secondary index

```
```

poolreg.xsat

```
/**
 * @brief synchronizer table.
 * @scope `get_self()`
 *
 * @field synchronizer - synchronizer address
 * @field reward_recipient - get rewards to address
 * @field memo - memo when receiving rewards
 * @field num_slots - number of slots owned
 * @field latest_produced_block_height - the height of the latest mined block
 * @field produced_block_limit - upload limit, for example, the number of blocks in 432 can
only be uploaded. 0 is
 * not limited
 * @field unclaimed - amount of unclaimed rewards
 * @field claimed  - amount of claimed rewards
 * @field latest_reward_block  - amount of claimed rewards
 * @field latest_reward_time  - amount of claimed rewards
 *
 */
struct [[eosio::table]] synchronizer_row {
    name synchronizer;
    name reward_recipient;
    string memo;
    uint16_t num_slots;
    uint64_t latest_produced_block_height;
    uint16_t produced_block_limit;
    asset unclaimed;
    asset claimed;
    uint64_t latest_reward_block;
    time_point_sec latest_reward_time;
    uint64_t primary_key() const { return synchronizer.value; }
};
typedef eosio::multi_index<"synchronizer"_n, synchronizer_row> synchronizer_table;

/**
 * @brief miner table.
 * @scope `get_self()`
 *
 * @field id - primary key
 * @field synchronizer - synchronizer address
 * @field miner - miner address
 *
 */
struct [[eosio::table]] miner_row {
    uint64_t id;
    name synchronizer;
    std::string miner;
    uint64_t primary_key() const { return id; }
    uint64_t by_syncer() const { return synchronizer.value; }
    checksum256 by_miner() const { return xsat::utils::hash(miner); }
};
typedef eosio::multi_index<
    "miners"_n, miner_row,
    eosio::indexed_by<"bysyncer"_n, const_mem_fun<miner_row, uint64_t,
&miner_row::by_syncer>>,
    eosio::indexed_by<"byminer"_n, const_mem_fun<miner_row, checksum256,
&miner_row::by_miner>>>
    miner_table;
```

rescmng.xsat

```
/**
 * @brief config table.
 * @scope `get_self()`
 *
 * @field fee_account - account number for receiving handling fees
 * @field disabled_withdraw - whether withdrawal of balance is allowed
 * @field cost_per_slot - cost per slot
 * @field cost_per_upload - cost per upload chunk
 * @field cost_per_verification - the cost of each verification performed
 * @field cost_per_endorsement - the cost of each execution of an endorsement
 * @field cost_per_parse - cost per execution of parsing
 *
 **/
struct [[eosio::table]] config_row {
    name fee_account;
    bool disabled_withdraw = false;
    asset cost_per_slot = {0, BTC_SYMBOL};
    asset cost_per_upload = {0, BTC_SYMBOL};
    asset cost_per_verification = {0, BTC_SYMBOL};
    asset cost_per_endorsement = {0, BTC_SYMBOL};
    asset cost_per_parse = {0, BTC_SYMBOL};
};
typedef eosio::singleton<"config"_n, config_row> config_table;

/**
 * @brief account balance table.
 * @scope `get_self()`
 *
 * @field owner - primary key
 * @field balance - account balance
 *
 **/
struct [[eosio::table]] account_row {
    name owner;
    asset balance;
    uint64_t primary_key() const { return owner.value; }
};
typedef eosio::multi_index<"accounts"_n, account_row> account_table;
```

staking.xsat

```
/**
 * @brief global id table.
 * @scope `get_self()`
 *
 * @field staking_id - stake number
 * @field release_id - the number of the release pledge
 */
struct [[eosio::table()]] global_id_row {
    uint64_t staking_id;
    uint64_t release_id;
};
typedef eosio::singleton<"globalid"_n, global_id_row> global_id_table;

/**
 * @brief whitelist token table.
