Database

Santa Clara University
Center for Science, Technology, and Society

Introduction - Databases

- Databases have incredible value to business.
  - Probably the most important technology for support operations.

- Vastly superior to file processing systems

- Businesses cannot survive without quality data about their
  - Internal operations
  - External environment.

Foundation Data Concepts (2)

Database Concepts

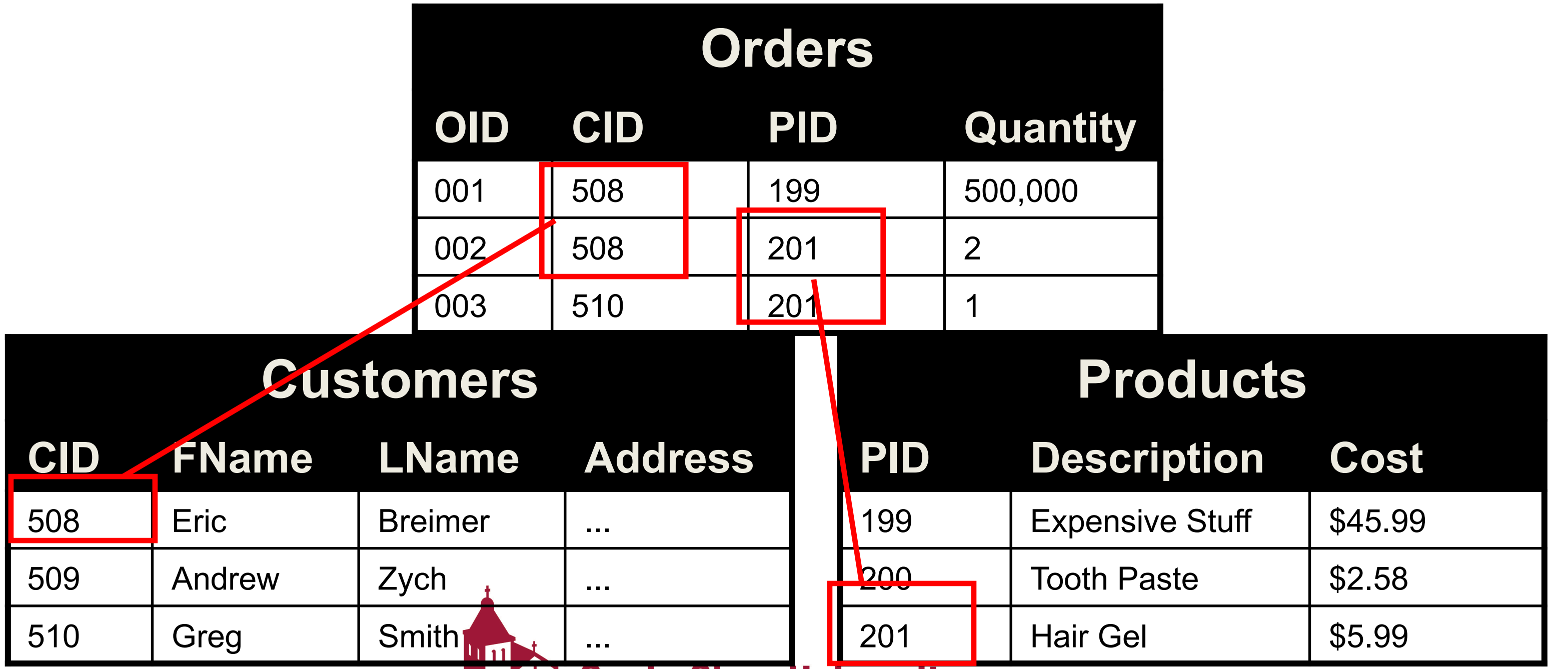
- Database : a collection of related tables
- Tables : a collection of related records
- Record : collection of fields (table row)
- Field : collection of characters (table column)
- Character : single alphabetic, numeric or other symbol

Santa Clara University

(Center for Science, Technology, and Society

Databases (2)

- But, databases are not just a bunch of tables
- A database also includes relationships between the different tables



Orders

OID	CID	PID	Quantity
001	508	199	500,000
002	508	201	2
003	510	201	1

Customers

CID	FName	LName	Address
508	Eric	Breimer	...
509	Andrew	Zych	...
510	Greg	Smith	...

