_err_t ib_cfg_set (const char * *name*, ...)

Set a configuration variable. The second argument's type depends on the type of the variable with the given "name". Returns DB_SUCCESS if the variable with name "name" was found and if its value was set.

Parameters

name is the config variable name whose value is to be set

Returns

DB_SUCCESS if set

4.4.2.4 HAILDB_API ib_err_t ib_cfg_var_get_type (const char * *name*, ib_cfg_type_t * *type*)

Get the type of a configuration variable. Returns DB_SUCCESS if the variable with name "name" was found and "type" was set.

Parameters

name is the variable name to look up

[out] ***type*** is the type of the variable name if found

Returns

DB_SUCCESS if successful

4.4.2.5 HAILDB_API void ib_set_panic_handler (ib_panic_handler_t *new_panic_handler*)

Set panic handler.

HailDB will "panic" upon finding certain forms of corruption. By setting a panic handler, you can implement your own notification to the end user of this corruption (e.g. popping up a dialog box).

Parameters

new_panic_handler your panic handler

4.4.2.6 HAILDB_API void ib_set_trx_is_interrupted_handler (ib_trx_is_interrupted_handler_t *handler*)

Set trx_is_interrupted_handler.

You may specificy a callback that HailDB will check during certain wait situations to see if it should abort the operation or not. This lets you implement MySQL/Drizzle KILL command style functionality.

Parameters

handler the trx_is_interrupted callback

4.4.2.7 HAILDB_API ib_err_t ib_status_get_all (const char *** *names*, ib_u32_t * *names_num*)

Get a list of the names of all status variables. The caller is responsible for free(3)ing the returned array of strings when it is not needed anymore and for not modifying the individual strings.

Parameters

[out] ***names*** pointer to array of strings

[out] ***names_num*** number of strings returned

Returns

DB_SUCCESS or error code

4.5 DDL functions

Functions

- HAILDB_API ib_err_t ib_table_schema_add_col (ib_tbl_sch_t ib_tbl_sch, const char *name, ib_col_type_t ib_col_type, ib_col_attr_t ib_col_attr, ib_u16_t client_type, ib_ulint_t len) UNIV_NO_IGNORE

- HAILDB_API ib_err_t ib_table_schema_add_index (ib_tbl_sch_t ib_tbl_sch, const char *name, ib_idx_sch_t *ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API void ib_table_schema_delete (ib_tbl_sch_t ib_tbl_sch)
- HAILDB_API ib_err_t ib_table_schema_create (const char *name, ib_tbl_sch_t *ib_tbl_sch, ib_tbl_fmt_t ib_tbl_fmt, ib_ulint_t page_size) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_add_col (ib_idx_sch_t ib_idx_sch, const char *name, ib_ulint_t prefix_len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_create (ib_trx_t ib_usr_trx, const char *name, const char *table_name, ib_idx_sch_t *ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_set_clustered (ib_idx_sch_t ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_set_unique (ib_idx_sch_t ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API void ib_index_schema_delete (ib_idx_sch_t ib_idx_sch)
- HAILDB_API ib_err_t ib_table_create (ib_trx_t ib_trx, const ib_tbl_sch_t ib_tbl_sch, ib_id_t *id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_rename (ib_trx_t ib_trx, const char *old_name, const char *new_name) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_create (ib_idx_sch_t ib_idx_sch, ib_id_t *index_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_drop (ib_trx_t trx, const char *name) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_drop (ib_trx_t trx, ib_id_t index_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_truncate (ib_crsr_t *ib_crsr, ib_id_t *table_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_truncate (const char *table_name, ib_id_t *table_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_get_id (const char *table_name, ib_id_t *table_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_get_id (const char *table_name, const char *index_name, ib_id_t *index_id) UNIV_NO_IGNORE
- HAILDB_API ib_bool_t ib_database_create (const char *dbname) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_database_drop (const char *dbname) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_lock_shared (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_lock_exclusive (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_bool_t ib_schema_lock_is_exclusive (const ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_unlock (ib_trx_t ib_trx)
- HAILDB_API ib_err_t ib_table_schema_visit (ib_trx_t ib_trx, const char *name, const ib_schema_visitor_t *visitor, void *arg) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_tables_iterate (ib_trx_t ib_trx, ib_schema_visitor_table_all_t visitor, void *arg) UNIV_NO_IGNORE

4.5.1 Function Documentation

4.5.1.1 HAILDB_API ib_err_t ib_cursor_truncate (ib_crsr_t * *ib_crsr*, ib_id_t * *table_id*)

Truncate a table. The cursor handle will be closed and set to NULL on success.

Parameters

`[out]` ***ib_crsr*** is the cursor for table to truncate

`[out]` ***table_id*** is the new table id

Returns

DB_SUCCESS or error code

4.5.1.2 HAILDB_API ib_bool_t ib_database_create (const char * *dbname*)

Create a database if it doesn't exist.

Parameters

dbname is the name of the database to create

Returns

IB_TRUE on success

4.5.1.3 HAILDB_API ib_err_t ib_database_drop (const char * *dbname*)

Drop a database if it exists. This function will also drop all tables within the database.

Parameters

dbname is the name of the database to drop

Returns

DB_SUCCESS or error code

4.5.1.4 HAILDB_API ib_err_t ib_index_create (ib_idx_sch_t *ib_idx_sch*, ib_id_t * *index_id*)

Create a secondary index. The index id encodes the table id in the high 4 bytes and the index id in the lower 4 bytes.

Parameters

`[in,out]` ***ib_idx_sch*** the schema for the index

`[out]` ***index_id*** is the new index id that was created

Returns

DB_SUCCESS or err code

4.5.1.5 HAILDB_API ib_err_t ib_index_drop (ib_trx_t *trx*, ib_id_t *index_id*)

Drop a secondary index. Ensure that you have acquired the schema lock in exclusive mode.

Parameters

trx is the covering transaction.

index_id is the id of the index to drop

Returns

DB_SUCCESS or err code

4.5.1.6 HAILDB_API ib_err_t ib_index_get_id (const char * *table_name*, const char * *index_name*, ib_id_t * *index_id*)

Get an index id.

Parameters

table_name is the name of the table that contains the index

index_name is the name of the index to lookup

`[out]` ***index_id*** contains the index id if found

Returns

DB_SUCCESS if found

4.5.1.7 HAILDB_API ib_err_t ib_index_schema_add_col (ib_idx_sch_t *ib_idx_sch*, const char * *name*, ib_ulint_t *prefix_len*)

Add columns to an index schema definition.

Parameters

`[in,out]` ***ib_idx_sch*** is the index schema instance

name is the name of the column to add to the index schema

prefix_len is the prefix length of the index or 0 if no prefix

Returns

DB_SUCCESS or err code

4.5.1.8 HAILDB_API ib_err_t ib_index_schema_create (ib_trx_t *ib_usr_trx*, const char * *name*, const char * *table_name*, ib_idx_sch_t * *ib_idx_sch*)

Create an index schema instance.

Parameters

ib_usr_trx is the current user transaction

name is the name of the index to create

table_name is the name of the table the index belongs to

`[out]` ***ib_idx_sch*** is the newly created index schema instance

Returns

DB_SUCCESS or err code

4.5.1.9 HAILDB_API void ib_index_schema_delete (ib_idx_sch_t *ib_idx_sch*)

Destroy an index schema.

Parameters

ib_idx_sch is the index schema to delete

4.5.1.10 HAILDB_API ib_err_t ib_index_schema_set_clustered (ib_idx_sch_t *ib_idx_sch*)

Set index as clustered index. Implies UNIQUE.

Parameters

`[in,out]` ***ib_idx_sch*** is the index schema to update

Returns

DB_SUCCESS or err code

4.5.1.11 HAILDB_API ib_err_t ib_index_schema_set_unique (ib_idx_sch_t *ib_idx_sch*)

Set index as a unique index.

Parameters

`[in,out]` ***ib_idx_sch*** is the index schema to update

Returns

DB_SUCCESS or err code

4.5.1.12 HAILDB_API ib_err_t ib_schema_lock_exclusive (ib_trx_t *ib_trx*)

Latches the data dictionary in exclusive mode.

Parameters

ib_trx is the transaction instance

Returns

DB_SUCCESS or error code

4.5.1.13 HAILDB_API ib_bool_t ib_schema_lock_is_exclusive (const ib_trx_t *ib_trx*)

Checks if the data dictionary is latched in exclusive mode by a user transaction.

Parameters

ib_trx is a transaction instance

Returns

TRUE if exclusive latch

4.5.1.14 HAILDB_API ib_err_t ib_schema_lock_shared (ib_trx_t *ib_trx*)

Latches the data dictionary in shared mode.

Parameters

ib_trx is the transaction instance

Returns

DB_SUCCESS or error code

4.5.1.15 HAILDB_API ib_err_t ib_schema_tables_iterate (ib_trx_t *ib_trx*, ib_schema_visitor_table_all_t *visitor*, void * *arg*)

List all the tables in the InnoDB's data dictionary. It will abort if visitor returns a non-zero value.

It will call the function: visitor.tables(arg, const char* name, int name_len);

The function will abort if visitor.tables() returns non-zero.