 * @scope `get_self()`
 *
 * @field id - primary key
 * @field token - whitelist token
 *
 */
struct [[eosio::table]] token_row {
    uint64_t id;
    extended_symbol token;
    bool disabled_staking;
    uint64_t primary_key() const { return id; }
    uint128_t by_token() const { return xsat::utils::compute_id(token); }
};
typedef eosio::multi_index<"tokens"_n, token_row,
                           indexed_by<"bytoken"_n, const_mem_fun<token_row, uint128_t,
&token_row::by_token>>>
    token_table;

/**
 * @brief staking table.
 * @scope `staker`
 *
 * @field id - primary key
 * @field quantity - total number of stakes
 *
 */
struct [[eosio::table]] staking_row {
    uint64_t id;
    extended_asset quantity;
    uint64_t primary_key() const { return id; }
    uint128_t by_token() const { return
xsat::utils::compute_id(quantity.get_extended_symbol()); }
};
typedef eosio::multi_index<"staking"_n, staking_row,
                           indexed_by<"bytoken"_n, const_mem_fun<staking_row, uint128_t,
&staking_row::by_token>>>
    staking_table;

/**
 * @brief  release table.
 * @scope `staker`
 *
 * @field id - primary key
 * @field quantity - the amount to release the token
 * @field expiration_time - unstake expiration time
 *
 */
struct [[eosio::table]] release_row {
    uint64_t id;
    extended_asset quantity;
    time_point_sec expiration_time;
    uint64_t primary_key() const { return id; }
    uint64_t by_expiration_time() const { return expiration_time.sec_since_epoch(); }
};
typedef eosio::multi_index<
    "releases"_n, release_row,
    eosio::indexed_by<"byexpire"_n, const_mem_fun<release_row, uint64_t,
&release_row::by_expiration_time>>>
    release_table;
```

rwddist.xsat

```
struct validator_info {
    name account;
    uint64_t staking;
};
/**
 * @brief reward log table.
 * @scope `get_self()`
 *
 * @field height - block height
 * @field hash - block hash
 * @field synchronizer_rewards - the synchronizer assigns the number of rewards
 * @field consensus_rewards - the consensus validator allocates the number of rewards
 * @field staking_rewards - the validator assigns the number of rewards
 * @field num_validators - the number of validators who pledge more than 100 BTC
 * @field provider_validators - validators who can receive rewards
 * @field endorsed_staking - total endorsed pledge amount
 * @field reached_consensus_staking - the total pledge amount to reach consensus is (number
of validators * 2/3 + 1
 *pledge amount)
* @field num_validators_assigned - the number of validators that have been allocated rewards
* @field synchronizer - synchronizer address
* @field miner - miner address
* @field parser - parse the address of the block
* @field tx_id - transaction id
* @field latest_exec_time - latest reward distribution time
*
**/
struct [[eosio::table]] reward_log_row {
    uint64_t height;
    checksum256 hash;
    asset synchronizer_rewards;
    asset consensus_rewards;
    asset staking_rewards;
    uint32_t num_validators;
    std::vector<validator_info> provider_validators;
    uint64_t endorsed_staking;
    uint64_t reached_consensus_staking;
    uint32_t num_validators_assigned;
    name synchronizer;
    name miner;
    name parser;
    checksum256 tx_id;
    time_point_sec latest_exec_time;
    uint64_t primary_key() const { return height; }
};
typedef eosio::multi_index<"rewardlogs"_n, reward_log_row> reward_log_table;
```

blkendt.xsat

```
struct validator_info {
    name account;
    uint64_t staking;
};

/**
 * @brief endorsement table.