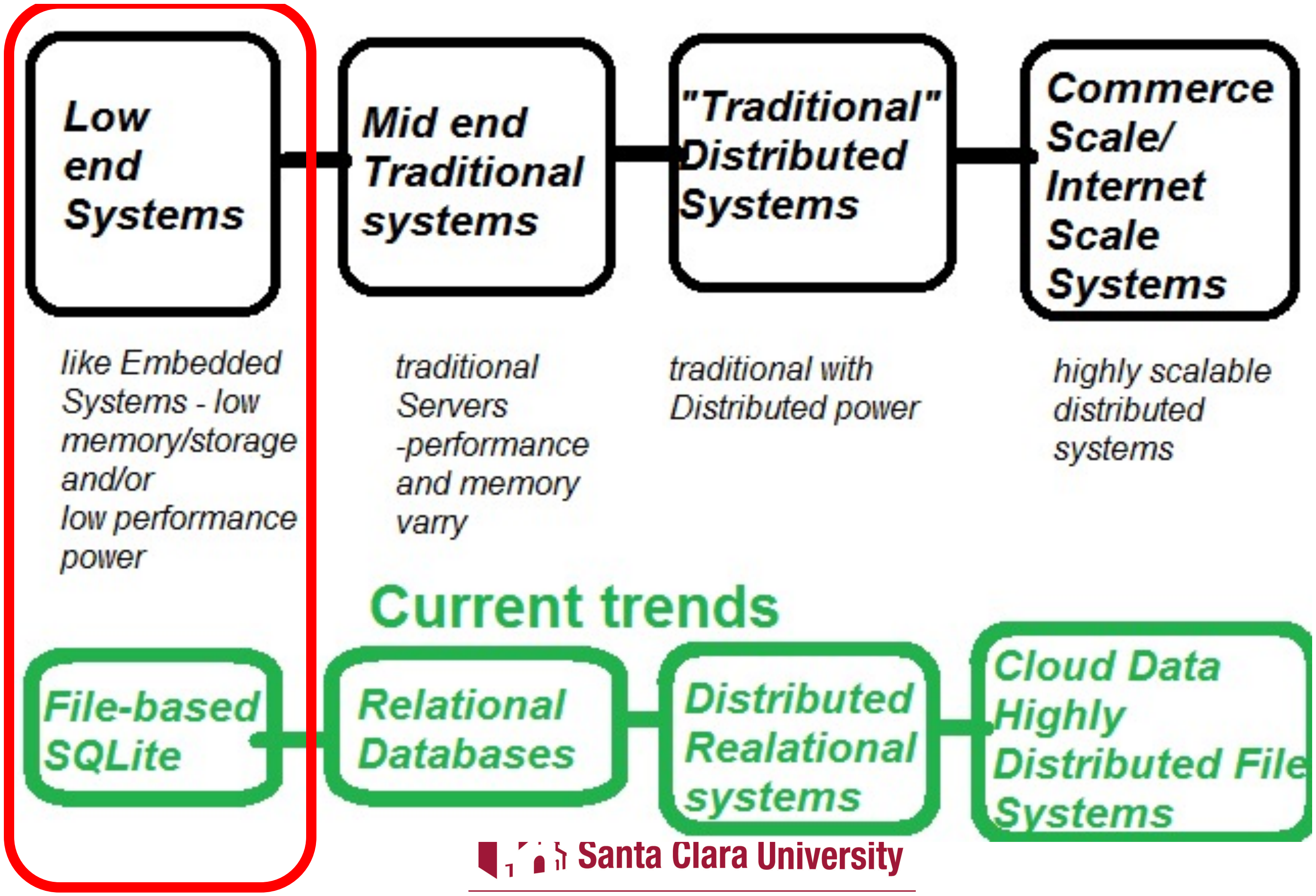
Products

PID	Description	Cost
199	Expensive Stuff	$45.99
200	Tooth Paste	$2.58
201	Hair Gel	$5.99

SQL VS. NoSQL

- **Relational databases** represent and store data in tables and rows. They're based on a branch of algebraic set theory known as relational algebra.
  - MySQL, PostgreSQL and SQLite3
- **non-relational databases** represent data in collections of JSON documents
  - column store: key -> column 1, 2, 3 (each column is a name-value -time stamp); **Cassandra**
  - key-value: uses map or dictionary data structure; **Redis**
  - graph: **neo4J, titan**
  - document oriented DB: user document, contact document, access document, etc .., **MongoDB**

