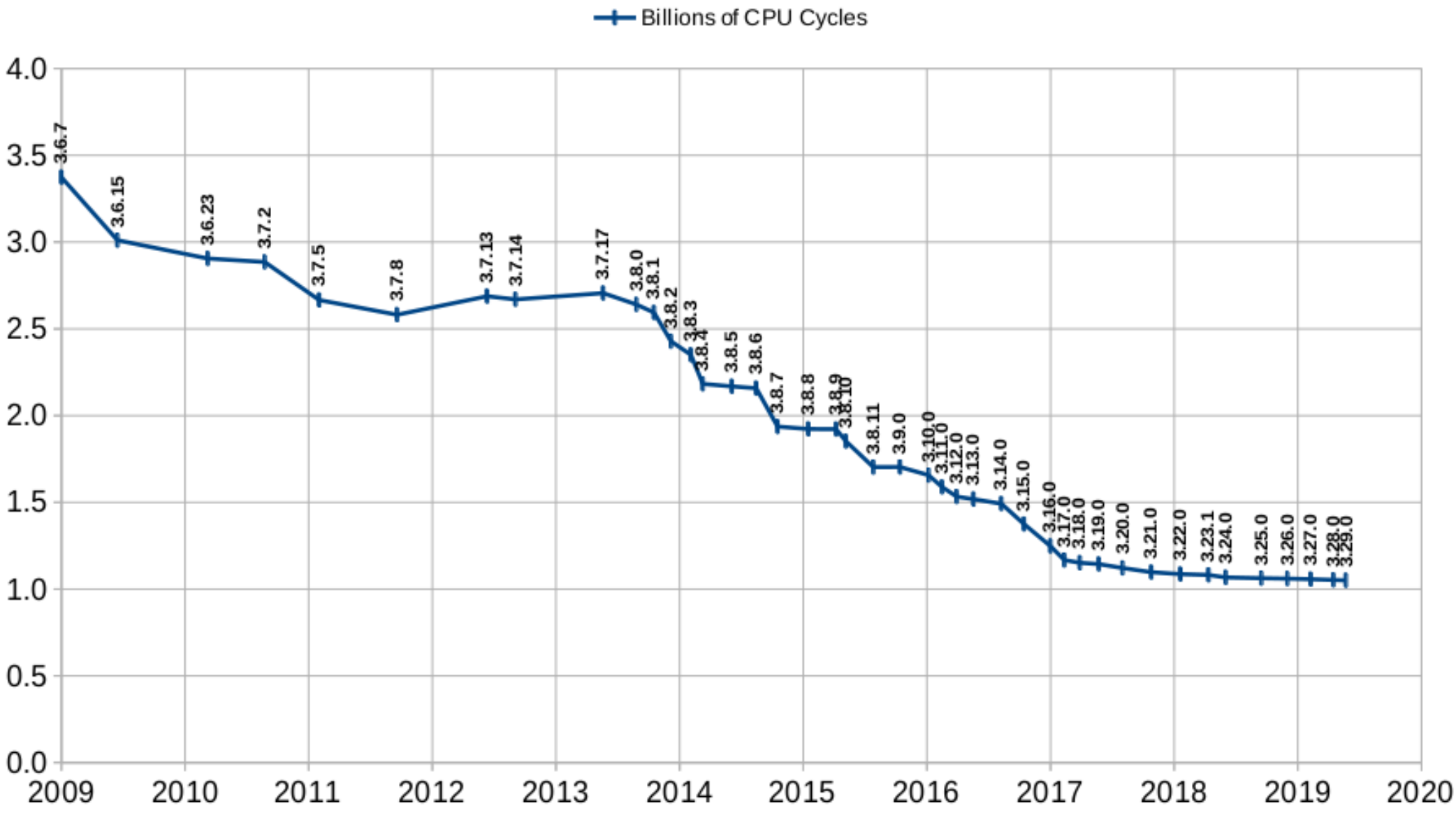
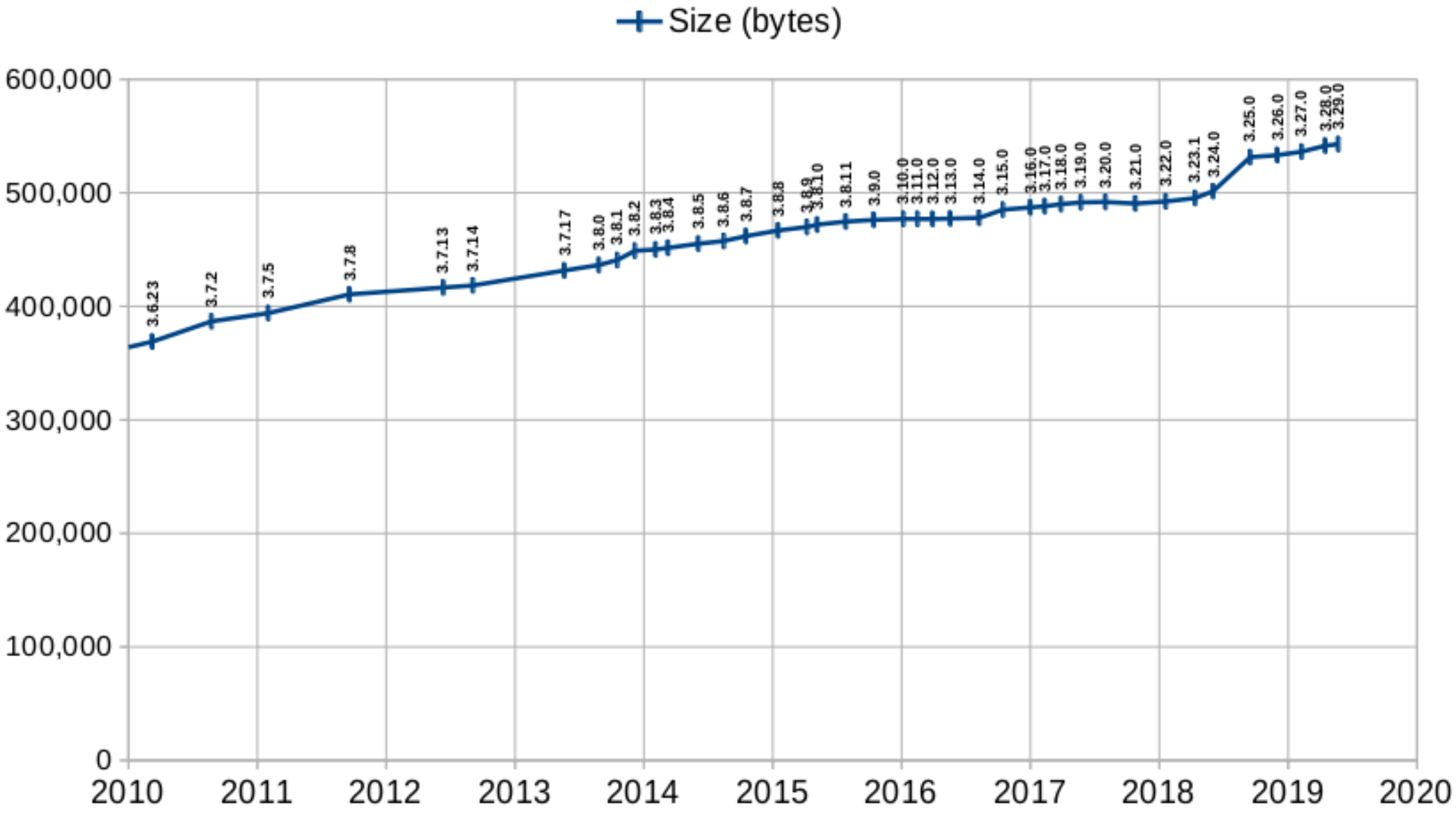
SQLite CPU Cycles