 * @scope `height`
 *
 * @field id - primary id
 * @field hash - endorsement block hash
 * @field requested_validators - list of unendorsed validators
 * @field provider_validators - list of endorsed validators
 * @field endorsed_stake - endorsed pledge amount
 * @field reached_consensus_stake - reach consensus pledge amount
 *
 */
struct [[eosio::table]] endorsement_row {
    uint64_t id;
    checksum256 hash;
    std::vector<validator_info> requested_validators;
    std::vector<validator_info> provider_validators;
    uint64_t primary_key() const { return id; }
    checksum256 by_hash() const { return hash; }

    uint16_t num_validators() const { return requested_validators.size() +
provider_validators.size(); }

    uint64_t num_reached_consensus() const {
        return xsat::utils::num_reached_consensus(requested_validators.size() +
provider_validators.size());
    }

    bool reached_consensus() const {
        return provider_validators.size() > 0 && provider_validators.size() >=
num_reached_consensus();
    }
};
typedef eosio::multi_index<
    "endorsements"_n, endorsement_row,
    eosio::indexed_by<"byhash"_n, const_mem_fun<endorsement_row, checksum256,
&endorsement_row::by_hash>>>
    endorsement_table;
```

blksync.xsat

```
typedef uint8_t block_status;
static const block_status uploading = 1;
static const block_status upload_complete = 2;
static const block_status verify_merkle = 3;
static const block_status verify_parent_hash = 4;
static const block_status verify_fail = 5;
static const block_status verify_pass = 6;

/**
 * @brief global id table.
 * @scope `get_self()`
 *
 * @field bucket_id - `blockbuckets` Specifies the primary key of the table
 */
struct [[eosio::table]] global_id_row {
    uint32_t bucket_id;
};
typedef eosio::singleton<"globalid"_n, global_id_row> global_id_table;

/**
 * @brief block upload info struct.
 *
 * @field previous_block_hash - hash in internal byte order of the previous block’s header.
 * @field work - block workload
 * @field witness_reserve_value - witness reserve value in the block
 * @field witness_commitment -witness commitment in the block
 * @field has_witness - whether any of the transactions in the block contains witness
 * @field header_merkle - the merkle root of the block
 * @field relay_header_merkle - check header merkle relay data
 * @field relay_witness_merkle - check witness merkle relay data
 * @field num_transactions - the number of transactions in the block
 * @field processed_position - the location of the block that has been resolved
 * @field processed_transactions - the number of processed transactions
 *
 */
struct verify_info_data {
    checksum256 previous_block_hash;
    checksum256 work;
    std::optional<checksum256> witness_reserve_value;
    std::optional<checksum256> witness_commitment;
    bool has_witness;
    checksum256 header_merkle;
    std::vector<checksum256> relay_header_merkle;
    std::vector<checksum256> relay_witness_merkle;
    uint64_t num_transactions = 0;
    uint64_t processed_transactions = 0;
    uint64_t processed_position = 0;
};

/**
 * @brief block bucket table.
 * @scope `synchronizer`
 *
 * @field bucket_id - primary key, bucket_id is the scope associated with block.bucket
 * @field height - block height
 * @field size - block size
 * @field num_chunks - number of chunks
 * @field uploaded_size - the size of the uploaded data
 * @field uploaded_chunks - number of chunks that have been uploaded
 * @field endorsements - obtain the maximum number of endorsements
 * @field status - status of pending block (uploading, verify_pass, verify_fail,
endorse_parse, endorse_fail)
 * @field reason - cause of failure
 * @field cumulative_work - the cumulative workload of the block
 * @field miner - block miner address
 * @field verify_info - @see verify_info_data
 *
 */
struct [[eosio::table]] block_bucket_row {
    uint64_t bucket_id;
    uint64_t height;
    checksum256 hash;
    uint32_t size;
    uint32_t uploaded_size;
    uint8_t num_chunks;
    uint8_t uploaded_num_chunks;
    block_status status;
    std::string reason;

    checksum256 cumulative_work;
    name miner;
    vector<string> btc_miners;
    std::optional<verify_info_data> verify_info;

    bool in_verifiable() const {
        return status == upload_complete || status == verify_merkle || status ==
verify_parent_hash;
    }

    uint64_t primary_key() const { return bucket_id; }
    uint64_t by_status() const { return status; }
    checksum256 by_block_id() const { return xsat::utils::compute_block_id(height, hash); }
};
typedef eosio::multi_index<
    "blockbuckets"_n, block_bucket_row,
    eosio::indexed_by<"bystatus"_n, const_mem_fun<block_bucket_row, uint64_t,
&block_bucket_row::by_status>>,
    eosio::indexed_by<"byblockid"_n, const_mem_fun<block_bucket_row, checksum256,
&block_bucket_row::by_block_id>>>
    block_bucket_table;

/**
 * @brief passed index table.