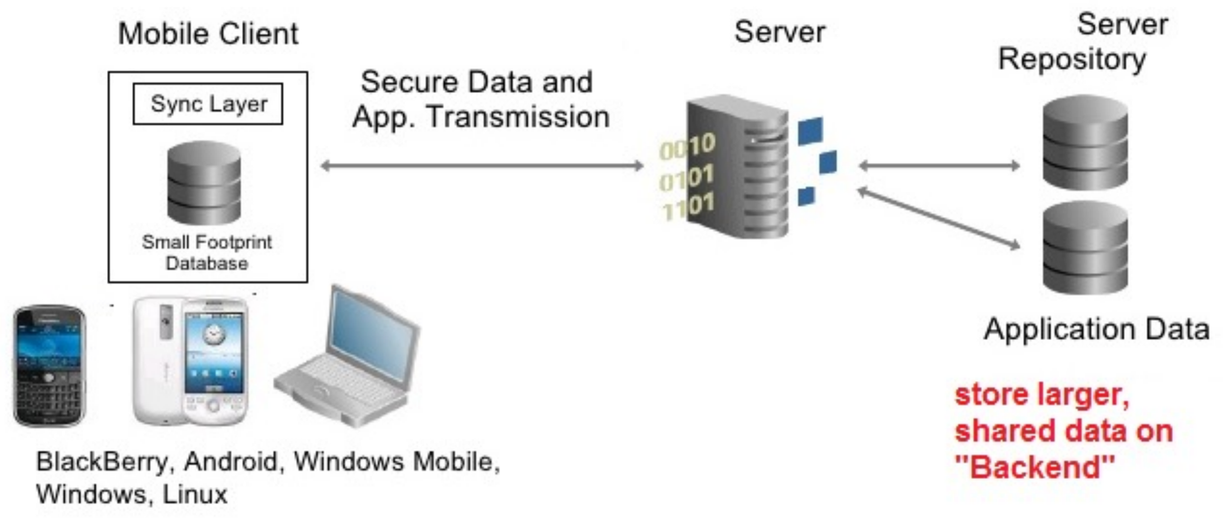
Whole range of Data solutions in general


ON Device Option
Low end Systems
Mid end Traditional systems
"Traditional" Distributed Systems
Commerce Scale/ Internet Scale Systems
like Embedded Systems - low memory/storage and/or low performance power
traditional Servers -performance and memory varry
traditional with Distributed power
highly scalable distributed systems
Current trends
File-based SQLite
Relational Databases
Distributed Realational systems
Cloud Data Highly Distributed File Systems

Santa Clara University
Center for Science, Technology, and Society

Data Architecture

- You may/can have many different “types” of ways and places you store data
  - mobile device, backend (remote)
- What you store locally may be different than what you store remotely
    - If it overlaps –you need to worry about synchronization or have a policy for how synched the data is


Mobile Client
Sync Layer
Small Footprint Database
Secure Data and App. Transmission
Server
0010
0101
1101
Server Repository
Application Data
store larger, shared data on "Backend"
BlackBerry, Android, Windows Mobile, Windows, Linux

Why Store on Device Database?

- When Data is particular to this user and does not need to be shared or does not need/want to be saved
- Sometimes you can temporally sync your local database data to remotely stored data –so can work off line (no internet) or work "faster".
- ONLY for small amounts of data → if more, then go to server-side database solution

android.database.sqlite.SQLiteDatabase.getMaximumSize() gives max size

SQLite

Android Database = SQLite

- SQLite has a relatively small memory footprint and decent speed.
- SQLite is public domain.
  - Lots of firms (Adobe, Apple, Google, Oracle) and open source projects (Mozilla, PHP, Python) all ship products with SQLite.

SQLite

- The native API of SQLite is not JDBC.
  - too much overhead for a memory-limited device like a phone
- Activities may access database directly in <u>code</u> OR access it via a <u>content provider or service</u>
- <u>Not a client/server architecture</u>
  - Accessed via function calls from the application

Santa Clara University
Center for Science, Technology, and Society

SQLite – what it meant for

- SQLite: (mainly) focused on single-threaded use cases with smallish datasets
- Apple and Google chose to use it in iOS and Android respectively
- For mobile – meant for use on single device

Santa Clara University
Center for Science, Technology, and Society

SQLite DB: only your classes in your app's package

- You *cannot access internal databases belonging to other apps or other packages from class it was created in*
  - *To do this you must use Content Providers or external SD resident DBs* *(ANOTHER TOPIC)*

SQLite Data Types

- The biggest difference from other SQL databases is data typing.
- **You can put whatever data you want in whatever column you want.**
  - You can put a string in an INTEGER column and vice versa.
  - SQLite refers to this as "manifest typing“.
- There are some standard SQL features not supported in SQLite:
    - FOREIGN KEY constraints,
    - nested transactions,
    - RIGHT OUTER JOIN and FULL OUTER JOIN,
    - and some flavors of ALTER TABLE.

SQLite Storage Classes

- Some differences with normal SQL data types so please read: http://www.sqlite.org/datatype3.html

- **NULL**. The value is a NULL value.
- **INTEGER**. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- **REAL**. The value is a floating point value, stored as an 8-byte IEEE floating point number.
- **TEXT**. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- **BLOB**. The value is a blob of data, stored exactly as it was input.