SQLite Size


Size (bytes)
600,000
500,000
400,000
300,000
200,000
100,000
0
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
3.6.23
3.7.2
3.7.5
3.7.8
3.7.13
3.7.14
3.7.17
3.8.0
3.8.1
3.8.2
3.8.3
3.8.4
3.8.5
3.8.6
3.8.7
3.8.8
3.8.9
3.8.10
3.8.11
3.9.0
3.10.0
3.11.0
3.12.0
3.13.0
3.14.0
3.15.0
3.16.0
3.17.0
3.18.0
3.19.0
3.20.0
3.21.0
3.22.0
3.23.1
3.24.0
3.25.0
3.26.0
3.27.0
3.28.0
3.29.0

SQLite Changes 2018-2019

- The reuse-schema extension
- Enhancements to Window Functions
  - The EXCLUDE clause
  - Window chaining
  - GROUPS frames
  - <expr> PRECEDING and <expr> FOLLOWING in RANGE frames
- The VACUUM INTO command

SQLite Changes 2018-2019

- Official GitHub mirror at https://github.com/sqlite/sqlite
  - Fossil enhanced to automate Git mirroring
- Added the fossildelta.c extension
- The sqlite_dbdata virtual table
- Issue SQLITE_WARNING if a double-quoted string literal is used

SQLite Changes 2018-2019

- New C-language APIs
  - sqlite3_stmt_isexplain(STMT)
  - sqlite3_value_frombind(VALUE)
  - sqlite3_normalized_sql(STMT)
  - SQLITE_DBCONFIG_DEFENSIVE
  - SQLITE_DBCONFIG_WRITABLE_SCHEMA

SQLite Changes 2018-2019

- Continuous testing with the dbsqlfuzz fuzzer
- Added SQLITE_DBCONFIG_DEFENSIVE
  - Prevents hostile SQL from corrupting the database
  - Shadow tables in FTS3, FTS5, and RTREE become read-only
  - Disables “PRAGMA writable_schema=ON”
  - The sqlite_dbpage virtual table becomes read-only
  - “PRAGMA journal_mode=OFF” is disallowed
- Many security enhancements

SQLite Changes 2018-2019

- CLI enhancements
  - Improved “.help”
  - Added “.recover” (uses the sqlite_dbdata vtab)
  - Added “.eqp trace”
  - Bound parameters using the “.parameter” command
  - The --memtrace command-line option
  - The --deserialize option uses sqlite3_deserialize()

SQLite Changes 2018-2019

- CLI enhancements (continued)
  - Added “.open --hexdb”
  - The --update option on “.archive” skips files that are already in the archive
  - SQLITE_HISTORY environment variable
  - Added the --async option to the .backup command
  - New options for “.trace”: --expanded --normalized --plain --profile --row --stmt --close
  - Added “--maxsize N” on “.open --deserialize”

SQLite Changes 2018-2019

- New Optimizations
  - Omit AND and OR operators if one operand is constant
  - Avoid unnecessary tests on queries driven by a partial index
  - The LIKE optimization works when the ESCAPE keyword is present and “PRAGMA case_sensitive_like” is on
  - Avoid changing an index on expressions if the expressions do not reference any column being changed by an UPDATE

Last year's changes...