 * @scope `height`
 *
 * @field id - primary key
 * @field hash - block hash
 * @field bucket_id - primary key of blockbuckets table
 * @field synchronizer - account for uploading block data
 *
 */
struct [[eosio::table]] passed_index_row {
    uint64_t id;
    checksum256 hash;
    uint64_t bucket_id;
    name synchronizer;
    uint64_t primary_key() const { return id; }
    uint64_t by_bucket_id() const { return bucket_id; }
    checksum256 by_hash() const { return hash; }
};

typedef eosio::multi_index<
    "passedindexs"_n, passed_index_row,
    eosio::indexed_by<"bybucketid"_n, const_mem_fun<passed_index_row, uint64_t,
&passed_index_row::by_bucket_id>>,
    eosio::indexed_by<"byhash"_n, const_mem_fun<passed_index_row, checksum256,
&passed_index_row::by_hash>>>
    passed_index_table;

/**
 * @brief block chunk table.
 * @scope `bucket_id`
 *
 * @field id - primary key
 * @field data - the block chunk for block
 *
 */
struct [[eosio::table]] block_chunk_row {
    uint64_t id;
    std::vector<char> data;
    uint64_t primary_key() const { return id; }
};
typedef eosio::multi_index<"block.chunk"_n, block_chunk_row> block_chunk_table;

struct verify_block_result {
    std::string status;
    checksum256 block_hash;
};
```

endrmng.xsat

```
/**
* @brief global id table.
* @scope `get_self()`
*
* @field staking_id - stake number
*/
struct [[eosio::table]] global_id_row {
    uint64_t staking_id;
};
typedef singleton<"globalid"_n, global_id_row> global_id_table;

/**
* @brief whitelist table.
* @scope `get_self()`
*
* @field account - whitelist account
*/
struct [[eosio::table]] whitelist_row {
    name account;
    uint64_t primary_key() const { return account.value; }
};
typedef eosio::multi_index<"whitelist"_n, whitelist_row> whitelist_table;

/**
* @brief evm proxy table.
* @scope `get_self()`
*
* @field id - primary id
* @field proxy - evm proxy account
```

```
*/
struct [[eosio::table]] evm_proxy_row {
    uint64_t id;
    checksum160 proxy;
    uint64_t primary_key() const { return id; }
    checksum256 by_proxy() const { return xsat::utils::compute_id(proxy); }
};
typedef eosio::multi_index<
    "evmproxys"_n, evm_proxy_row,
    eosio::indexed_by<"byproxy"_n, const_mem_fun<evm_proxy_row, checksum256,
&evm_proxy_row::by_proxy>>>
    evm_proxy_table;

/**
* @brief evm stake table.
* @scope `get_self()`
*
* @field id - primary key
* @field proxy - proxy address
* @field staker - staker address
* @field validator - validator address
* @field quantity - total number of staking
* @field stake_debt - amount of requested stake debt
* @field staking_reward_unclaimed - amount of stake unclaimed rewards
* @field staking_reward_claimed  - amount of stake claimed rewards
* @field consensus_debt - amount of requested consensus debt
* @field consensus_reward_unclaimed - amount of consensus unclaimed rewards
* @field consensus_reward_claimed  - amount of consensus claimed rewards
*/
struct [[eosio::table]] evm_staker_row {
    uint64_t id;
    checksum160 proxy;
    checksum160 staker;
    name validator;
    asset quantity;
    uint64_t stake_debt;
    asset staking_reward_unclaimed;
    asset staking_reward_claimed;
    uint64_t consensus_debt;
    asset consensus_reward_unclaimed;
    asset consensus_reward_claimed;
    uint64_t primary_key() const { return id; }
    uint64_t by_validator() const { return validator.value; }
    checksum256 by_staker() const { return xsat::utils::compute_id(staker); }
    checksum256 by_proxy() const { return xsat::utils::compute_id(proxy); }
    checksum256 by_staking_id() const { return compute_staking_id(proxy, staker, validator);
}
};
typedef eosio::multi_index<
    "evmstakers"_n, evm_staker_row,
    eosio::indexed_by<"byvalidator"_n, const_mem_fun<evm_staker_row, uint64_t,
&evm_staker_row::by_validator>>,
    eosio::indexed_by<"bystaker"_n, const_mem_fun<evm_staker_row, checksum256,
&evm_staker_row::by_staker>>,
    eosio::indexed_by<"bystakingid"_n, const_mem_fun<evm_staker_row, checksum256,
&evm_staker_row::by_staking_id>>,
    eosio::indexed_by<"byproxy"_n, const_mem_fun<evm_staker_row, checksum256,
&evm_staker_row::by_proxy>>>
    evm_staker_table;
```

```
/**
* @brief native stake table.