Comparison of SQLite and SQL Data Types

Example Type names From The CREATE TABLE Statement or CAST Expression	Resulting SQLite data type
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER

CHARACTER(20)
VARCHAR(255)
VARYING CHARACTER(255)
NCHAR(55)
NATIVE CHARACTER(70)
NVARCHAR(100)
TEXT
CLOB
	TEXT

BLOB *no datatype specified*	NONE
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC

Example SQLite to Create a Table

```
CREATE TABLE t1( t TEXT,
                 nu NUMERIC,
                 i INTEGER,
                 r REAL,
                 no BLOB);
```

NOTE SQLite allows and will convert to proper data type for you (easier than normal SQL which you must enter in correct data type)

```
INSERT INTO t1 VALUES('500.0', '500.0', '500.0', '500.0', '500.0’);
```

```
sqlite> CREATE TABLE t1(c1 INTEGER, c2 DATE);
sqlite> INSERT INTO t1 VALUES('abcd', '14/23/2033’);
sqlite> SELECT * FROM t1;
abcd|14/23/2033
```

Santa Clara University

Center for Science, Technology, and Society

Example SQLite to Create a Table

```
CREATE TABLE t1( a TEXT,                -- text affinity
                 b NUMERIC,             -- numeric affinity
                 c BLOB,                -- no affinity
                 d                      -- no affinity
               );

-- Values will be stored as TEXT, INTEGER, TEXT, and INTEGER respectively
INSERT INTO t1 VALUES('500', '500', '500', 500);
SELECT typeof(a), typeof(b), typeof(c), typeof(d) FROM t1;
    text|integer|text|integer
SELECT a < '40', a < '60', a < '600' FROM t1;
SELECT a < 40, a < 60, a < 600 FROM t1;
SELECT b < 40, b < 60, b < 600 FROM t1;
SELECT c < 40, c < 60, c < 600 FROM t1;
SELECT d < '40', d < '60', d < '600' FROM t1;
```

Santa Clara University

Center for Science, Technology, and Society

android.database.sqlite

- Contains the SQLite database management classes that an application would use to manage its own private database.

Working with SQLite

Steps in working with Sqlite database:

- ✓ Creating Sqlite Object
- ✓ Creating Database
- ✓ Creating Table
- ✓ Working with Tables

Create SQLiteObject

Creating Sqlite Object

Step:1 Importing package "android.database.sqlite.SQLiteDatabase".

Step:2 Creating object
SQLiteDatabase mydb;

Create DB

Creating Database Name

EX: mydb=openOrCreateDatabase("DatabaseName", *MODE_PRIVATE,null);*

//mydb is sqlite object name .
//DatabaseName is nothing but database name
//MODE_PRIVATE is permissions of a table accessing

Create Table

Creating Table

```
mydb.execSQL("CREATE TABLE IF NOT EXISTS
“ +TableName+" (ColumnName  DataType);");
```

Work on Tables

DDL(Create ,Alter ,Drop)

Create:

mydb.execSQL("CREATE TABLE IF NOT EXISTS " +TableName+" (ColumnName DataType);");

Alter:

ALTER TABLE TableName RENAME TO *new-table-name*

Drop:

DROP TABLE TableName

Work on Tables (2)

DML(SELECT , INSERT , DELETE)

Select:

```
Cursor c=mydb.rawQuery("SELECT * FROM "+TableName+" where Name='"+city+"'",null);
```

Insert:

```
mydb.execSQL("INSERT INTO "+TableName+" (Name, Area)" + "VALUES ('RedFort','40.8 acres');");
```

Delete:

```
mydb.execSQL("Delete"+TableName);
```

Santa Clara University
Center for Science, Technology, and Society

SQLiteDatabase

- Contains the methods for: creating, opening, closing, inserting, updating, deleting and quering an SQLite database

- These methods are similar to JDBC but more method oriented than what we see with JDBC (remember there is not a RDBMS server running)

SQLiteOpenHelper

- SQLite OpenHelper is a class to manage database creation and version management
- <u>Opening</u> the database if it exists, <u>creating</u> it if it does not, and <u>upgrading</u> it as necessary
- This is for creating db *"onCreate(SQLiteDataBase)"*
- When the database needs to be upgraded *"onUpgrade (SQLiteDataBase db, int oldVersion, int newVersion)"*
- When the database has been opened ***"onOpen (SQLiteDataBase db)"***

NOTE

- No databases are automatically supplied to you by Android.
- If you want to use SQLite, you have to create your own database, then populate it with your own tables, indices, and data.