Parameters

ib_trx is the transaction that owns the schema lock

visitor is the visitor function

arg argument passed to the visitor function

Returns

DB_SUCCESS if successful

4.5.1.16 HAILDB_API ib_err_t ib_schema_unlock (ib_trx_t *ib_trx*)

Unlocks the data dictionary.

Parameters

ib_trx is a transaction instance

Returns

DB_SUCCESS or error code

4.5.1.17 HAILDB_API ib_err_t ib_table_create (ib_trx_t *ib_trx*, const ib_tbl_sch_t *ib_tbl_sch*, ib_id_t * *id*)

Create a table in the InnoDB data dictionary using the schema definition. If the table exists in the database then this function will return DB_TABLE_IS_BEING_USED and id will contain that table's id.

Parameters

`[in,out]` ***ib_trx*** the current user transaction

ib_tbl_sch the the schema for the table to create

[out] ***id*** table id that was created

Returns

DB_SUCCESS or err code

4.5.1.18 HAILDB_API ib_err_t ib_table_drop (ib_trx_t *trx*, const char * *name*)

Drop a table. Ensure that you have acquired the schema lock in exclusive mode.

Parameters

trx is the covering transaction.

name is the name of the table to drop

Returns

DB_SUCCESS or err code

4.5.1.19 HAILDB_API ib_err_t ib_table_get_id (const char * *table_name*, ib_id_t * *table_id*)

Get a table id.

Parameters

table_name is the name of the table to lookup

[out] ***table_id*** is the new table id if found

Returns

DB_SUCCESS if found

4.5.1.20 HAILDB_API ib_err_t ib_table_rename (ib_trx_t *ib_trx*, const char * *old_name*, const char * *new_name*)

Rename a table. Ensure that you have acquired the schema lock in exclusive mode.

Parameters

[in,out] ***ib_trx*** is the current user transaction

old_name the current name of the table

new_name the new name for the table

Returns

DB_SUCCESS or err code

4.5.1.21 HAILDB_API ib_err_t ib_table_schema_add_col (ib_tbl_sch_t *ib_tbl_sch,* const char * *name,* ib_col_type_t *ib_col_type,* ib_col_attr_t *ib_col_attr,* ib_u16_t *client_type,* ib_ulint_t *len*)

Add columns to a table schema. Tables are created in InnoDB by first creating a table schema which is identified by a handle. Then you add the column definitions to the table schema.

Parameters

ib_tbl_sch is the table schema instance

name is the name of the column to add

ib_col_type is the type of the column

ib_col_attr are the attributes of the column, including constraints

client_type is any 16 bit number relevant only to the client

len is the maximum length of the column

Returns

DB_SUCCESS or err code

4.5.1.22 HAILDB_API ib_err_t ib_table_schema_add_index (ib_tbl_sch_t *ib_tbl_sch,* const char * *name,* ib_idx_sch_t * *ib_idx_sch*)

Create and add an index key definition to a table schema. The index schema is owned by the table schema instance and will be freed when the table schema instance is freed.

Parameters

`[in,out]` ***ib_tbl_sch*** is the schema instance

name name of the key definition to create

`[out]` ***ib_idx_sch*** is the key definition schema instance

Returns

DB_SUCCESS or err code

4.5.1.23 HAILDB_API ib_err_t ib_table_schema_create (const char * *name,* ib_tbl_sch_t * *ib_tbl_sch,* ib_tbl_fmt_t *ib_tbl_fmt,* ib_ulint_t *page_size*)

Create a table schema.

Parameters

name is the table name for which to create the schema

`[out]` ***ib_tbl_sch*** is the schema instance that is created

ib_tbl_fmt is the format of the table to be created

page_size is the page size for the table or 0 for default

Returns

DB_SUCCESS or err code

4.5.1.24 HAILDB_API void ib_table_schema_delete (ib_tbl_sch_t *ib_tbl_sch*)

Destroy a schema. The handle is freed by this function.

Parameters

ib_tbl_sch is the table schema to delte

4.5.1.25 HAILDB_API ib_err_t ib_table_schema_visit (ib_trx_t *ib_trx*, const char * *name*, const ib_schema_visitor_t * *visitor*, void * *arg*)

Read a table's schema using the visitor pattern. It will make the following sequence of calls:

visitor->table() visitor->table_col() for each user column visitor->index() for each user index visitor->index_col() for each column in user index

It will stop if any of the above functions returns a non-zero value. The caller must have an exclusive lock on the InnoDB data dictionary

Parameters

ib_trx transaction that owns the schema lock

name is the table name to read

visitor visitor functions to invoke on each definition

arg is the argument passed to the visitor functions.

Returns

DB_SUCCESS or DB_ERROR

4.5.1.26 HAILDB_API ib_err_t ib_table_truncate (const char * *table_name*, ib_id_t * *table_id*)

Truncate a table.

Parameters

table_name is the name of the table to truncate

`[out]` ***table_id*** is the new table id

Returns

DB_SUCCESS or error code

4.6 Miscellaneous functions

Functions

- HAILDB_API ib_u64_t ib_api_version (void) UNIV_NO_IGNORE
- HAILDB_API void ib_set_client_compare (ib_client_cmp_t client_cmp_func)
- HAILDB_API void ib_logger_set (ib_msg_log_t ib_msg_log, ib_msg_stream_t ib_msg_stream)
- HAILDB_API const char * ib_strerror (ib_err_t db_errno) UNIV_NO_IGNORE

- HAILDB_API ib_err_t ib_status_get_i64 (const char *name, ib_i64_t *dst) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_get_table_statistics (ib_crsr_t ib_crsr, ib_table_stats_t *table_stats, size_t sizeof_ib_table_stats_t)
- HAILDB_API ib_err_t ib_get_index_stat_n_diff_key_vals (ib_crsr_t ib_crsr, const char *index_name, ib_u64_t *ncols, ib_i64_t **n_diff)
- HAILDB_API ib_err_t ib_update_table_statistics (ib_crsr_t crsr)

4.6.1 Function Documentation

4.6.1.1 HAILDB_API ib_u64_t ib_api_version (void)

Return the API version number, the version number format is: | 16 bits future use | 16 bits current | 16 bits revision | 16 bits age |

- If the library source code has changed at all since the last release, then revision will be incremented ('c:r:a' becomes 'c:r+1:a').
- If any interfaces have been added, removed, or changed since the last update, current will be incremented, and revision will be set to 0.
- If any interfaces have been added (but not changed or removed) since the last release, then age will be incremented.
- If any interfaces have been changed or removed since the last release, then age will be set to 0.

Returns

API version number

4.6.1.2 HAILDB_API ib_err_t ib_get_index_stat_n_diff_key_vals (ib_crsr_t *ib_crsr*, const char * *index_name*, ib_u64_t * *ncols*, ib_i64_t ** *n_diff*)

Get statistics on number of different key values per index part

This function returns the approximate different key values for this index. They are periodically recalculated.

Parameters

ib_crsr A Cursor that is opened to a table

index_name name of the index

ncols returns the number of elements in n_diff

n_diff An array allocated with malloc() (user needs to free()) containing the statistics

Returns

DB_SUCCESS or error. DB_NOT_FOUND if index is not found

4.6.1.3 HAILDB_API ib_err_t ib_get_table_statistics (ib_crsr_t *ib_crsr*, ib_table_stats_t * *table_stats*, size_t *sizeof_ib_table_stats_t*)

Get table statistics.

This function will fill out the provided ib_table_stats_t with statistics about the table on the currently opened ib_crsr_t

Parameters

ib_crsr A Cursor that is opened to a table

table_stats a ib_table_stats_t to be filled out by HailDB

sizeof_ib_table_stats_t sizeof(ib_table_stats_t). This allows for ABI compatible changes to the size of ib_table_stats_t.

Returns

DB_SUCCESS or error

4.6.1.4 HAILDB_API void ib_logger_set (ib_msg_log_t *ib_msg_log*, ib_msg_stream_t *ib_msg_stream*)

Set the message logging function.

Parameters

ib_msg_log is the message logging function

ib_msg_stream is the message stream, this is the first argument to the loggingfunction

4.6.1.5 HAILDB_API void ib_set_client_compare (ib_client_cmp_t *client_cmp_func*)

Set the client comparison function for BLOBs and client types.

Parameters

client_cmp_func is the index key compare callback function

4.6.1.6 HAILDB_API ib_err_t ib_status_get_i64 (const char * *name*, ib_i64_t * *dst*)

Get the value of an INT status variable.

Parameters

name is the status variable name

`[out]` ***dst*** is where the output value is copied if name is found

Returns

DB_SUCCESS if found and type is INT, DB_DATA_MISMATCH if found but type is not INT, DB_NOT_FOUND otherwise.

4.6.1.7 HAILDB_API const char* ib_strerror (ib_err_t *db_errno*)

Convert an error number to a human readable text message. The returned string is static and should not be freed or modified.

Parameters

db_errno is the error number

Returns

string, describing the error

4.6.1.8 HAILDB_API ib_err_t ib_update_table_statistics (ib_crsr_t *crsr*)

Force an update of table and index statistics

This function forces an update to the table and index statistics for the table crsr is opened on.

Parameters

crsr A Cursor that is opened to a table

Returns

DB_SUCCESS or error.