* @scope `get_self()`
*
* @field id - primary key
* @field staker - staker address
* @field validator - validator address
* @field quantity - total number of staking
* @field stake_debt - amount of requested stake debt
* @field staking_reward_unclaimed - amount of stake unclaimed rewards
* @field staking_reward_claimed  - amount of stake claimed rewards
* @field consensus_debt - amount of requested consensus debt
* @field consensus_reward_unclaimed - amount of consensus unclaimed rewards
* @field consensus_reward_claimed  - amount of consensus claimed rewards
*
*/
struct [[eosio::table]] native_staker_row {
    uint64_t id;
    name staker;
    name validator;
    asset quantity;
    uint64_t stake_debt;
    asset staking_reward_unclaimed;
    asset staking_reward_claimed;
    uint64_t consensus_debt;
    asset consensus_reward_unclaimed;
    asset consensus_reward_claimed;
    uint64_t primary_key() const { return id; }
    uint64_t by_validator() const { return validator.value; }
    uint64_t by_staker() const { return staker.value; }
    uint128_t by_staking_id() const { return compute_staking_id(staker, validator); }
};
typedef eosio::multi_index<
    "stakers"_n, native_staker_row,
    eosio::indexed_by<"byvalidator"_n,
                        const_mem_fun<native_staker_row, uint64_t,
&native_staker_row::by_validator>>,
    eosio::indexed_by<"bystaker"_n, const_mem_fun<native_staker_row, uint64_t,
&native_staker_row::by_staker>>,
    eosio::indexed_by<"bystakingid"_n,
                        const_mem_fun<native_staker_row, uint128_t,
&native_staker_row::by_staking_id>>>
    native_staker_table;

/**
 * @brief validator table.
 * @scope `get_self()`
 *
 * @field owner - primary key
 * @field reward_recipient - get rewards to address
 * @field memo - memo when receiving rewards
 * @field commission_rate - commission rate
 * @field quantity - total number of staking
 * @field stake_acc_per_share - staking rewards earnings per share
 * @field consensus_acc_per_share - consensus reward earnings per share
 * @field stake_debt - amount of requested stake debt
 * @field staking_reward_unclaimed - amount of stake unclaimed rewards
 * @field staking_reward_claimed  - amount of stake claimed rewards
 * @field consensus_debt - amount of requested consensus debt
 * @field consensus_reward_unclaimed - amount of consensus unclaimed rewards
 * @field consensus_reward_claimed  - amount of consensus claimed rewards
 * @field latest_staking_time - latest staking or unstaking time
 * @field latest_reward_block - latest reward block
 * @field latest_reward_time - latest reward time
 * @field disabled_staking - whether to disable staking
 *
 */
struct [[eosio::table]] validator_row {
    name owner;
    name reward_recipient;
    string memo;
    uint64_t commission_rate;
    asset quantity;
    uint128_t stake_acc_per_share;
    uint128_t consensus_acc_per_share;
    asset staking_reward_unclaimed;
    asset staking_reward_claimed;
    asset consensus_reward_unclaimed;
    asset consensus_reward_claimed;
    asset total_consensus_reward;
    asset consensus_reward_balance;
    asset total_staking_reward;
    asset staking_reward_balance;
    time_point_sec latest_staking_time;
    uint64_t latest_reward_block;
    time_point_sec latest_reward_time;
    bool disabled_staking;
    uint64_t primary_key() const { return owner.value; }
    uint64_t by_total_staking() const { return quantity.amount; }
    uint64_t by_disabled_staking() const { return disabled_staking; }
};
typedef eosio::multi_index<
    "validators"_n, validator_row,
    eosio::indexed_by<"bystate"_n, const_mem_fun<validator_row, uint64_t,
&validator_row::by_disabled_staking>>,
    eosio::indexed_by<"bystaked"_n, const_mem_fun<validator_row, uint64_t,
&validator_row::by_total_staking>>>
    validator_table;
```

utxomng.xsat

```
typedef uint8_t parsing_status;
static const parsing_status waiting = 1;
static const parsing_status parsing = 2;
static const parsing_status deleting_data = 3;
static const parsing_status distributing_rewards = 4;

/**
 * @brief chain state table.