SQLite Changes 2018

- Window functions
- Improvements to ALTER TABLE
  - RENAME COLUMN
  - Updates table names inside TRIGGERs and VIEWs
- Geopoly extension to R-Tree
- Upsert
- Auxiliary columns in R-Tree
- Improvements to EXPLAIN QUERY PLAN

SQLite Changes 2017-2018

- C-APIs to access SQLite Keywords.
- The sqlite3_str object
- The sqlite3_serialize() and sqlite3_deserialize() APIs
- Keywords TRUE and FALSE
- rtreecheck()
- Add the ability to read WAL-mode databases even if the application lacks write permission on the database file or the -shm and -wal files.

SQLite Changes 2017-2018

- Command-line tool enhancements
  - Read/write ZIP archives like a database
  - Access SQL archives
  - Access via AppendVFS
  - Tab-completion
  - The ".excel" command
- ATTACH and DETACH work inside of a transaction
- The sqlite_stmt virtual table
- date/time functions now usable within indexes on expressions and partial indexes if they are "deterministic"

Allows an SQLite database to be appended to some other file, such as an EXE.

SQLite Changes 2017-2018

- Support for the F2FS filesystem
- Union and Swarm virtual tables
- The ZIPFILE virtual table
- The sqlite3_analyzer.exe utility program now shows the number of bytes of metadata on btree pages.
- PRAGMA secure_delete=FAST
- Add the pointer-passing APIs and update FTS5, CARRAY, and REMEMBER extensions to use them.

SQLite Changes 2017-2018

- The sqlite_dbpage virtual table
- The sqlite_memstat virtual table
- The sqlite_offset() SQL function
- The COMPLETION virtual table
- The sqlite3_prepare_v3() interface and the SQLITE_PREPARE_PERSIST flag.
- The SQLITE_DBCONFIG_ENABLE_QPSG flag and the SQLITE_ENABLE_QPSG compile option.
- Countless query planner improvements and microoptimizations

The remaining slides show further details about various enhancements.

reuse-schema

- Compile with -DSQLITE_ENABLE_SHARED_SCHEMA
  - Do not confuse “SHARED_CACHE” with “SHARED_SCHEMA”
- Add SQLITE_OPEN_SHARED_SCHEMA to sqlite3_open_v2()
- Memory space reused only if:
  - sqlite_master tables are identical
  - Same “PRAGMA schema_version” value

reuse-schema

- Currently maintained on a separate branch of the SQLite core
  - https://www.sqlite.org/src/timeline?r=reuse-schema
  - Other SQLite Consortium member branches include:
    - apple-osx
    - begin-concurrent-pnu
    - sessions (now merged to trunk)
    - branch-3.28
- Should reuse-schema become the default branch used to build the NDS dev-kit?

reuse-schema CLI enhancements

- ".shared-schema check DB1 DB2 ..."
  - Check to see if the named database files can take advantage of the shared-schema optimization.
- ".shared-schema fix DB1 DB2 ..."
  - Try to patch up the schemas and schema_version numbers on the named databases so that they are usable by the optimization

Go Back....

The fossildelta.c extension

- delta_create(X,Y)
  - Return a delta that converts X into Y.
- delta_apply(X,D)
  - Apply delta D to input X. Return the result Y.
- delta_output_size(D)
  - Return the size of the output file for delta D
- delta_parse(D)
  - Table-valued function - return rows that describe the content of delta D

Go Back...

sqlite_dbdata

```
CREATE TABLE t1(a,b,c,d,e);
INSERT INTO t1
  VALUES(1,2.5,'three',x'4444',null);
CREATE INDEX t1bc ON t1(b,c);
SELECT * FROM sqlite_dbdata;
```

'pgno'	'cell'	'field'	'value'	
1	0	-1	1	sqlite_master
1	0	0	'table'	sqlite_master
1	0	1	't1'	sqlite_master
1	0	2	't1'	sqlite_master
1	0	3	2	sqlite_master
1	0	4	'CREATE TABLE t1(a,b,c,d,e)'	sqlite_master
1	1	-1	2	sqlite_master
1	1	0	'index'	sqlite_master
1	1	1	't1bc'	sqlite_master
1	1	2	't1'	sqlite_master
1	1	3	3	sqlite_master
1	1	4	'CREATE INDEX t1bc ON t1(b,c)'	sqlite_master
2	0	-1	1	t1
2	0	0	1	t1
2	0	1	2.5	t1
2	0	2	'three'	t1
2	0	3	X'4444'	t1
3	0	0	2.5	t1bc
3	0	1	'three'	t1bc
3	0	2	1	t1bc

Go Back...