4.7 Tuple functions

Functions

- HAILDB_API ib_tpl_t ib_tuple_clear (ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_get_cluster_key (ib_crsr_t ib_crsr, ib_tpl_t *ib_dst_tpl, const ib_tpl_t ib_src_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_copy (ib_tpl_t ib_dst_tpl, const ib_tpl_t ib_src_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_sec_search_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_sec_read_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_clust_search_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_clust_read_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_tuple_get_n_user_cols (const ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_tuple_get_n_cols (const ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API void ib_tuple_delete (ib_tpl_t ib_tpl)

4.7.1 Function Documentation

4.7.1.1 HAILDB_API ib_tpl_t ib_clust_read_tuple_create (ib_crsr_t *ib_crsr*)

Create an InnoDB tuple for table row operations.

Parameters

ib_crsr is the cursor instance

Returns

tTuple for current table

4.7.1.2 HAILDB_API ib_tpl_t ib_clust_search_tuple_create (ib_crsr_t *ib_crsr*)

Create an InnoDB tuple used for table key operations.

Parameters

ib_crsr is the cursor instance

Returns

tuple for current table

4.7.1.3 HAILDB_API ib_tpl_t ib_sec_read_tuple_create (ib_crsr_t *ib_crsr*)

Create an InnoDB tuple used for index/table search.

Parameters

ib_crsr is the cursor instance

Returns

tuple for current index

4.7.1.4 HAILDB_API ib_tpl_t ib_sec_search_tuple_create (ib_crsr_t *ib_crsr*)

Create an InnoDB tuple used for index/table search.

Parameters

ib_crsr is the cursor instance

Returns

tuple for current index

4.7.1.5 HAILDB_API ib_tpl_t ib_tuple_clear (ib_tpl_t *ib_tpl*)

"Clear" or reset an InnoDB tuple. We free the heap and recreate the tuple.

Parameters

ib_tpl is the tuple to be freed

Returns

new tuple, or NULL

4.7.1.6 HAILDB_API ib_err_t ib_tuple_copy (ib_tpl_t *ib_dst_tpl*, const ib_tpl_t *ib_src_tpl*)

Copy the contents of source tuple to destination tuple. The tuples must be of the same type and belong to the same table/index.

Parameters

ib_dst_tpl is the destination tuple

ib_src_tpl is the source tuple

Returns

DB_SUCCESS or error code

4.7.1.7 HAILDB_API void ib_tuple_delete (ib_tpl_t *ib_tpl*)

Destroy an InnoDB tuple.

Parameters

ib_tpl is the tuple instance to delete

4.7.1.8 HAILDB_API ib_err_t ib_tuple_get_cluster_key (ib_crsr_t *ib_crsr*, ib_tpl_t * *ib_dst_tpl*, const ib_tpl_t *ib_src_tpl*)

Create a new cluster key search tuple and copy the contents of the secondary index key tuple columns that refer to the cluster index record to the cluster key. It does a deep copy of the column data.

Parameters

ib_crsr is a cursor opened on a secondary index

`[out]` ***ib_dst_tpl*** is the tuple where the key data will be copied

ib_src_tpl is the source secondary index tuple to copy from

Returns

DB_SUCCESS or error code

4.7.1.9 HAILDB_API ib_ulint_t ib_tuple_get_n_cols (const ib_tpl_t *ib_tpl*)

Return the number of columns in the tuple definition.

Parameters

ib_tpl is a tuple

Returns

number of columns

4.7.1.10 HAILDB_API ib_ulint_t ib_tuple_get_n_user_cols (const ib_tpl_t *ib_tpl*)

Return the number of user columns in the tuple definition.

Parameters

ib_tpl is a tuple

Returns

number of user columns

4.8 SQL functions

Functions

- HAILDB_API void ib_cursor_set_simple_select (ib_crsr_t ib_crsr)

4.8.1 Function Documentation

4.8.1.1 HAILDB_API void ib_cursor_set_simple_select (ib_crsr_t *ib_crsr*)

Set to true if it's a simple select.

Parameters

`[in,out]` ***ib_crsr*** is the cursor to update

4.9 DML functions

Functions

- HAILDB_API ib_err_t ib_cursor_insert_row (ib_crsr_t ib_crsr, const ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_update_row (ib_crsr_t ib_crsr, const ib_tpl_t ib_old_tpl, const ib_tpl_t ib_new_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_delete_row (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_read_row (ib_crsr_t ib_crsr, ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_col_set_value (ib_tpl_t ib_tpl, ib_ulint_t col_no, const void *src, ib_ulint_t len) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_col_get_len (ib_tpl_t ib_tpl, ib_ulint_t i) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_col_copy_value (ib_tpl_t ib_tpl, ib_ulint_t i, void *dst, ib_ulint_t len)
- HAILDB_API ib_err_t ib_tuple_read_i8 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i8_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u8 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u8_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_i16 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i16_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u16 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u16_t *ival) UNIV_NO_IGNORE

- HAILDB_API ib_err_t ib_tuple_read_i32 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i32_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u32 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u32_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_i64 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i64_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u64 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u64_t *ival) UNIV_NO_IGNORE
- HAILDB_API const void * ib_col_get_value (ib_tpl_t ib_tpl, ib_ulint_t i) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_col_get_meta (ib_tpl_t ib_tpl, ib_ulint_t i, ib_col_meta_t *ib_col_meta)
- HAILDB_API void ib_cursor_set_cluster_access (ib_crsr_t ib_crsr)
- HAILDB_API ib_err_t ib_tuple_write_i8 (ib_tpl_t ib_tpl, int col_no, ib_i8_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i16 (ib_tpl_t ib_tpl, int col_no, ib_i16_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i32 (ib_tpl_t ib_tpl, int col_no, ib_i32_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i64 (ib_tpl_t ib_tpl, int col_no, ib_i64_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u8 (ib_tpl_t ib_tpl, int col_no, ib_u8_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u16 (ib_tpl_t ib_tpl, int col_no, ib_u16_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u32 (ib_tpl_t ib_tpl, int col_no, ib_u32_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u64 (ib_tpl_t ib_tpl, int col_no, ib_u64_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_double (ib_tpl_t ib_tpl, int col_no, double val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_double (ib_tpl_t ib_tpl, ib_ulint_t col_no, double *dval) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_float (ib_tpl_t ib_tpl, int col_no, float val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_float (ib_tpl_t ib_tpl, ib_ulint_t col_no, float *fval) UNIV_NO_IGNORE

4.9.1 Function Documentation

4.9.1.1 HAILDB_API ib_ulint_t ib_col_copy_value (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, void * *dst*, ib_ulint_t *len*)

Copy a column value from the tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***dst*** is where the data will be copied

len is the maximum number of bytes that can be copied to dst

Returns

bytes copied or IB_SQL_NULL

4.9.1.2 HAILDB_API ib_ulint_t ib_col_get_len (ib_tpl_t *ib_tpl,* ib_ulint_t *i*)

Get the size of the data available in the column the tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

Returns

bytes avail or IB_SQL_NULL

4.9.1.3 HAILDB_API ib_ulint_t ib_col_get_meta (ib_tpl_t *ib_tpl,* ib_ulint_t *i,* ib_col_meta_t * *ib_col_meta*)

Get a column type, length and attributes from the tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ib_col_meta*** the column meta data

Returns

len of column data

4.9.1.4 HAILDB_API const void* ib_col_get_value (ib_tpl_t *ib_tpl,* ib_ulint_t *i*)

Get a column value pointer from the tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

Returns

NULL or pointer to buffer

4.9.1.5 HAILDB_API ib_err_t ib_col_set_value (ib_tpl_t *ib_tpl,* ib_ulint_t *col_no,* const void * *src,* ib_ulint_t *len*)

Set a column of the tuple. Make a copy using the tuple's heap.

Parameters

ib_tpl is the tuple instance

col_no is the column index in the tuple

src is the data value to set

len is the data value (src) length in bytes

Returns

DB_SUCCESS or error code

4.9.1.6 HAILDB_API ib_err_t ib_cursor_delete_row (ib_crsr_t *ib_crsr*)

Delete a row in a table.

Parameters

ib_crsr is the cursor instance

Returns

DB_SUCCESS or err code

4.9.1.7 HAILDB_API ib_err_t ib_cursor_insert_row (ib_crsr_t *ib_crsr,* const ib_tpl_t *ib_tpl*)

Insert a row to a table.

Parameters

ib_crsr is an open cursor

ib_tpl is the tuple to insert

Returns

DB_SUCCESS or err code

4.9.1.8 HAILDB_API ib_err_t ib_cursor_read_row (ib_crsr_t *ib_crsr,* ib_tpl_t *ib_tpl*)

Read current row.

Parameters

ib_crsr is the cursor instance

`[out]` ***ib_tpl*** is the tuple to read the column values

Returns

DB_SUCCESS or err code

4.9.1.9 HAILDB_API void ib_cursor_set_cluster_access (ib_crsr_t *ib_crsr*)

Set need to access clustered index record flag.

Parameters

ib_crsr is the cursor instance for which we want to set the flag

4.9.1.10 HAILDB_API ib_err_t ib_cursor_update_row (ib_crsr_t *ib_crsr*, const ib_tpl_t *ib_old_tpl*, const ib_tpl_t *ib_new_tpl*)

Update a row in a table.