Create Database in Code

- **Option 1:** SQLiteDatabase.openDatabase(*)

- **Option 2:** SQLiteDatabase.openOrCreateDatabase(*)

- **Option 3:** using your own SQLiteOpenHelper class

1: SQLiteDatabase.openDatabase(*)

public static *SQLiteDatabase.openDatabase(**String path,**
SQLiteDatabase.CursorFactory factory, int flags)

```
    android.database.sqlite.SQLiteDatabase db;
   db =SQLiteDatabase.openDatabase("/data/data/ApplicationPACKAGE/databases/MyDB",
                                        null, SQLiteDatabase.CREATE_IF_NECESSARY);
```

- **Open the database according to the flags OPEN_READWRITE, OPEN_READONLY CREATE_IF_NECESSARY . Sets the locale of the database to the system's current locale.**
- ***Parameters***
  - **path to database file to open and/or create**
  - **factory = *optional* factory class that is called to instantiate a cursor when query is called, or null for default**
  - **Flags to control database access mode**
  - **Returns the newly opened database**
  - **Throws *SQLiteException if the database cannot be opened***

2: SQLiteDatabase.openOrCreateDatabase(*)

public static *SQLiteDatabase.openOrCreateDatabase(*String path, MODE, SQLiteDatabase.CursorFactory factory)*

```
SQLiteDatabase db;
db = this.openOrCreateDatabase ("/data/data/ApplicationPACKAGE/databases/MyDB",
                                MODE_PRIVATE, null);
```

- Open the database according to the flags OPEN_READWRITE OPEN_READONLY CREATE_IF_NECESSARY . Sets the locale of the database to the system's current locale.
- *Parameters*

  1. path The prefix is assigned by Android as: /data/data/<app namespace>/databases/MyDB.
  2. MODEcould be: MODE_PRIVATE, MODE_WORLD_READABLE, and MODE_WORLD_WRITEABLE. Meaningful for apps consisting of multiple activities.
  3. Null refers to optional factory class parameter (skip for now)

Santa Clara University
Center for Science, Technology, and Society

3: SQLiteOpenHelper.getWriteableDatabase(*)

- **public *SQLiteOpenHelper.getWritableDatabase()***

  android.database.sqlite.SQLiteDatabase db;

  db= DatabaseHelper.getWriteableDatabase();

  This method will open an existing database or create one in the application data area first time this is called, the database will be opened and onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) and/or onOpen(SQLiteDatabase) will be called

- You must create DatabaseHelper class extending SQLiteOpenHelper

Santa Clara University
Center for Science, Technology, and Society

DatabaseHelper extends SQLiteOpenHelper

- Your subclass DatabaseHelper of SQLiteOpenHelper will need three methods:
  - **DatabaseHelper(*)** calling parent constructor.
  - **onCreate(*)**, which passes you a SQLiteDatabase object that you need to populate with tables and initial data.
  - **onUpgrade(*)**, which passes you a SQLiteDatabase object and the old and new version numbers, so you can figure out how best to convert the database from the old schema to the new one.

DatabaseHelper Constructor

- constructor takes Context (e.g., an Activity), the name of the database, an optional cursor factory (typically, just pass null), and an integer representing the version of the database schema you are using.

This DatabaseHelper class creates a table named myTable

```
class DatabaseHelper extends SQLiteOpenH
        private static final int DATABASE_VERSION = 2;
        private static final String DATABASE_NAME = “myDB";
        private static final String DICTIONARY_TABLE_NAME = “myTable";
        private static final String KEY_WORD = “TITLE”;
        private static final String KEY_DEFINITION = “DEFINITION”;
        private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +  KEY_WORD +" TEXT,
      “ + KEY_DEFINITION + " TEXT);";

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null,
        }
```

Here we have set the class variables DATABASE_NAME and DATABASE VERSION

Example:

```
static final String DATABASE_NAME = “myDB";
static final int DATABASE_VERSION = 2;
```

Santa Clara University
Center for Science, Technology, and

- call execSQL() on your SQLiteDatabase, providing the SQL statement you wish to apply against the database.

```
// INSIDE DatabaseHelper class
@Override
    public void onCreate(SQLiteDatabase db)
    {
       try {
          db.execSQL(DATABASE_CREATE);
       } catch (SQLException e) {
          e.printStackTrace();
       }
    }
```

Here we have set the class variables DATABASE_CREATE

This Code:

“create table myTable (TITLE Text, DESCRIPTON Text);”

Different Example:

static final String *DATABASE_CREATE* =
"create table contacts (_id integer primary key autoincrement, “ + "name text not null, email text not null);";

Santa Clara U

Center for Science, Technology, and Society

DatabaseHelper onUpgrade()

- It passes you a SQLiteDatabase object and the old and new version numbers, you figure out how best to convert the database from the old schema to the new one.

- The simplest, albeit least friendly, approach is to simply drop the old tables and create new ones.