UPSERT

```
CREATE TABLE names(
  name TEXT PRIMARY KEY,
  count INT DEFAULT 1
);
...
INSERT INTO names(name) VALUES('alice')
  ON CONFLICT(name) UPDATE SET count=count+1;
```

Try to insert the name 'alice'.

- *If 'alice' was not previously in the table, add it with a “count” of 1.*
- *If 'alice' was already in the table, increment its count.*

Go Back....

RENAME COLUMN

Limited ALTER TABLE capability:

Add a new column to a table.

Rename the table

ALTER TABLE *table* ADD COLUMN ...;
ALTER TABLE *table* RENAME TO *newname*;
New! → ALTER TABLE *table* RENAME COLUMN *oldname* TO *newname*;

RENAME COLUMN Example

```
CREATE TABLE xyz(aa INT, bb TEXT, cc FLOAT);
CREATE INDEX xyz_x1 ON xyz(aa, cc);
CREATE VIEW uvw AS SELECT aa+bb FROM xyz;
CREATE TRIGGER r1 AFTER INSERT ON xyz BEGIN
   INSERT INTO log VALUES(NEW.aa,NEW.cc);
END;
```

```
ALTER TABLE xyz RENAME aa TO dd;
```

```
CREATE TABLE xyz(dd INT, bb TEXT, cc FLOAT);
CREATE INDEX xyz_x1 ON xyz(dd, cc);
CREATE VIEW uvw AS SELECT dd+bb FROM xyz;
CREATE TRIGGER r1 AFTER INSERT ON xyz BEGIN
   INSERT INTO log VALUES(NEW.dd,NEW.cc);
END;
```

Go Back....

Enhanced ALTER TABLE

```
CREATE TABLE xyz(aa INT, bb TEXT, cc FLOAT);
CREATE INDEX xyz_x1 ON xyz(aa, cc) WHERE xyz.aa NOT NULL;
CREATE VIEW uvw AS SELECT aa+bb FROM xyz;
CREATE TRIGGER r1 AFTER INSERT ON other BEGIN
   INSERT INTO xyz VALUES(1,2,3);
END;
```

```
ALTER TABLE xyz RENAME TO mmm;
```

```
CREATE TABLE mmm(aa INT, bb TEXT, cc FLOAT);
CREATE INDEX xyz_x1 ON mmm(aa, cc) WHERE mmm.aa NOT NULL;
CREATE VIEW uvw AS SELECT aa+bb FROM mmm;
CREATE TRIGGER r1 AFTER INSERT ON other BEGIN
   INSERT INTO mmm VALUES(1,2,3);
END;
```

All versions

Version 3.25 and later

Go Back....

Ascii-Art EXPLAIN QUERY PLAN

EXPLAIN QUERY PLAN
WITH cnt(i) AS (SELECT 1 UNION ALL SELECT i+1 FROM cnt LIMIT 1)
SELECT * FROM cnt, y1 WHERE i=a

Version 3.23 and before:

```
3|0|0|SCAN TABLE cnt
1|0|0|COMPOUND SUBQUERIES 0 AND 0 (UNION ALL)
0|0|0|SCAN SUBQUERY 1
0|1|1|SEARCH TABLE y1 USING INDEX y1a (a=?)
```

Version 3.24 and after:

```
QUERY PLAN
|--MATERIALIZE 2
|   |--SETUP
|   |   `--SCAN CONSTANT ROW
|   `--RECURSIVE STEP
|       `--SCAN TABLE cnt
|--SCAN SUBQUERY 2
`--SEARCH TABLE y1 USING INDEX y1a (a=?)
```

Go Back....

SQLite Keyword APIs

```
int sqlite3_keyword_count(void);
```

Return the number of SQLite keywords.

```
int sqlite3_keyword_name(
  int i
  const char **pzKeyword,
  int *pnKeyword
);
```

Which keyword to fetch

*Make *pzKeyword point to the start of the keyword. NOT zero-terminated!*

Length of the keyword in bytes

```
int sqlite3_keyword_count(
  const char *zWordToCheck,
  int nWord
);
```

Is this a keyword?

Length of zWordToCheck in bytes

Go Back....

sqlite3_str

Constructor for sqlite3_str

```
sqlite3_str *pStr = sqlite3_str_new(db);
```

Database pointer may be NULL

```
sqlite3_str_appendf(pStr, “Hello, %s!”, “World”);
```

printf()-style format string

```
char *z = sqlite3_str_finish(pStr);
```

Destructor for the sqlite3_str

Pass z to sqlite3_free() when done

sqlite3_str

```
sqlite3_str *sqlite3_str_new(sqlite3*);
```

Constructor

```
void sqlite3_str_appendf(sqlite3_str*, const char *zFormat, ...);
void sqlite3_str_vappendf(sqlite3_str*, const char *zFormat, va_list);
void sqlite3_str_append(sqlite3_str*, const char *zIn, int N);
void sqlite3_str_appendall(sqlite3_str*, const char *zIn);
void sqlite3_str_appendchar(sqlite3_str*, int N, char C);
void sqlite3_str_reset(sqlite3_str*);
int sqlite3_str_errcode(sqlite3_str*);
int sqlite3_str_length(sqlite3_str*);
char *sqlite3_str_value(sqlite3_str*);
```