Parameters

ib_crsr is the cursor instance

ib_old_tpl is the old tuple in the table

ib_new_tpl is the new tuple with the updated values

Returns

DB_SUCCESS or err code

4.9.1.11 HAILDB_API ib_err_t ib_tuple_read_double (ib_tpl_t *ib_tpl*, ib_ulint_t *col_no*, double * *dval*)

Read a double column value from an InnoDB tuple.

Parameters

ib_tpl is the tuple to read from

col_no is the column number to read

[out] ***dval*** is where the value is copied

Returns

DB_SUCCESS or error

4.9.1.12 HAILDB_API ib_err_t ib_tuple_read_float (ib_tpl_t *ib_tpl*, ib_ulint_t *col_no*, float * *fval*)

Read a float value from an InnoDB tuple.

Parameters

ib_tpl is the tuple to read from

col_no is the column number to read

[out] ***fval*** is where the value is copied

Returns

DB_SUCCESS or error

4.9.1.13 HAILDB_API ib_err_t ib_tuple_read_i16 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_i16_t * *ival*)

Read a signed int 16 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.14 HAILDB_API ib_err_t ib_tuple_read_i32 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_i32_t * *ival*)

Read a signed int 32 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.15 HAILDB_API ib_err_t ib_tuple_read_i64 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_i64_t * *ival*)

Read a signed int 64 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.16 HAILDB_API ib_err_t ib_tuple_read_i8 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_i8_t * *ival*)

Read a signed int 8 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.17 HAILDB_API ib_err_t ib_tuple_read_u16 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_u16_t * *ival*)

Read an unsigned int 16 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.18 HAILDB_API ib_err_t ib_tuple_read_u32 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_u32_t * *ival*)

Read an unsigned int 32 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.19 HAILDB_API ib_err_t ib_tuple_read_u64 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_u64_t * *ival*)

Read an unsigned int 64 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.20 HAILDB_API ib_err_t ib_tuple_read_u8 (ib_tpl_t *ib_tpl*, ib_ulint_t *i*, ib_u8_t * *ival*)

Read an unsigned int 8 bit column from an InnoDB tuple.

Parameters

ib_tpl is the tuple instance

i is the index (ordinal position) of the column within the tuple

`[out]` ***ival*** is the integer value

Returns

DB_SUCCESS or error

4.9.1.21 HAILDB_API ib_err_t ib_tuple_write_double (ib_tpl_t *ib_tpl*, int *col_no*, double *val*)

Write a double value to a column.

Parameters

`[in,out]` ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCCESS or error

4.9.1.22 HAILDB_API ib_err_t ib_tuple_write_float (ib_tpl_t *ib_tpl*, int *col_no*, float *val*)

Write a float value to a column.

Parameters

`[in,out]` ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCCESS or error

4.9.1.23 HAILDB_API ib_err_t ib_tuple_write_i16 (ib_tpl_t *ib_tpl*, int *col_no*, ib_i16_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

`[in,out]` ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.24 HAILDB_API ib_err_t ib_tuple_write_i32 (ib_tpl_t *ib_tpl*, int *col_no*, ib_i32_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.25 HAILDB_API ib_err_t ib_tuple_write_i64 (ib_tpl_t *ib_tpl*, int *col_no*, ib_i64_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.26 HAILDB_API ib_err_t ib_tuple_write_i8 (ib_tpl_t *ib_tpl*, int *col_no*, ib_i8_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.27 HAILDB_API ib_err_t ib_tuple_write_u16 (ib_tpl_t *ib_tpl*, int *col_no*, ib_u16_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.28 HAILDB_API ib_err_t ib_tuple_write_u32 (ib_tpl_t *ib_tpl*, int *col_no*, ib_u32_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.29 HAILDB_API ib_err_t ib_tuple_write_u64 (ib_tpl_t *ib_tpl*, int *col_no*, ib_u64_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.9.1.30 HAILDB_API ib_err_t ib_tuple_write_u8 (ib_tpl_t *ib_tpl*, int *col_no*, ib_u8_t *val*)

Write an integer value to a column. Integers are stored in big-endian format and will need to be converted from the host format.

Parameters

[in,out] ***ib_tpl*** is the tuple to write to

col_no is the column number to update

val is the value to write

Returns

DB_SUCESS or error

4.10 Debug and Testing functions

Functions

- HAILDB_API ib_err_t ib_error_inject (int error_to_inject)

4.10.1 Function Documentation

4.10.1.1 HAILDB_API ib_err_t ib_error_inject (int *error_to_inject*)

Inject an error into HailDB

This function will simulate an error condition inside HailDB. You should not rely on this function. It is for HailDB test suite use only, parts may only be compiled into debug libraries and this function can quite legitimately just return DB_ERROR and cause Voldemort to pay you a visit.

Parameters

error_to_inject The error inject code to insert.

Chapter 5

Data Structure Documentation

5.1 ib_col_meta_t Struct Reference

Data Fields

- ib_col_type_t type
- ib_col_attr_t attr
- ib_u32_t type_len
- ib_u16_t client_type
- ib_charset_t * charset

5.1.1 Detailed Description

meta data.

5.1.2 Field Documentation

5.1.2.1 ib_col_attr_t attr

Column attributes

5.1.2.2 ib_charset_t* charset

Column charset

5.1.2.3 ib_u16_t client_type

16 bits of data relevant only to the client. InnoDB doesn't care

5.1.2.4 ib_col_type_t type

Type of the column

5.1.2.5 ib_u32_t type_len

Length of type

The documentation for this struct was generated from the following file:

- haildb.h

5.2 ib_schema_visitor_t Struct Reference

```
#include <haildb.h>
```

Data Fields

- ib_schema_visitor_version_t version
- ib_schema_visitor_table_t table
- ib_schema_visitor_table_col_t table_col
- ib_schema_visitor_index_t index
- ib_schema_visitor_index_col_t index_col

5.2.1 Detailed Description

Callback functions to traverse the schema of a table.

5.2.2 Field Documentation

5.2.2.1 ib_schema_visitor_index_t index

For travesing index info

5.2.2.2 ib_schema_visitor_index_col_t index_col

For travesing index column info

5.2.2.3 ib_schema_visitor_table_t table

For travesing table info

5.2.2.4 ib_schema_visitor_table_col_t table_col

For travesing table column info

5.2.2.5 ib_schema_visitor_version_t version

Visitor version

The documentation for this struct was generated from the following file:

- haildb.h

5.3 ib_table_stats_t Struct Reference

Data Fields

- ib_i64_t stat_n_rows
- ib_u64_t stat_clustered_index_size
- ib_u64_t stat_sum_of_other_index_sizes
- ib_u64_t stat_modified_counter

5.3.1 Detailed Description

and index statistics.

5.3.2 Field Documentation

5.3.2.1 ib_u64_t stat_clustered_index_size

approximate clustered index size in bytes.

5.3.2.2 ib_u64_t stat_modified_counter

when a row is inserted, updated, or deleted, we add 1 to this number; we calculate new estimates for the stat_... values for the table and the indexes at an interval of 2 GB or when about 1 / 16 of table has been modified; also when an estimate operation is called for; the counter is reset to zero at statistics calculation; this counter is not protected by any latch, because this is only used for heuristics

5.3.2.3 ib_i64_t stat_n_rows

approximate number of rows in the table; we periodically calculate new estimates

5.3.2.4 ib_u64_t stat_sum_of_other_index_sizes

other indexes in bytes

The documentation for this struct was generated from the following file:

- haildb.h

Chapter 6

File Documentation

6.1 haildb.h File Reference

```
#include <stdio.h>
#include <stdint.h>
```

Data Structures

- struct ib_col_meta_t
- struct ib_schema_visitor_t
- struct ib_table_stats_t

Defines

- #define **HAILDB_API**
- #define **HAILDB_LOCAL**
- #define UNIV_NO_IGNORE
- #define IB_TRUE 0x1UL
- #define IB_FALSE 0x0UL
- #define IB_SQL_NULL 0xFFFFFFFF
- #define IB_N_SYS_COLS 3
- #define MAX_TEXT_LEN 4096
- #define IB_MAX_COL_NAME_LEN (64 * 3)
- #define IB_MAX_TABLE_NAME_LEN (64 * 3)
- #define ib_tbl_sch_add_blob_col(s, n) ib_table_schema_add_col(s, n, IB_BLOB, IB_COL_NONE, 0, 0)
- #define ib_tbl_sch_add_text_col(s, n) ib_table_schema_add_col(s, n, IB_VARCHAR, IB_COL_NONE, 0, MAX_TEXT_LEN)
- #define ib_tbl_sch_add_varchar_col(s, n, l) ib_table_schema_add_col(s, n, IB_VARCHAR, IB_COL_NONE, 0, l)
- #define ib_tbl_sch_add_u32_col(s, n) ib_table_schema_add_col(s, n, IB_INT, IB_COL_UNSIGNED, 0, 4)
- #define ib_tbl_sch_add_u64_col(s, n) ib_table_schema_add_col(s, n, IB_INT, IB_COL_UNSIGNED, 0, 8)