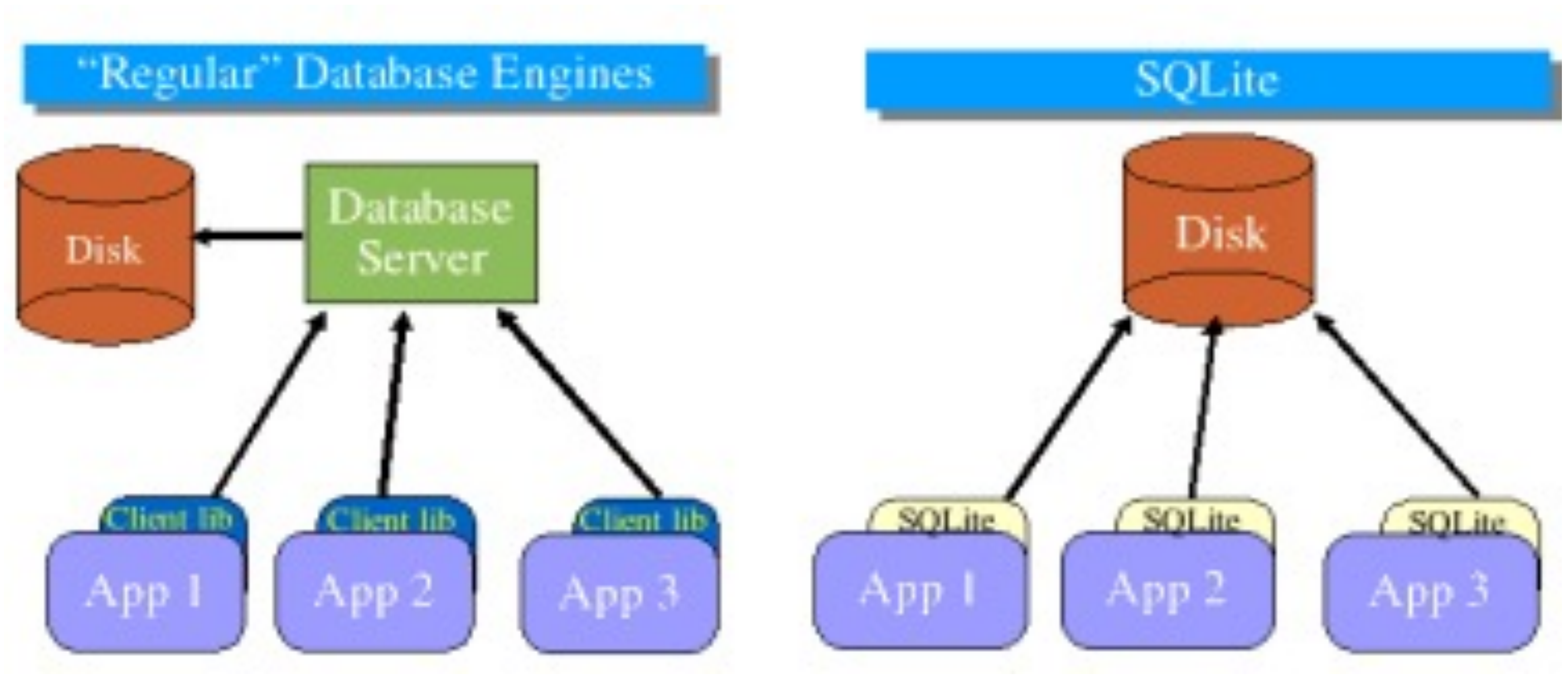
```
// INSIDE DatabaseHelper class
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)
    {
      Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
      db.execSQL("DROP TABLE IF EXISTS contacts");   //we simply drop the old table contacts
                                                     //in old database
      onCreate(db);    //create new database
    }
```