Methods

```
char *sqlite3_str_finish(sqlite3_str*);
```

Destructor

Go Back....

serialize/deserialize

- Use an SQLite database file as a wire-transfer protocol
  - Robustly transfer text and/or binary data
  - Self-describing file format
  - Cross-platform: 32/64-bit, UTF8/16, big/little-endian
  - Easily extensible
- sqlite3_serialize() converts a database into a BLOB in memory → Capture and send over the wire.
- sqlite3_deserialize() takes a BLOB in memory and uses it as a database.

Go Back....

TRUE and FALSE keywords

- TRUE → An alias for 1
- FALSE → An alias for 0
- “<expr> IS TRUE” and “<expr> IS FALSE” are handled specially and do the right thing

Go Back....

ZIP archive as database

```
sqlite3 xyzzy.zip
```

Open the command-line shell on the file "xyzzy.zip" as if that file where a database with a single table "zip".

```
CREATE TABLE zip(
  name TEXT PRIMARY KEY,
  mode INT,
  mtime INT,
  sz INT,
  rawdata BLOB,
  data BLOB,
  method INT
);
```

One row in the table for each file in the ZIP archive

ZIP archive as database examples

Show the names of all files in xyzzy.zip that contain the string ‘hello’:

sqlite3 xyzzy.zip “SELECT name FROM zip WHERE data GLOB ‘*hello*’ ”

Show the names and (uncompressed) sizes of the five largest *.txt files in the xyzzy.zip file:

sqlite3 xyzzy.zip “SELECT name, sz FROM zip WHERE name GLOB ‘*.txt’ ORDER BY sz DESC LIMIT 5 ”

Go Back...

SQL Archive Files

- Like a ZIP archive or tarball, but based on SQLite
- Space efficient, especially for small objects
- An SQLite database with this schema:

```
CREATE TABLE sqlar(
  name TEXT PRIMARY KEY,  -- name of the file
  mode INT,               -- access permissions
  mtime INT,              -- last modification time
  sz INT,                 -- original file size
  data BLOB               -- compressed content
);
```

SQL Archive Files

- Flexible. ZIP archive stores only files. SQL Archive stores any relational data
- Transactional
- Incremental updates
- High-level query language
- Bypass email filters

SQL Archive Files

sqlite3 attachment.db -Ac prototype.exe

Construct a new SQL archive named “attachment.db” containing the file “prototype.exe”.

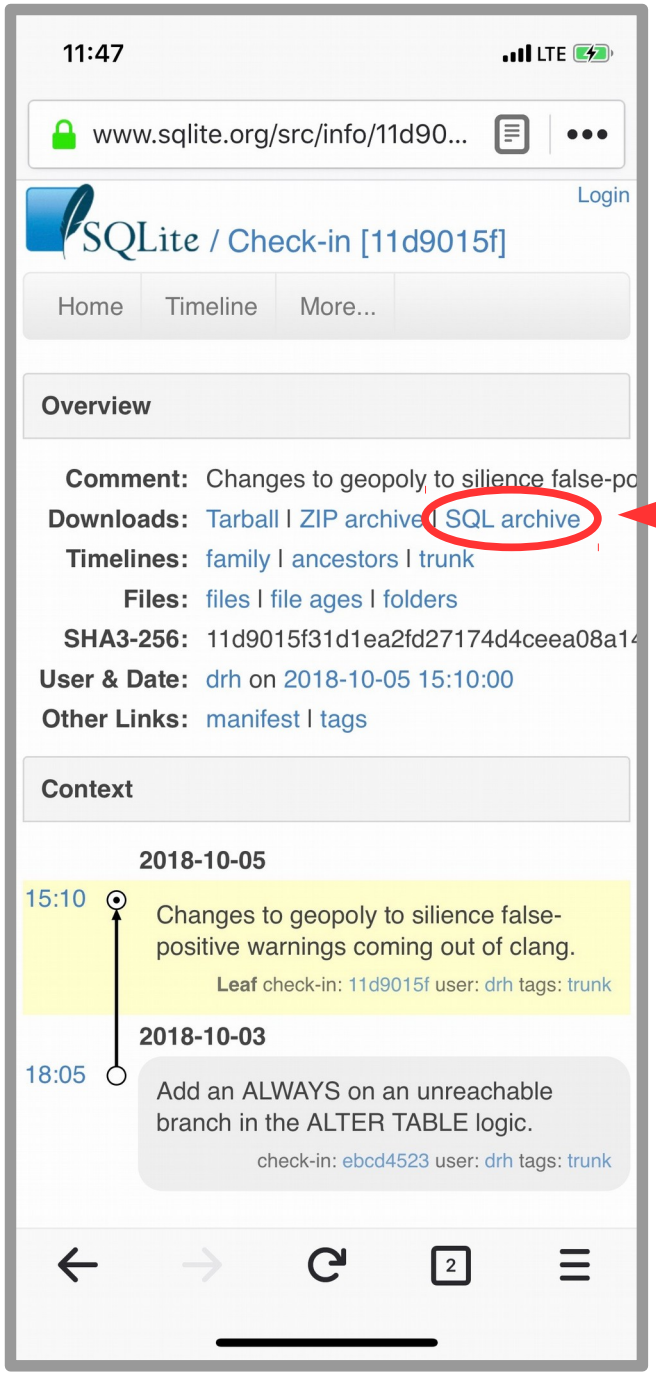
Send the SQL archive via email attachment to a colleague.