- #define ib_tbl_sch_add_u64_notnull_col(s, n)
- #define ib_cfg_set_int(name, value) ib_cfg_set(name, value)
- #define ib_cfg_set_text(name, value) ib_cfg_set(name, value)
- #define ib_cfg_set_bool_on(name) ib_cfg_set(name, IB_TRUE)
- #define ib_cfg_set_bool_off(name) ib_cfg_set(name, IB_FALSE)
- #define ib_cfg_set_callback(name, value) ib_cfg_set(name, value)

Typedefs

- typedef enum db_err ib_err_t
- typedef unsigned char ib_byte_t
- typedef unsigned long int ib_ulint_t
- typedef void * ib_opaque_t
- typedef ib_ulint_t ib_bool_t
- typedef ib_opaque_t ib_charset_t
- typedef int8_t ib_i8_t
- typedef uint8_t ib_u8_t
- typedef int16_t ib_i16_t
- typedef uint16_t ib_u16_t
- typedef int32_t ib_i32_t
- typedef uint32_t ib_u32_t
- typedef int64_t ib_i64_t
- typedef uint64_t ib_u64_t
- typedef ib_u64_t ib_id_t
- typedef void(* ib_cb_t)(void)
- typedef FILE * ib_msg_stream_t
- typedef int(* ib_msg_log_t)(ib_msg_stream_t, const char *,...)
- typedef struct ib_tpl_struct * ib_tpl_t
- typedef struct ib_trx_struct * ib_trx_t
- typedef struct ib_crsr_struct * ib_crsr_t
- typedef struct ib_tbl_sch_struct * ib_tbl_sch_t
- typedef struct ib_idx_sch_struct * ib_idx_sch_t
- typedef int(* ib_schema_visitor_table_all_t)(void *arg, const char *name, int name_len)
- typedef int(* ib_schema_visitor_table_t)(void *arg, const char *name, ib_tbl_fmt_t tbl_fmt, ib_ulint_t page_size, int n_cols, int n_indexes)
- typedef int(* ib_schema_visitor_table_col_t)(void *arg, const char *name, ib_col_type_t col_type, ib_ulint_t len, ib_col_attr_t attr)
- typedef int(* ib_schema_visitor_index_t)(void *arg, const char *name, ib_bool_t clustered, ib_bool_t unique, int n_cols)
- typedef int(* ib_schema_visitor_index_col_t)(void *arg, const char *name, ib_ulint_t prefix_len)
- typedef int(* ib_client_cmp_t)(const ib_col_meta_t *col_meta, const ib_byte_t *p1, ib_ulint_t p1_len, const ib_byte_t *p2, ib_ulint_t p2_len)
- typedef void(* ib_panic_handler_t)(void *, int, char *,...)
- typedef int(* ib_trx_is_interrupted_handler_t)(void *)

Enumerations

- enum db_err {

  DB_SUCCESS = 10, DB_ERROR, DB_INTERRUPTED, DB_OUT_OF_MEMORY,

  DB_OUT_OF_FILE_SPACE, DB_LOCK_WAIT, DB_DEADLOCK, DB_ROLLBACK,

  DB_DUPLICATE_KEY, DB_QUE_THR_SUSPENDED, DB_MISSING_HISTORY, DB_-CLUSTER_NOT_FOUND = 30,

  DB_TABLE_NOT_FOUND, DB_MUST_GET_MORE_FILE_SPACE, DB_TABLE_IS_BEING_-USED, DB_TOO_BIG_RECORD,

  DB_LOCK_WAIT_TIMEOUT, DB_NO_REFERENCED_ROW, DB_ROW_IS_REFERENCED, DB_CANNOT_ADD_CONSTRAINT,

  DB_CORRUPTION, DB_COL_APPEARS_TWICE_IN_INDEX, DB_CANNOT_DROP_-CONSTRAINT, DB_NO_SAVEPOINT,

  DB_TABLESPACE_ALREADY_EXISTS, DB_TABLESPACE_DELETED, DB_LOCK_-TABLE_FULL, DB_FOREIGN_DUPLICATE_KEY,

  DB_TOO_MANY_CONCURRENT_TRXS, DB_UNSUPPORTED, DB_PRIMARY_KEY_IS_-NULL, DB_FATAL,

  DB_FAIL = 1000, DB_OVERFLOW, DB_UNDERFLOW, DB_STRONG_FAIL,

  DB_ZIP_OVERFLOW, DB_RECORD_NOT_FOUND = 1500, DB_END_OF_INDEX, DB_-SCHEMA_ERROR = 2000,

  DB_DATA_MISMATCH, DB_SCHEMA_NOT_LOCKED, DB_NOT_FOUND, DB_-READONLY,

  DB_INVALID_INPUT }
- enum ib_cfg_type_t {

  IB_CFG_IBOOL, IB_CFG_ULINT, IB_CFG_ULONG, IB_CFG_TEXT,

  IB_CFG_CB }
- enum ib_col_type_t {

  IB_VARCHAR = 1, IB_CHAR = 2, IB_BINARY = 3, IB_VARBINARY = 4,

  IB_BLOB = 5, IB_INT = 6, IB_SYS = 8, IB_FLOAT = 9,

  IB_DOUBLE = 10, IB_DECIMAL = 11, IB_VARCHAR_ANYCHARSET = 12, IB_CHAR_-ANYCHARSET = 13 }
- enum ib_tbl_fmt_t { IB_TBL_REDUNDANT, IB_TBL_COMPACT, IB_TBL_DYNAMIC, IB_-TBL_COMPRESSED }
- enum ib_col_attr_t {

  IB_COL_NONE = 0, IB_COL_NOT_NULL = 1, IB_COL_UNSIGNED = 2, IB_COL_NOT_USED = 4,

  IB_COL_CUSTOM1 = 8, IB_COL_CUSTOM2 = 16, IB_COL_CUSTOM3 = 32 }
- enum ib_lck_mode_t {

  IB_LOCK_IS = 0, IB_LOCK_IX, IB_LOCK_S, IB_LOCK_X,

  IB_LOCK_NOT_USED, IB_LOCK_NONE, IB_LOCK_NUM = IB_LOCK_NONE }
- enum ib_srch_mode_t { IB_CUR_G = 1, IB_CUR_GE = 2, IB_CUR_L = 3, IB_CUR_LE = 4 }
- enum ib_match_mode_t { IB_CLOSEST_MATCH, IB_EXACT_MATCH, IB_EXACT_PREFIX }
- enum ib_trx_state_t { IB_TRX_NOT_STARTED, IB_TRX_ACTIVE, IB_TRX_COMMITTED_-IN_MEMORY, IB_TRX_PREPARED }
- enum ib_trx_level_t { IB_TRX_READ_UNCOMMITTED = 0, IB_TRX_READ_COMMITTED = 1, IB_TRX_REPEATABLE_READ = 2, IB_TRX_SERIALIZABLE = 3 }

- enum ib_shutdown_t { IB_SHUTDOWN_NORMAL, IB_SHUTDOWN_NO_IBUFMERGE_PURGE, IB_SHUTDOWN_NO_BUFPOOL_FLUSH }
- enum ib_schema_visitor_version_t { **IB_SCHEMA_VISITOR_TABLE** = 1, **IB_SCHEMA_VISITOR_TABLE_COL** = 2, **IB_SCHEMA_VISITOR_TABLE_AND_INDEX** = 3, **IB_SCHEMA_VISITOR_TABLE_AND_INDEX_COL** = 4 }

Functions

- HAILDB_API ib_u64_t ib_api_version (void) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_init (void) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_startup (const char *format) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_shutdown (ib_shutdown_t flag) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_start (ib_trx_t ib_trx, ib_trx_level_t ib_trx_level) UNIV_NO_IGNORE
- HAILDB_API ib_trx_t ib_trx_begin (ib_trx_level_t ib_trx_level) UNIV_NO_IGNORE
- HAILDB_API void ib_trx_set_client_data (ib_trx_t ib_trx, void *client_data)
- HAILDB_API ib_trx_state_t ib_trx_state (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_release (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_commit (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_trx_rollback (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_schema_add_col (ib_tbl_sch_t ib_tbl_sch, const char *name, ib_col_type_t ib_col_type, ib_col_attr_t ib_col_attr, ib_u16_t client_type, ib_ulint_t len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_schema_add_index (ib_tbl_sch_t ib_tbl_sch, const char *name, ib_idx_sch_t *ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API void ib_table_schema_delete (ib_tbl_sch_t ib_tbl_sch)
- HAILDB_API ib_err_t ib_table_schema_create (const char *name, ib_tbl_sch_t *ib_tbl_sch, ib_tbl_fmt_t ib_tbl_fmt, ib_ulint_t page_size) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_add_col (ib_idx_sch_t ib_idx_sch, const char *name, ib_ulint_t prefix_len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_create (ib_trx_t ib_usr_trx, const char *name, const char *table_name, ib_idx_sch_t *ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_schema_set_clustered (ib_idx_sch_t ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API void ib_cursor_set_simple_select (ib_crsr_t ib_crsr)
- HAILDB_API ib_err_t ib_index_schema_set_unique (ib_idx_sch_t ib_idx_sch) UNIV_NO_IGNORE
- HAILDB_API void ib_index_schema_delete (ib_idx_sch_t ib_idx_sch)
- HAILDB_API ib_err_t ib_table_create (ib_trx_t ib_trx, const ib_tbl_sch_t ib_tbl_sch, ib_id_t *id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_rename (ib_trx_t ib_trx, const char *old_name, const char *new_name) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_create (ib_idx_sch_t ib_idx_sch, ib_id_t *index_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_drop (ib_trx_t trx, const char *name) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_drop (ib_trx_t trx, ib_id_t index_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_open_table_using_id (ib_id_t table_id, ib_trx_t ib_trx, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_open_index_using_id (ib_id_t index_id, ib_trx_t ib_trx, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE

- HAILDB_API ib_err_t ib_cursor_open_index_using_name (ib_crsr_t ib_open_crsr, const char *index_name, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_open_table (const char *name, ib_trx_t ib_trx, ib_crsr_t *ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_reset (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_close (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_insert_row (ib_crsr_t ib_crsr, const ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_update_row (ib_crsr_t ib_crsr, const ib_tpl_t ib_old_tpl, const ib_tpl_t ib_new_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_delete_row (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_read_row (ib_crsr_t ib_crsr, ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_prev (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_next (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_first (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_last (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_moveto (ib_crsr_t ib_crsr, ib_tpl_t ib_tpl, ib_srch_mode_t ib_srch_mode, int *result) UNIV_NO_IGNORE
- HAILDB_API void ib_cursor_attach_trx (ib_crsr_t ib_crsr, ib_trx_t ib_trx)
- HAILDB_API void ib_set_client_compare (ib_client_cmp_t client_cmp_func)
- HAILDB_API void ib_cursor_set_match_mode (ib_crsr_t ib_crsr, ib_match_mode_t match_mode)
- HAILDB_API ib_err_t ib_col_set_value (ib_tpl_t ib_tpl, ib_ulint_t col_no, const void *src, ib_ulint_t len) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_col_get_len (ib_tpl_t ib_tpl, ib_ulint_t i) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_col_copy_value (ib_tpl_t ib_tpl, ib_ulint_t i, void *dst, ib_ulint_t len)
- HAILDB_API ib_err_t ib_tuple_read_i8 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i8_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u8 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u8_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_i16 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i16_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u16 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u16_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_i32 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i32_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u32 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u32_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_i64 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_i64_t *ival) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_read_u64 (ib_tpl_t ib_tpl, ib_ulint_t i, ib_u64_t *ival) UNIV_NO_IGNORE
- HAILDB_API const void * ib_col_get_value (ib_tpl_t ib_tpl, ib_ulint_t i) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_col_get_meta (ib_tpl_t ib_tpl, ib_ulint_t i, ib_col_meta_t *ib_col_meta)
- HAILDB_API ib_tpl_t ib_tuple_clear (ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_get_cluster_key (ib_crsr_t ib_crsr, ib_tpl_t *ib_dst_tpl, const ib_tpl_t ib_src_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_copy (ib_tpl_t ib_dst_tpl, const ib_tpl_t ib_src_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_sec_search_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_sec_read_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_tpl_t ib_clust_search_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE

- HAILDB_API ib_tpl_t ib_clust_read_tuple_create (ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_tuple_get_n_user_cols (const ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API ib_ulint_t ib_tuple_get_n_cols (const ib_tpl_t ib_tpl) UNIV_NO_IGNORE
- HAILDB_API void ib_tuple_delete (ib_tpl_t ib_tpl)
- HAILDB_API ib_err_t ib_cursor_truncate (ib_crsr_t *ib_crsr, ib_id_t *table_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_truncate (const char *table_name, ib_id_t *table_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_get_id (const char *table_name, ib_id_t *table_id) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_index_get_id (const char *table_name, const char *index_name, ib_id_t *index_id) UNIV_NO_IGNORE
- HAILDB_API ib_bool_t ib_database_create (const char *dbname) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_database_drop (const char *dbname) UNIV_NO_IGNORE
- HAILDB_API ib_bool_t ib_cursor_is_positioned (const ib_crsr_t ib_crsr) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_lock_shared (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_lock_exclusive (ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_bool_t ib_schema_lock_is_exclusive (const ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_bool_t ib_schema_lock_is_shared (const ib_trx_t ib_trx) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_unlock (ib_trx_t ib_trx)
- HAILDB_API ib_err_t ib_cursor_lock (ib_crsr_t ib_crsr, ib_lck_mode_t ib_lck_mode) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_table_lock (ib_trx_t ib_trx, ib_id_t table_id, ib_lck_mode_t ib_lck_mode) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cursor_set_lock_mode (ib_crsr_t ib_crsr, ib_lck_mode_t ib_lck_mode) UNIV_NO_IGNORE
- HAILDB_API void ib_cursor_set_cluster_access (ib_crsr_t ib_crsr)
- HAILDB_API ib_err_t ib_table_schema_visit (ib_trx_t ib_trx, const char *name, const ib_schema_visitor_t *visitor, void *arg) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_schema_tables_iterate (ib_trx_t ib_trx, ib_schema_visitor_table_all_t visitor, void *arg) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_var_get_type (const char *name, ib_cfg_type_t *type) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_set (const char *name,...) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_get (const char *name, void *value) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_cfg_get_all (const char ***names, ib_u32_t *names_num) UNIV_NO_IGNORE
- HAILDB_API void ib_savepoint_take (ib_trx_t ib_trx, const void *name, ib_ulint_t name_len)
- HAILDB_API ib_err_t ib_savepoint_release (ib_trx_t ib_trx, const void *name, ib_ulint_t name_len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_savepoint_rollback (ib_trx_t ib_trx, const void *name, ib_ulint_t name_len) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i8 (ib_tpl_t ib_tpl, int col_no, ib_i8_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i16 (ib_tpl_t ib_tpl, int col_no, ib_i16_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i32 (ib_tpl_t ib_tpl, int col_no, ib_i32_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_i64 (ib_tpl_t ib_tpl, int col_no, ib_i64_t val) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u8 (ib_tpl_t ib_tpl, int col_no, ib_u8_t val) UNIV_NO_IGNORE

- HAILDB_API ib_err_t ib_tuple_write_u16 (ib_tpl_t ib_tpl, int col_no, ib_u16_t val) UNIV_NO_-IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u32 (ib_tpl_t ib_tpl, int col_no, ib_u32_t val) UNIV_NO_-IGNORE
- HAILDB_API ib_err_t ib_tuple_write_u64 (ib_tpl_t ib_tpl, int col_no, ib_u64_t val) UNIV_NO_-IGNORE
- HAILDB_API void ib_cursor_stmt_begin (ib_crsr_t ib_crsr)
- HAILDB_API ib_err_t ib_tuple_write_double (ib_tpl_t ib_tpl, int col_no, double val) UNIV_NO_-IGNORE
- HAILDB_API ib_err_t ib_tuple_read_double (ib_tpl_t ib_tpl, ib_ulint_t col_no, double *dval) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_tuple_write_float (ib_tpl_t ib_tpl, int col_no, float val) UNIV_NO_-IGNORE
- HAILDB_API ib_err_t ib_tuple_read_float (ib_tpl_t ib_tpl, ib_ulint_t col_no, float *fval) UNIV_-NO_IGNORE
- HAILDB_API void ib_logger_set (ib_msg_log_t ib_msg_log, ib_msg_stream_t ib_msg_stream)
- HAILDB_API const char * ib_strerror (ib_err_t db_errno) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_status_get_i64 (const char *name, ib_i64_t *dst) UNIV_NO_IGNORE
- HAILDB_API ib_err_t ib_status_get_all (const char ***names, ib_u32_t *names_num) UNIV_-NO_IGNORE
- HAILDB_API void ib_set_panic_handler (ib_panic_handler_t new_panic_handler)
- HAILDB_API void ib_set_trx_is_interrupted_handler (ib_trx_is_interrupted_handler_t handler)
- HAILDB_API ib_err_t ib_get_duplicate_key (ib_trx_t ib_trx, const char **table_name, const char **index_name)
- HAILDB_API ib_err_t ib_get_table_statistics (ib_crsr_t ib_crsr, ib_table_stats_t *table_stats, size_t sizeof_ib_table_stats_t)
- HAILDB_API ib_err_t ib_get_index_stat_n_diff_key_vals (ib_crsr_t ib_crsr, const char *index_-name, ib_u64_t *ncols, ib_i64_t **n_diff)
- HAILDB_API ib_err_t ib_update_table_statistics (ib_crsr_t crsr)
- HAILDB_API ib_err_t ib_error_inject (int error_to_inject)

Variables

- ib_client_cmp_t ib_client_compare

6.1.1 Detailed Description

<

6.1.2 Define Documentation

6.1.2.1 #define IB_FALSE 0x0UL

The boolean value of "false" used internally within InnoDB

6.1.2.2 #define IB_MAX_COL_NAME_LEN (64 * 3)

The maximum length of a column name in a table schema.

6.1.2.3 #define IB_MAX_TABLE_NAME_LEN (64 * 3)

The maximum length of a table name (plus database name).

6.1.2.4 #define IB_N_SYS_COLS 3

The number of system columns in a row.

6.1.2.5 #define IB_SQL_NULL 0xFFFFFFFF

Represents SQL_NULL length

6.1.2.6 #define ib_tbl_sch_add_blob_col(*s*, *n*) ib_table_schema_add_col(s, n, IB_BLOB, IB_COL_NONE, 0, 0)

Add a BLOB column to a table schema.