Android SQLite Tutorial

https://www.youtube.com/watch?v=cp2rL3sAFml

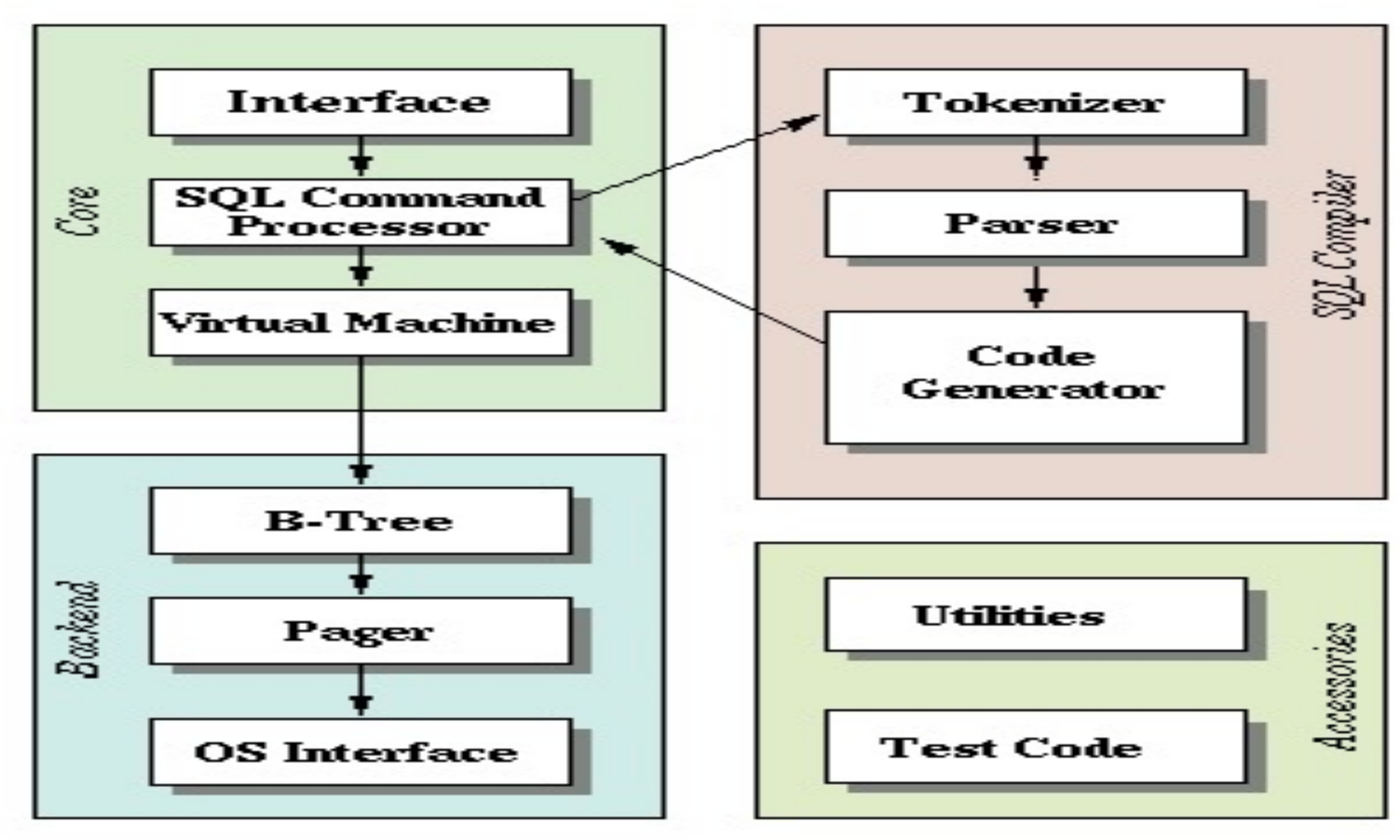
https://www.youtube.com/watch?annotation_id=annotation_2120971985&feature=iv&src_vid=cp2rL3sAFml&v=p8TaTgr4uKM

Difference between SQLite and other DB



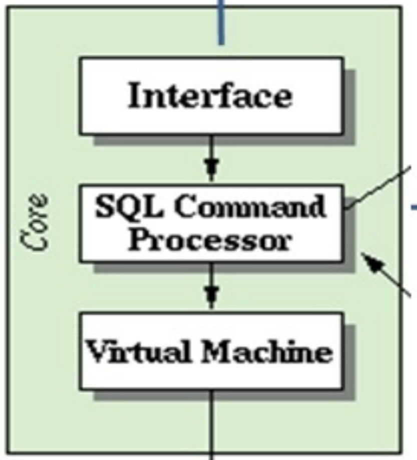
Santa Clara University
Center for Science, Technology, and Society

Architecture of SQLite



Santa Clara University
Center for Science, Technology, and Society

Architecture of SQLite (2)

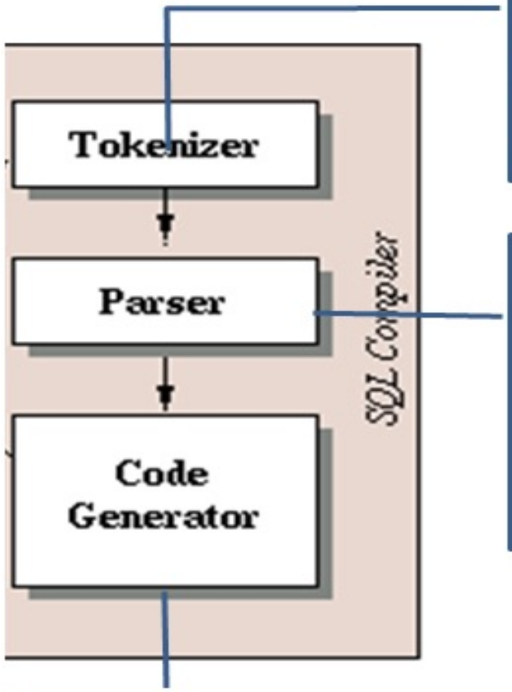


This interface will read the SQL queries what ever it is generated by the user and it will pass this SQL Commands to the SQL Command Processor