sqlite3 attachment.db -Ax

Extract all files from the archive

sqlite3 -A

Summary of SQL archive commands



Download SQLite source code as an SQL Archive

Go Back...

View Query Results In Spreadsheet

```
sqlite3 database.db
sqlite> .excel
sqlite> SELECT ...;
```

Bring up the SQLite CLI on the database

The ".excel" command means to send the output of the next SQL query to a spreadsheet rather than to the screen.

The result of this query will be loaded into the spreadsheet.

View Query Results In Spreadsheet

Populate a spreadsheet with the names and sizes of all files in xyzzy.zip:

```
sqlite3 xyzzy.zip
sqlite> .excel
sqlite> SELECT name, sz FROM zip ORDER BY 1;
```

Go Back...

sqlite_stmt

```
CREATE TABLE sqlite_stmt(
  sql TEXT,
  ncol INT,
  ro BOOLEAN,
  busy BOOLEAN,
  nScan INT,
  nSort INT,
  naidx INT,
  nStep INT,
  reprep INT,
  run INT,
  mem INT
);
```

- One row for each active prepared statement
- Columns show status and performance measurements

sqlite_stmt

```
CREATE TABLE sqlite_stmt(
  sql TEXT,
  ncol INT,
  ro BOOLEAN,
  busy BOOLEAN,
  nScan INT,
  nSort INT,
  naidx INT,
  nStep INT,
  reprep INT,
  run INT,
  mem INT
);
```

SQL for this statement

Number of columns in the result

Read-only flag

True if statement is currently running

Number of full-scan steps

Number of sort operations

Number of rows added to automatic indexes

Number of byte-codes executed

Number of re-prepare operations

Number of times this statement has been run

Memory used by this one prepared stmt

sqlite_stmt Use Cases

```
CREATE TABLE sqlite_stmt(
  sql TEXT,
  ncol INT,
  ro BOOLEAN,
  busy BOOLEAN,
  nScan INT,
  nSort INT,
  naidx INT,
  nStep INT,
  reprep INT,
  run INT,
  mem INT
);
```

- Check for unfinalized statements
- Check for statements left pending (not reset)
- Find statements using excess memory
- Find statements that could benefit from new indexes in the schema

Go Back...

```
*** Indices of table TAGXREF ****************************************

Percentage of total database.....................   1.4%
Number of entries.................................. 96150
Bytes of storage consumed.......................... 1605632
Bytes of payload................................... 1227689     76.5%
Bytes of metadata.................................. 290794      18.1%
Average payload per entry.......................... 12.77
Average unused bytes per entry..................... 0.91
Average metadata per entry......................... 3.02
Average fanout..................................... 98.00
Maximum payload per entry.......................... 17
Entries that use overflow.......................... 0            0.0%
Index pages used................................... 2
Primary pages used................................. 194
Overflow pages used................................ 0
Total pages used................................... 196
Unused bytes on index pages........................ 12457       76.0%
Unused bytes on primary pages...................... 74692        4.7%
Unused bytes on overflow pages..................... 0
Unused bytes on all pages.......................... 87149        5.4%
```

Go Back...

sqlite_dbpage

```
CREATE TABLE sqlite_dbpage(
  pgno INTEGER PRIMARY KEY,
  data BLOB
);
```

- One row for each page in the database file
- Read/write raw pages
- Writing to this table can easily corrupt the database file!

Go Back...

The COMPLETION table

```
SELECT candidate
    FROM completion('sliCri','SELECT sliCri');
```

- Suggest completions for partially entered words in an SQLite statement
- Arguments are optional
  - 1st: Characters of the last word entered so far
  - 2nd: The whole line seen so far

Go Back...

sqlite3_prepare_v3()

```
int sqlite3_prepare_v2(
  sqlite3 *db,
  const char *zSql,
  int nByte,

  sqlite3_stmt **ppStmt,
  const char **pzTail
);
```

```
int sqlite3_prepare_v3(
  sqlite3 *db,
  const char *zSql,
  int nByte,
  unsigned int prepFlags,
  sqlite3_stmt **ppStmt,
  const char **pzTail
);
```

SQLITE_PREPARE_PERSISTENT:
The prepared statement is expected to persist in memory for a long time.

Go Back...

Query Planner Stability Guarantee

- SQLite will always pick the same query plan for any given SQL statement as long as:
  - the database schema does not change in significant ways such as adding or dropping indices,
  - the ANALYZE command is not rerun,
  - the same version of SQLite is used.
- Important for safety-critical systems
- Off by default. Enable at compile-time or run-time

Go Back...