 * @scope `get_self()`
 *
 * @field head_height - header block height for consensus success
 * @field irreversible_height -irreversible block height
 * @field irreversible_hash - irreversible block hash
 * @field num_utxos - total number of UTXOs
 * @field status - parsing status @see parsing_status
 * @field parsing_hash - parsing block hash
 * @field num_transactions - the number of transactions in the block that was resolved, the
value of which was not
 * parsed for the first time was 0
 * @field processed_position - the location of the block that has been resolved
 * @field processed_transactions - the number of resolved transactions
 * @field processed_vin - parse to the index of vin
 * @field processed_vout - parse to the index of vout
 * @field parser - current block resolution account
 * @field parsed_expiration_time - the timeout of the next block resolution, and the reward
obtained by other
 * synchronizers can be resolved
 * @field miner - block miner address
 * @field synchronizer - block synchronizer address
 * @field num_provider_validators - the number of validators endorsed by the current block
 * @field num_validators_assigned - the number of validators that have been allocated
rewards
 *
 */
struct [[eosio::table("chainstate")]] chain_state_row {
    uint64_t head_height;
    uint64_t irreversible_height;
    checksum256 irreversible_hash;
    uint64_t num_utxos;
    parsing_status status;
    uint64_t parsing_height;
    checksum256 parsing_hash;
    uint64_t parsing_bucket_id;
    uint64_t num_transactions;
    uint64_t processed_transactions;
    uint64_t processed_position;
    uint64_t processed_vin;
    uint64_t processed_vout;
    uint32_t num_provider_validators;
    uint32_t num_validators_assigned;
    name miner;
    name synchronizer;
    name parser;
    eosio::time_point_sec parsed_expiration_time;
};
typedef eosio::singleton<"chainstate"_n, chain_state_row> chain_state_table;

/**
 * @brief chain state table.
 * @scope `get_self()`
 *
 * @field parse_timeout_seconds - parsing timeout duration
 * @field num_validators_per_distribution - number of endorsing users each time rewards are
distributed
 * @field num_retain_data_blocks - number of blocks to retain data
 * @field num_txs_per_verification - the number of tx for each verification (2^n)
 * @field num_merkle_layer - verify the number of merkle levels
(log(num_txs_per_verification))
 *
 * */
struct [[eosio::table("config")]] config_row {
    uint16_t parse_timeout_seconds = 600;
    uint16_t num_validators_per_distribution = 100;
    uint16_t num_retain_data_blocks = 100;
    uint16_t num_txs_per_verification = 2048;
    uint8_t num_merkle_layer = 11;
    uint16_t num_miner_priority_blocks = 10;
};
typedef eosio::singleton<"config"_n, config_row> config_table;

/**
 * @brief utxo table.
 * @scope `get_self()`
 *
 * @field id - the height of the block in resolution
 * @field txid - transaction id
 * @field index - vout index
 * @field scriptpubkey - script public key
 * @field value - utxo quantity
 *
 */
struct [[eosio::table]] utxo_row {
    uint64_t id;
    checksum256 txid;
    uint32_t index;
    std::vector<uint8_t> scriptpubkey;
    uint64_t value;
    uint64_t primary_key() const { return id; }
    checksum256 by_scriptpubkey() const { return xsat::utils::hash(scriptpubkey); }
    checksum256 by_utxo_id() const { return compute_utxo_id(txid, index); }
};
typedef eosio::multi_index<
    "utxos"_n, utxo_row,
    eosio::indexed_by<"scriptpubkey"_n, const_mem_fun<utxo_row, checksum256,
&utxo_row::by_scriptpubkey>>,
    eosio::indexed_by<"byutxoid"_n, const_mem_fun<utxo_row, checksum256,
&utxo_row::by_utxo_id>>>
    utxo_table;

/**
 * @brief block table.