Parameters

s is the the schema handle

n is the column name

6.1.2.7 #define ib_tbl_sch_add_text_col(*s*, *n*) ib_table_schema_add_col(s, n, IB_VARCHAR, IB_COL_NONE, 0, MAX_TEXT_LEN)

Add a BLOB column to a table schema.

Parameters

s is the the schema handle

n is the column name Add a TEXT column to a table schema.

6.1.2.8 #define ib_tbl_sch_add_u32_col(*s*, *n*) ib_table_schema_add_col(s, n, IB_INT, IB_COL_UNSIGNED, 0, 4)

Add an UNSIGNED INT column to a table schema.

Parameters

s is the schema handle

n is the column name

Returns

DB_SUCCESS or error code

6.1.2.9 #define ib_tbl_sch_add_u64_col(*s*, *n*) ib_table_schema_add_col(s, n, IB_INT, IB_COL_UNSIGNED, 0, 8)

Add an UNSIGNED BIGINT column to a table schema.

Parameters

s is the schema handle

n is the column name

Returns

DB_SUCCESS or error code

6.1.2.10 #define ib_tbl_sch_add_u64_notnull_col(*s*, *n*)

Value:

```
ib_table_schema_add_col(s, n, IB_INT, \
                                IB_COL_NOT_NULL | IB_COL_UNSIGNED,0,\
                                8)
```

Add an UNSIGNED BIGINT NOT NULL column to a table schema.

Parameters

s is the schema handle

n is the column name

Returns

DB_SUCCESS or error code

6.1.2.11 #define ib_tbl_sch_add_varchar_col(*s*, *n*, *l*) ib_table_schema_add_col(s, n, IB_VARCHAR, IB_COL_NONE, 0, l)

Add a VARCHAR column to a table schema.

Parameters

s is the schema handle

n is the column name

l the max length of the VARCHAR column

Returns

DB_SUCCESS or error code

6.1.2.12 #define IB_TRUE 0x1UL

The boolean value of "true" used internally within InnoDB

6.1.2.13 #define MAX_TEXT_LEN 4096

The maximum length of a text column.

6.1.2.14 #define UNIV_NO_IGNORE

Some HailDB methods will produce warnings if the result of them is not checked in client code. This uses a GCC compiler extension.

6.1.3 Typedef Documentation

6.1.3.1 typedef ib_ulint_t ib_bool_t

Representation of a "boolean" type within InnoDB

6.1.3.2 typedef unsigned char ib_byte_t

Representation of a byte within InnoDB

6.1.3.3 typedef void(* ib_cb_t)(void)

Generical InnoDB callback prototype.

6.1.3.4 typedef ib_opaque_t ib_charset_t

A character set pointer

6.1.3.5 typedef int(* ib_client_cmp_t)(const ib_col_meta_t *col_meta,const ib_byte_t *p1,ib_ulint_t p1_len,const ib_byte_t *p2,ib_ulint_t p2_len)

This function is used to compare two data fields for which the data type is such that we must use the client code to compare them. in: key length

6.1.3.6 typedef struct ib_crsr_struct* ib_crsr_t

InnoDB cursor handle

6.1.3.7 typedef enum db_err ib_err_t

All InnoDB error codes are represented by ib_err_t. See db_err for a complete list of possible error codes.

6.1.3.8 typedef int16_t ib_i16_t

A signed 16 bit integral type.

6.1.3.9 typedef int32_t ib_i32_t

A signed 32 bit integral type.

6.1.3.10 typedef int64_t ib_i64_t

A signed 64 bit integral type.

6.1.3.11 typedef int8_t ib_i8_t

A signed 8 bit integral type.

6.1.3.12 typedef ib_u64_t ib_id_t

The integral type that represents internal table and index ids.

6.1.3.13 typedef struct ib_idx_sch_struct* ib_idx_sch_t

InnoDB index schema handle

6.1.3.14 typedef int(* ib_msg_log_t)(ib_msg_stream_t, const char *,...)

All log messages are written to this function.It should have the same behavior as fprintf(3).

6.1.3.15 typedef FILE* ib_msg_stream_t

The first argument to the InnoDB message logging function. By default it's set to stderr. You should treat ib_msg_stream_t as a void*, since it will probably change in the future.

6.1.3.16 typedef void* ib_opaque_t

Representation of a void* within InnoDB

6.1.3.17 typedef void(* ib_panic_handler_t)(void *, int, char *,...)

Type of callback in the event of HailDB panicing. Your callback should call exit() rather soon, as continuing after a panic will lead to errors returned from every API function. We have also not fully tested every possible outcome from not immediately calling exit().

6.1.3.18 typedef int(* ib_schema_visitor_index_col_t)(void *arg,const char *name,ib_ulint_t prefix_len)

Index column visitor Prefix length

6.1.3.19 typedef int(∗ ib_schema_visitor_index_t)(void ∗arg,const char ∗name,ib_bool_t clustered,ib_bool_t unique,int n_cols)

Index visitor No. of cols defined

6.1.3.20 typedef int(∗ ib_schema_visitor_table_all_t)(void ∗arg,const char ∗name,int name_len)

Visit all tables in the InnoDB schem. Length of name in bytes

6.1.3.21 typedef int(∗ ib_schema_visitor_table_col_t)(void ∗arg,const char ∗name,ib_col_type_t col_type,ib_ulint_t len,ib_col_attr_t attr)

Table column visitor Column attributes

6.1.3.22 typedef int(∗ ib_schema_visitor_table_t)(void ∗arg,const char ∗name,ib_tbl_fmt_t tbl_fmt,ib_ulint_t page_size,int n_cols,int n_indexes)

Table visitor No. of indexes defined

6.1.3.23 typedef struct ib_tbl_sch_struct∗ ib_tbl_sch_t

InnoDB table schema handle

6.1.3.24 typedef struct ib_tpl_struct∗ ib_tpl_t

InnoDB tuple handle. This handle can refer to either a cluster index tuple or a secondary index tuple. There are two types of tuples for each type of index, making a total of four types of tuple handles. There is a tuple for reading the entire row contents and another for searching on the index key.

6.1.3.25 typedef int(∗ ib_trx_is_interrupted_handler_t)(void ∗)

Callback for checking if a transaction has been interrupted. This callback lets you implement the MySQL KILL command kind of functionality. A transaction may block in the thread it's running in (for example, while acquiring row locks or doing IO) but other threads may do something that causes ib_trx_is_interrupted_handler_t to return true.

6.1.3.26 typedef struct ib_trx_struct∗ ib_trx_t

InnoDB transaction handle, all database operations need to be covered by transactions. This handle represents a transaction. The handle can be created with ib_trx_begin(), you commit your changes with ib_trx_commit() and undo your changes using ib_trx_rollback(). If the InnoDB deadlock monitor rolls back the transaction then you need to free the transaction using the function ib_trx_release(). You can query the state of an InnoDB transaction by calling ib_trx_state().

6.1.3.27 typedef uint16_t ib_u16_t

An unsigned 16 bit integral type.

6.1.3.28 typedef uint32_t ib_u32_t

An unsigned 32 bit integral type.

6.1.3.29 typedef uint64_t ib_u64_t

An unsigned 64 bit integral type.

6.1.3.30 typedef uint8_t ib_u8_t

An unsigned 8 bit integral type.

6.1.3.31 typedef unsigned long int ib_ulint_t

Representation of an unsigned long int within InnoDB

6.1.4 Enumeration Type Documentation

6.1.4.1 enum db_err

InnoDB error codes. Most of the error codes are internal to the engine and will not be seen by user applications. The partial error codes reflect the sub-state of an operation within InnoDB. Some of the error codes are deprecated and are no longer used.

Enumerator:

DB_SUCCESS A successult result

DB_ERROR This is a generic error code. It is used to classify error conditions that can't be represented by other codes

DB_INTERRUPTED An operation was interrupted by a user.

DB_OUT_OF_MEMORY Operation caused an out of memory error. Within InnoDB core code this is normally a fatal error

DB_OUT_OF_FILE_SPACE The operating system returned an out of file space error when trying to do an IO operation.

DB_LOCK_WAIT A lock request by transaction resulted in a lock wait. The thread is suspended internally by InnoDB and is put on a lock wait queue.

DB_DEADLOCK A lock request by a transaction resulted in a deadlock. The transaction was rolled back

DB_ROLLBACK Not used

DB_DUPLICATE_KEY A record insert or update violates a unique contraint.

DB_QUE_THR_SUSPENDED A query thread should be in state suspended but is trying to acquire a lock. Currently this is treated as a hard error and a violation of an invariant.