PRAGMA secure_delete

- PRAGMA secure_delete=OFF; ← *Default*
  - Disk space holding deleted content is reusable.
- PRAGMA secure_delete=ON; ← *Available since 2010*
  - All deleted content is overwritten with zeros.
- PRAGMA secure_delete=FAST; ← *New!*
  - Deleted content is overwritten as long as this does not increase the amount of disk I/O. Cells on btree pages are overwritten, but freelist pages are not.

Go Back...

ZIPFILE

```
CREATE VIRTUAL TABLE xyz USING zipfile('/name/of/file.zip');
```

Name of ZIP archive on disk

```
CREATE TABLE xyz(
  name TEXT PRIMARY KEY,
  mode INT,
  mtime INT,
  sz INT,
  rawdata BLOB,
  data BLOB,
  method INT
);
```

One row for each file in the ZIP archive

Go Back...

Pointer Passing APIs

- sqlite3_result_pointer()
  - Pointer as a result from an application-defined function or virtual table
- sqlite3_bind_pointer()
  - Application binds a pointer to a parameter
- sqlite3_value_pointer()
  - Extract a pointer set by result_pointer() or bind_pointer()

Formerly accomplished by encoding the pointer as an integer. Using the pointer-APIs is more secure, as it prevents an attacker from forging pointers.

Go Back...

Window Functions

- row_number()
- rank()
- dense_rank()
- percent_rank()
- cume_dist()
- ntile(N)
- lag(X,O,D)
- lead(X,O,D)
- first_value(X)
- last_value(X)
- nth_value(X,N)
- *plus all existing aggregate functions...*

Every window function must be followed by an OVER clause

Window Functions

```
SELECT
  row_number() OVER (),
  login
FROM
  user;
```

```
1,'NDSeV'
2,'alibaba-inc.com'
3,'anonymous'
4,'dan'
5,'dan2'
6,'developer'
7,'drh'
8,'fklebert'
9,'garmin.com'
10,'hscharmann'
11,'mistachkin'
12,'mrozloznik'
13,'ndsev'
14,'nobody'
15,'reader'
16,'test-user'
```

Window Functions

```
SELECT
  row_number() OVER
    (PARTITION BY substr(login,1,1)),
  login
FROM
  user;
```

```
1,'NDSeV'
1,'alibaba-inc.com'
2,'anonymous'
1,'dan'
2,'dan2'
3,'developer'
4,'drh'
1,'fklebert'
1,'garmin.com'
1,'hscharmann'
1,'mistachkin'
2,'mrozloznik'
1,'ndsev'
2,'nobody'
1,'reader'
1,'test-user'
```

Window Functions

Show all employees by name, together with their salary and the amount by which their salary is above or below the average salary of everyone in their department.

```
CREATE TABLE employee(
  name TEXT,
  dept TEXT,
  salary REAL
);
```

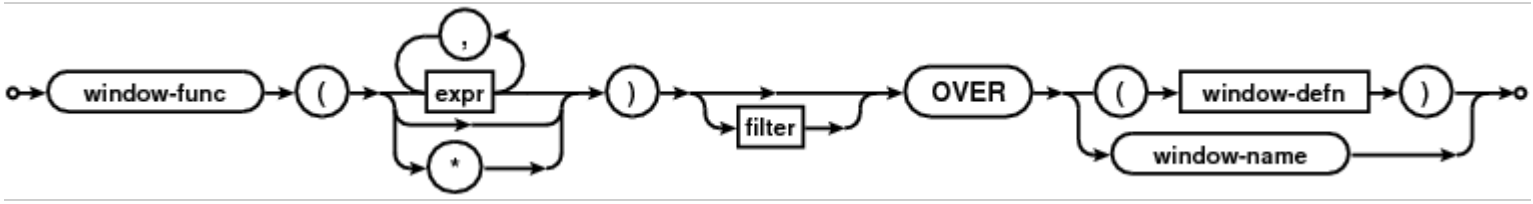
```
SELECT
  name,
  salary,
  salary - avg(salary) OVER (PARTITION BY dept)
FROM
  employee
ORDER BY name;
```

