SQL

Devon Cormack, David Cain, Tyler Harley

2012-11-28

History

Creation

- language for relational databases
- IBM
  - Early 1970's
  - `SEQUEL` $\longrightarrow$ `SQL`

Modern day

- many implementations
  - SQL Server
  - SQLite
  - MySQL
  - PostgreSQL
  - Oracle SQL

MySQL

MySQL

- FOSS
- popular
- scalable
  - Facebook
  - Twitter
- procedural support

Syntax

comments

- `-- line comment`
- `({ block comment})`
- `/* C-style block comment */`

symbols

- `SET @bob = 6; -- Sets bob to 6`
- `DECLARE var1 INT; SET var1 = 0;`

Expressions

```
SET @a = 1;
SET @b = 2;
SET @c = 3;
SET @d = 4;
IF  d < a OR b < c OR b < d THEN @q = 8;
```

Common Programming Statements

Core functions of persistent storage

- **C**: `INSERT`
- **R**: `SELECT`
- **U**: `UPDATE`
- **D**: `DELETE`

MySQL Procedural statements

- `CREATE PROCEDURE`
- `CREATE FUNCTION`
- `IF...THEN...ELSE`
- `label:LOOP...END LOOP`

Executing SQL statements

Interpreted or compiled?

- Depends on implementation
  - SQLite (`prepare()` functions)
- SQL optimization
  - SQL String → (Optimizer) → Execution Plan
  - Execution Plan → (Execution) → Result

Execute procedures/functions

```
CALL testproc(@a) -- stored procedure
SELECT name(in)   -- stored function
```

Memory management

Memory management

- Server managed
- Specify memory tables with memory engine

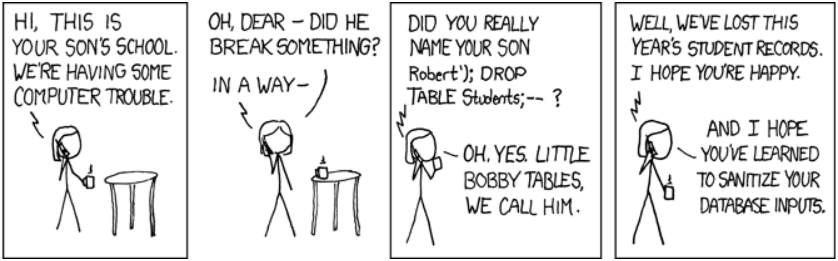
```
CREATE TABLE students ENGINE=MEMORY;
```

Interesting and Identifying Features of SQL

- non-procedural
- implicitly parallel
- declarative
- case-insensitive

SQL Injection

- Malicious exploitation of poorly written applications


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

SQL Injection

Comic Explanation

- Given frontend to school's database, input string name:

  `SELECT * FROM Students WHERE (student_name=name);`
- `name = "Robert'); DROP TABLE Students; --"`
- Resulting query:
  - `SELECT * FROM Students WHERE (student_name= 'Robert'); DROP TABLE Students;-- ');`

SQL Injection

How to prevent it?

- End goal: Sanitize inputs
- (Don't do it yourself!)
- Solution: **parameterized statements**
- See bobby-tables.com for language-specific examples.

Bad- vulnerable to injection

```
cmd = ("SELECT * FROM Students WHERE"
       "(user_name = '%s')" % student_name)
curs.execute(cmd)
```

Good - uses parameterized statements

```
cmd = "SELECT * FROM Students WHERE (student_name='%s')"
curs.execute(cmd, student_name)
```