DB_MISSING_HISTORY Required history data has been deleted due to lack of space in rollback segment

DB_CLUSTER_NOT_FOUND This error is not used

DB_TABLE_NOT_FOUND The table could not be found

DB_MUST_GET_MORE_FILE_SPACE The database has to be stopped and restarted with more file space

DB_TABLE_IS_BEING_USED The user is trying to create a table in the InnoDB data dictionary but a table with that name already exists

DB_TOO_BIG_RECORD A record in an index would not fit on a compressed page, or it would become bigger than 1/2 free space in an uncompressed page frame

DB_LOCK_WAIT_TIMEOUT Lock wait lasted too long

DB_NO_REFERENCED_ROW Referenced key value not found for a foreign key in an insert or update of a row

DB_ROW_IS_REFERENCED Cannot delete or update a row because it contains a key value which is referenced

DB_CANNOT_ADD_CONSTRAINT Adding a foreign key constraint to a table failed

DB_CORRUPTION Data structure corruption noticed

DB_COL_APPEARS_TWICE_IN_INDEX InnoDB cannot handle an index where same column appears twice

DB_CANNOT_DROP_CONSTRAINT Dropping a foreign key constraint from a table failed

DB_NO_SAVEPOINT No savepoint exists with the given name

DB_TABLESPACE_ALREADY_EXISTS We cannot create a new single-table tablespace because a file of the same name already exists

DB_TABLESPACE_DELETED Tablespace does not exist or is being dropped right now

DB_LOCK_TABLE_FULL Lock structs have exhausted the buffer pool (for big transactions, InnoDB stores the lock structs in the buffer pool)

DB_FOREIGN_DUPLICATE_KEY Foreign key constraints activated but the operation would lead to a duplicate key in some table

DB_TOO_MANY_CONCURRENT_TRXS When InnoDB runs out of the preconfigured undo slots, this can only happen when there are too many concurrent transactions

DB_UNSUPPORTED When InnoDB sees any artefact or a feature that it can't recoginize or work with e.g., FT indexes created by a later version of the engine.

DB_PRIMARY_KEY_IS_NULL A column in the PRIMARY KEY was found to be NULL

DB_FATAL The application should clean up and quite ASAP. Fatal error, InnoDB cannot continue operation without risking database corruption.

DB_FAIL Partial failure code.

DB_OVERFLOW If an update or insert of a record doesn't fit in a Btree page

DB_UNDERFLOW If an update or delete of a record causes a Btree page to be below a minimum threshold

DB_STRONG_FAIL Failure to insert a secondary index entry to the insert buffer

DB_ZIP_OVERFLOW Failure trying to compress a page

DB_RECORD_NOT_FOUND Record not found

DB_END_OF_INDEX A cursor operation or search operation scanned to the end of the index.

DB_SCHEMA_ERROR Generic schema error

DB_DATA_MISMATCH Column update or read failed because the types mismatch

DB_SCHEMA_NOT_LOCKED If an API function expects the schema to be locked in exclusive mode and if it's not then that API function will return this error code

DB_NOT_FOUND Generic error code for "Not found" type of errors

DB_READONLY Generic error code for "Readonly" type of errors

DB_INVALID_INPUT Generic error code for "Invalid input" type of errors

6.1.4.2 enum ib_cfg_type_t

Possible types for a configuration variable.

Enumerator:

IB_CFG_IBOOL The configuration parameter is of type ibool

IB_CFG_ULINT The configuration parameter is of type ulint

IB_CFG_ULONG The configuration parameter is of type ulong

IB_CFG_TEXT The configuration parameter is of type char∗

IB_CFG_CB The configuration parameter is a callback parameter

6.1.4.3 enum ib_col_attr_t

InnoDB column attributes

Enumerator:

IB_COL_NONE No special attributes.

IB_COL_NOT_NULL Column data can't be NULL.

IB_COL_UNSIGNED Column is IB_INT and unsigned.

IB_COL_NOT_USED Future use, reserved.

IB_COL_CUSTOM1 Custom precision type, this is a bit that is ignored by InnoDB and so can be set and queried by users.

IB_COL_CUSTOM2 Custom precision type, this is a bit that is ignored by InnoDB and so can be set and queried by users.

IB_COL_CUSTOM3 Custom precision type, this is a bit that is ignored by InnoDB and so can be set and queried by users.

6.1.4.4 enum ib_col_type_t

column types that are supported.

Enumerator:

IB_VARCHAR Character varying length. The column is not padded.

IB_CHAR Fixed length character string. The column is padded to the right.

IB_BINARY Fixed length binary, similar to IB_CHAR but the column is not padded to the right.

IB_VARBINARY Variable length binary

IB_BLOB Binary large object, or a TEXT type

IB_INT Integer: can be any size from 1 - 8 bytes. If the size is 1, 2, 4 and 8 bytes then you can use the typed read and write functions. For other sizes you will need to use the ib_col_get_value() function and do the conversion yourself.

IB_SYS System column, this column can be one of DATA_TRX_ID, DATA_ROLL_PTR or DATA_-ROW_ID.

IB_FLOAT C (float) floating point value.

IB_DECIMAL > C (double) floating point value. Decimal stored as an ASCII string

IB_VARCHAR_ANYCHARSET Any charset, varying length

IB_CHAR_ANYCHARSET Any charset, fixed length

6.1.4.5 enum ib_lck_mode_t

InnoDB lock modes.

Enumerator:

IB_LOCK_IS Intention shared, an intention lock should be used to lock tables

IB_LOCK_IX Intention exclusive, an intention lock should be used to lock tables

IB_LOCK_S Shared locks should be used to lock rows

IB_LOCK_X Exclusive locks should be used to lock rows

IB_LOCK_NOT_USED Future use, reserved

IB_LOCK_NONE This is used internally to note consistent read

IB_LOCK_NUM number of lock modes

6.1.4.6 enum ib_match_mode_t

Various match modes used by ib_cursor_moveto()

Enumerator:

IB_CLOSEST_MATCH Closest match possible

IB_EXACT_MATCH Search using a complete key value

IB_EXACT_PREFIX Search using a key prefix which must match to rows: the prefix may contain an incomplete field (the last field in prefix may be just a prefix of a fixed length column)

6.1.4.7 enum ib_schema_visitor_version_t

Currently, this is also the number of callback functions in the struct.

6.1.4.8 enum ib_shutdown_t

When ib_shutdown() is called InnoDB may take a long time to shutdown because of background tasks e.g., purging deleted records. The following flags allow the user to control the shutdown behavior.

Enumerator:

IB_SHUTDOWN_NORMAL Normal shutdown, do insert buffer merge and purge before complete shutdown.

IB_SHUTDOWN_NO_IBUFMERGE_PURGE Do not do a purge and index buffer merge at shutdown.

IB_SHUTDOWN_NO_BUFPOOL_FLUSH Same as NO_IBUFMERGE_PURGE and in addition do not even flush the buffer pool to data files. No committed transactions are lost

6.1.4.9 enum ib_srch_mode_t

InnoDB cursor search modes for ib_cursor_moveto(). Note: Values must match those found in page0cur.h

Enumerator:

IB_CUR_G If search key is not found then position the cursor on the row that is greater than the search key

IB_CUR_GE If the search key not found then position the cursor on the row that is greater than or equal to the search key

IB_CUR_L If search key is not found then position the cursor on the row that is less than the search key

IB_CUR_LE If search key is not found then position the cursor on the row that is less than or equal to the search key

6.1.4.10 enum ib_tbl_fmt_t

InnoDB table format types

Enumerator:

IB_TBL_REDUNDANT Redundant row format, the column type and length is stored in the row.

IB_TBL_COMPACT Compact row format, the column type is not stored in the row. The length is stored in the row but the storage format uses a compact format to store the length of the column data and record data storage format also uses less storage.

IB_TBL_DYNAMIC Compact row format. BLOB prefixes are not stored in the clustered index

IB_TBL_COMPRESSED Similar to dynamic format but with pages compressed

6.1.4.11 enum ib_trx_level_t

Transaction isolation levels

Enumerator:

IB_TRX_READ_UNCOMMITTED Dirty read: non-locking SELECTs are performed so that we do not look at a possible earlier version of a record; thus they are not 'consistent' reads under this isolation level; otherwise like level 2

IB_TRX_READ_COMMITTED Somewhat Oracle-like isolation, except that in range UPDATE and DELETE we must block phantom rows with next-key locks; SELECT ... FOR UPDATE and ... LOCK IN SHARE MODE only lock the index records, NOT the gaps before them, and thus allow free inserting; each consistent read reads its own snapshot

IB_TRX_REPEATABLE_READ All consistent reads in the same trx read the same snapshot; full next-key locking used in locking reads to block insertions into gaps

IB_TRX_SERIALIZABLE All plain SELECTs are converted to LOCK IN SHARE MODE reads

6.1.4.12 enum ib_trx_state_t

The transaction state can be queried using the ib_trx_state() function. The InnoDB deadlock monitor can roll back a transaction and users should be prepared for this, especially where there is high contention. The way to determine the state of the transaction is to query it's state and check.

Enumerator:

IB_TRX_NOT_STARTED Has not started yet, the transaction has not ben started yet.

IB_TRX_ACTIVE The transaction is currently active and needs to be either committed or rolled back.

IB_TRX_COMMITTED_IN_MEMORY Not committed to disk yet

IB_TRX_PREPARED Support for 2PC/XA

6.1.5 Function Documentation

6.1.5.1 HAILDB_API ib_bool_t ib_schema_lock_is_shared (const ib_trx_t *ib_trx*)

Checks if the data dictionary is latched in shared mode.

Parameters

ib_trx is a transaction instance

Returns

TRUE if shared latch

6.1.6 Variable Documentation

6.1.6.1 ib_client_cmp_t ib_client_compare

Callback function to compare InnoDB key columns in an index.

Index