 * @scope `get_self()`
 *
 * @field height - the height of the block
 * @field hash - the hash of the block
 * @field cumulative_work - the cumulative workload of the block
 * @field version - block version
 * @field previous_block_hash - hash in internal byte order of the previous block’s header.
 * @field merkle - the merkle root is derived from the hashes of all transactions included
in this block
 * @field timestamp - the block time is a Unix epoch time
 * @field bits - An encoded version of the target threshold this block’s header hash must be
less than or equal
 * to
 * @field nonce -An arbitrary number miners change to modify the header hash in order to
produce a hash less than or
 */
struct [[eosio::table]] block_row {
    uint64_t height;
    checksum256 hash;
    checksum256 cumulative_work;
    uint32_t version;
    checksum256 previous_block_hash;
    checksum256 merkle;
    uint32_t timestamp;
    uint32_t bits;
    uint32_t nonce;
    uint64_t primary_key() const { return height; }
    checksum256 by_hash() const { return hash; }
};
typedef eosio::multi_index<
    "blocks"_n, block_row,
    eosio::indexed_by<"byhash"_n, const_mem_fun<block_row, checksum256,
&block_row::by_hash>>>
    block_table;

/**
 * @brief block extra table.
 * @scope `height`
 *
 * @field bucket_id - the hash of the block
 */
struct [[eosio::table]] block_extra_row {
    uint16_t bucket_id;
};
typedef eosio::singleton<"block.extra"_n, block_extra_row> block_extra_table;

/**
 * @brief consensus success block table.
 * @scope `get_self()`
 *
 * @field bucket_id - primary key of blockbuckets table
 * @field height - the height of the block
 * @field hash - the hash of the block
 * @field cumulative_work - the cumulative workload of the block
 * @field version - block version
 * @field previous_block_hash - hash in internal byte order of the previous block’s header.
 * @field merkle - the merkle root is derived from the hashes of all transactions included
in this block
 * @field timestamp - the block time is a Unix epoch time
 * @field bits - An encoded version of the target threshold this block’s header hash must be
less than or equal
 * to
 * @field nonce -An arbitrary number miners change to modify the header hash in order to
produce a hash less than or
 * @field miner - block miner address
 * @field synchronizer - block synchronizer address
 * @field num_provider_validators - the number of users who have endorsed the current block
 */
struct [[eosio::table]] consensus_block_row {
    uint64_t bucket_id;
    uint64_t height;
    checksum256 hash;
    checksum256 cumulative_work;
    uint32_t version;
    checksum256 previous_block_hash;
    checksum256 merkle;
    uint32_t timestamp;
    uint32_t bits;
    uint32_t nonce;
    name miner;
    name synchronizer;
    uint32_t num_provider_validators;
    uint64_t primary_key() const { return bucket_id; }
    uint64_t by_height() const { return height; }
    checksum256 by_block_id() const { return xsat::utils::compute_block_id(height, hash); }
    uint64_t by_synchronizer() const { return synchronizer.value; }
};
typedef eosio::multi_index<
    "consensusblk"_n, consensus_block_row,
    eosio::indexed_by<"byheight"_n, const_mem_fun<consensus_block_row, uint64_t,
&consensus_block_row::by_height>>,
    eosio::indexed_by<"byblockid"_n,
                        const_mem_fun<consensus_block_row, checksum256,
&consensus_block_row::by_block_id>>,
    eosio::indexed_by<"bysyncer"_n,
                        const_mem_fun<consensus_block_row, uint64_t,
&consensus_block_row::by_synchronizer>>>
    consensus_block_table;

struct process_block_result {
    uint64_t height;
    checksum256 block_hash;
    string status;
};
```