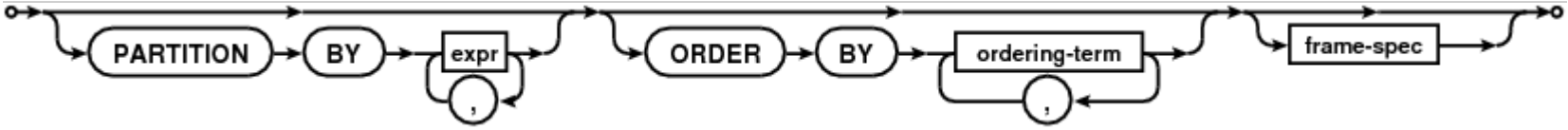
Window Functions

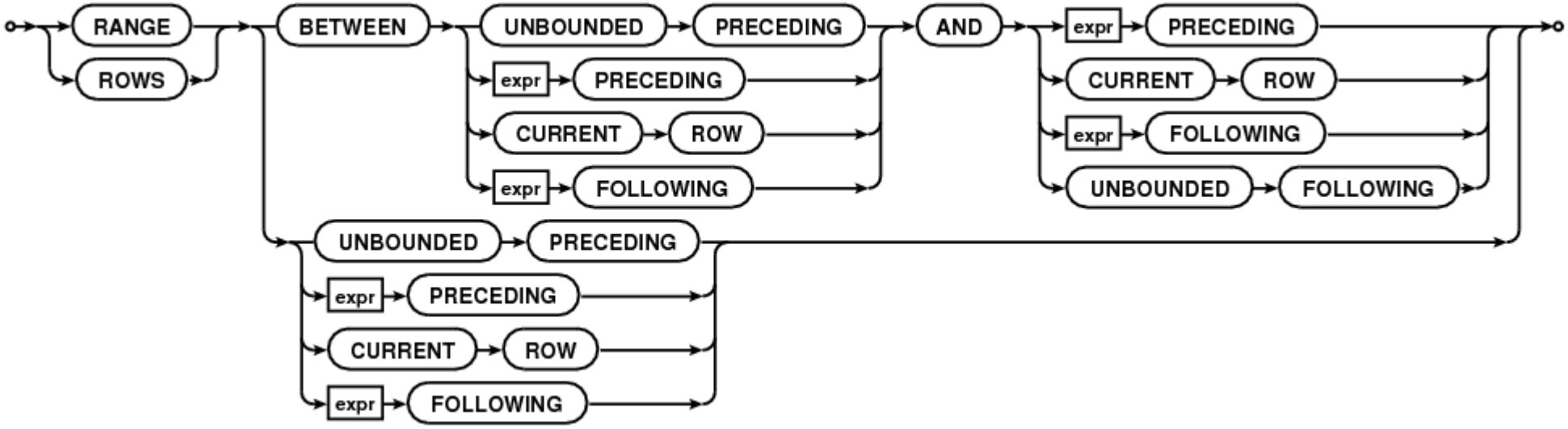
name	dept	salary
Alice	eng	5000.0
Bob	acct	3750.0
Cindy	art	2800.0
Drew	eng	4850.0
Ellen	sales	3000.0
Fred	sales	3100.0
Gina	eng	5100.0
Harry	sales	3050.0
Ingrid	acct	4000.0
Jack	eng	4900.0
Karen	sales	3800.0
Larry	art	3800.0
Mary	eng	2250.0
Nick	sales	2800.0
Olivia	acct	4000.0
Paul	eng	6000.0

name	salary	diff
Alice	5000.0	316.67
Bob	3750.0	-166.67
Cindy	2800.0	-500.00
Drew	4850.0	166.67
Ellen	3000.0	-150.00
Fred	3100.0	-50.00
Gina	5100.0	416.67
Harry	3050.0	-100.00
Ingrid	4000.0	83.33
Jack	4900.0	216.67
Karen	3800.0	650.00
Larry	3800.0	500.00
Mary	2250.0	-2433.33
Nick	2800.0	-350.00
Olivia	4000.0	83.33
Paul	6000.0	1316.67

Window Functions







Window Functions

- Window functions are implemented using an additional pass over the output rows
- The SQLite window function syntax matches PostgreSQL
- Window functions are mostly used for analytic queries - queries that summarize data
- Search the web for tutorials about window functions

Go Back....

sqlite_memstat

name	schema	value	hiwtr
MEMORY_USED	NULL	568768	572776
MALLOC_SIZE	NULL	NULL	120000
MALLOC_COUNT	NULL	967	996
PAGECACHE_USED	NULL	0	0
PAGECACHE_OVERFLOW	NULL	16384	16384
PAGECACHE_SIZE	NULL	NULL	4096
PARSER_STACK	NULL	NULL	0
DB_LOOKASIDE_USED	NULL	40	60
DB_LOOKASIDE_HIT	NULL	NULL	393
DB_LOOKASIDE_MISS_SIZE	NULL	NULL	0
DB_LOOKASIDE_MISS_FULL	NULL	NULL	0
DB_CACHE_USED	NULL	46752	NULL
DB_SCHEMA_USED	NULL	58912	NULL
DB_STMT_USED	NULL	15600	NULL
DB_CACHE_HIT	NULL	9	NULL
DB_CACHE_MISS	NULL	8	NULL
DB_CACHE_WRITE	NULL	0	NULL
DB_DEFERRED_FKS	NULL	0	NULL
ZIPVFS_CACHE_USED	main	9208	NULL
ZIPVFS_CACHE_USED	aux1	9208	NULL
ZIPVFS_CACHE_HIT	main	16	NULL
ZIPVFS_CACHE_HIT	aux1	6	NULL
ZIPVFS_CACHE_MISS	main	1	NULL
ZIPVFS_CACHE_MISS	aux1	1	NULL
ZIPVFS_CACHE_WRITE	main	0	NULL
ZIPVFS_CACHE_WRITE	aux1	0	NULL
ZIPVFS_DIRECT_READ	main	10	NULL
ZIPVFS_DIRECT_READ	aux1	5	NULL
ZIPVFS_DIRECT_BYTES	main	13767	NULL
ZIPVFS_DIRECT_BYTES	aux1	16265	NULL

Go Back...