The SQL Command Processor will read the SQL Commands from Interface and it will forward that SQL Query Strings to the Tokenizer of SQL Compiler

This Virtual Machine will read the code what ever the code generated by the code generator of the SQL Compiler

Architecture of SQLite (2)



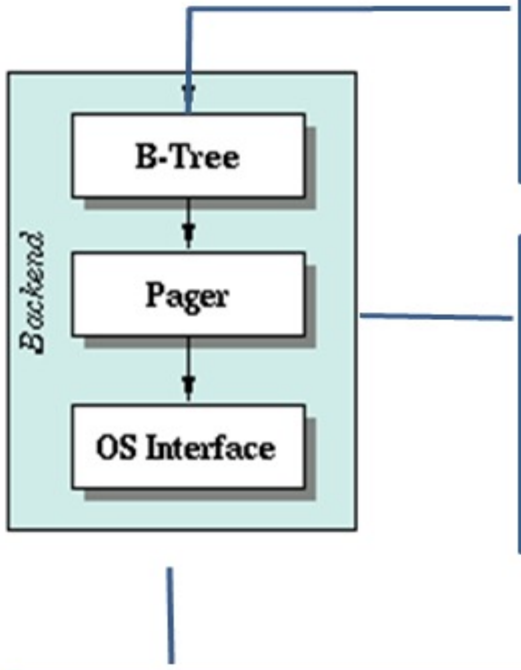
When a string containing SQL statements is to be executed, the interface passes that string to the tokenizer. The job of the tokenizer is to break the original string up into tokens and pass those tokens one by one to the parser. The tokenizer is hand-coded in C in the file **tokenize.c**.

The parser is the piece that assigns meaning to tokens based on their context, it does not leak memory when syntax errors are encountered, parser assembles tokens into complete SQL statements

the code generator to produce virtual machine code that will do the work that the SQL statements request. Ex: **vacuum.c** and **where.c** in which **where.c** handles code generation for WHERE clauses on SELECT, UPDATE and DELETE statements

Santa Clara University
Center for Science, Technology, and Society

Architecture of SQLite (2)



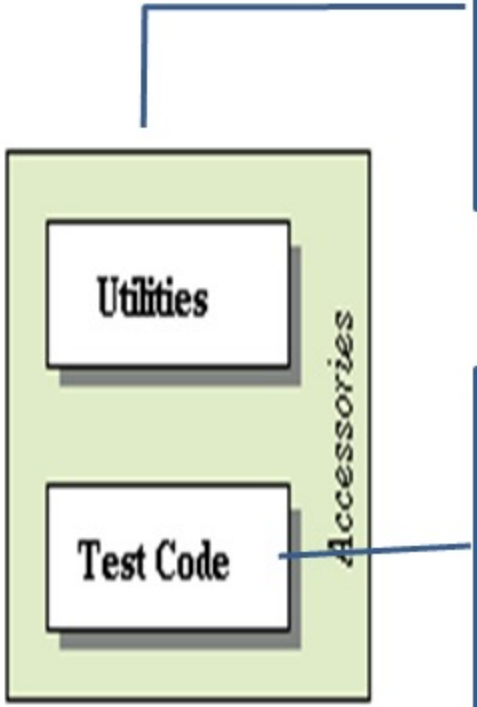
An SQLite database is maintained on disk using a B-tree implementation found in the **btree.c** source file. A separate B-tree is used for each table and index in the database. All B-trees are stored in the same disk file. Details of the file format are recorded in a large comment at the beginning of **btree.c**.

The B-tree module requests information from the disk in fixed-size chunks. The default chunk size is 1024 bytes but can vary between 512 and 65536 bytes. The page cache is responsible for reading, writing, and caching these chunks. The page cache also provides the rollback and atomic commit abstraction and takes care of locking of the database file.

In order to provide portability between POSIX (Portable Operating System Interface (for Unix))and Win32 operating systems, SQLite uses an abstraction layer to interface with the operating system. The interface to the OS abstraction layer is defined in **os.h.**

Santa Clara University
Center for Science, Technology, and Society

Architecture of SQLite (2)



Memory allocation and ceaseless string comparison routines are located in **util.c**

If you count regression test scripts, more than half the total code base of SQLite is devoted to testing. There are many **assert()** statements in the main code files

Thank You

Santa Clara University

Center for Science,Technology, and Society

Center for Science, Technology, and Society