CPE 455/555
Secure Software Development
Lab04 - SQL

Dr. David J. Coe
Software Safety and Security Engineering Lab
Electrical and Computer Engineering Department

Outline

- SQLite Crash Course
  - Hands-On: Query SQLite Table
- SQL Injection
- Complete **Quiz-Lab04**

SQLite Crash Course

SQLite Crash Course - 1

- Documentation for SQLite3 available at **sqlite.org**

SQLite Crash Course - 2

- **Identifiers**
  - Up to 128 characters in length
  - May contain letters, digits, and underscores
  - ***Must start with a letter***
  - Cannot use reserved words
  - ***Quoted/delimited identifier***
    - To use special characters in the identifier, you must **double quote** the identifier

    "% Complete"
  - Literal String: 'stuff'
  - Identifier: "stuff"

SQLite Reserved Words

ABORT
ACTION
ADD
AFTER
ALL
ALTER
ALWAYS
ANALYZE
AND
AS
ASC
ATTACH
AUTOINCREMENT
BEFORE
BEGIN
BETWEEN
BY
CASCADE
CASE
CAST
CHECK
COLLATE
COLUMN
COMMIT
CONFLICT
CONSTRAINT
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
DATABASE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DETACH
DISTINCT
DO
DROP
EACH
ELSE
END
ESCAPE
EXCEPT
EXCLUDE
EXCLUSIVE
EXISTS
EXPLAIN
FAIL
FILTER
FIRST
FOLLOWING
FOR
FOREIGN
FROM
FULL
GENERATED
GLOB
GROUP
GROUPS
HAVING
IF
IGNORE
IMMEDIATE
IN
INDEX
INDEXED
INITIALLY
INNER
INSERT
INSTEAD
INTERSECT
INTO
IS
ISNULL
JOIN
KEY
LAST
LEFT
LIKE
LIMIT
MATCH
NATURAL
NO
NOT
NOTHING
NOTNULL
NULL
NULLS
OF
OFFSET
ON
OR
ORDER
OTHERS
OUTER
OVER
PARTITION
PLAN
PRAGMA
PRECEDING
PRIMARY
QUERY
RAISE
RANGE
RECURSIVE
REFERENCES
REGEXP
REINDEX
RELEASE
RENAME
REPLACE
RESTRICT
RIGHT
ROLLBACK
ROW
ROWS
SAVEPOINT
SELECT
SET
TABLE
TEMP
TEMPORARY
THEN
TIES
TO
TRANSACTION
TRIGGER
UNBOUNDED
UNION
UNIQUE
UPDATE
USING
VACUUM
VALUES
VIEW
VIRTUAL
WHEN
WHERE
WINDOW
WITH
WITHOUT

SQLite Crash Course - 3

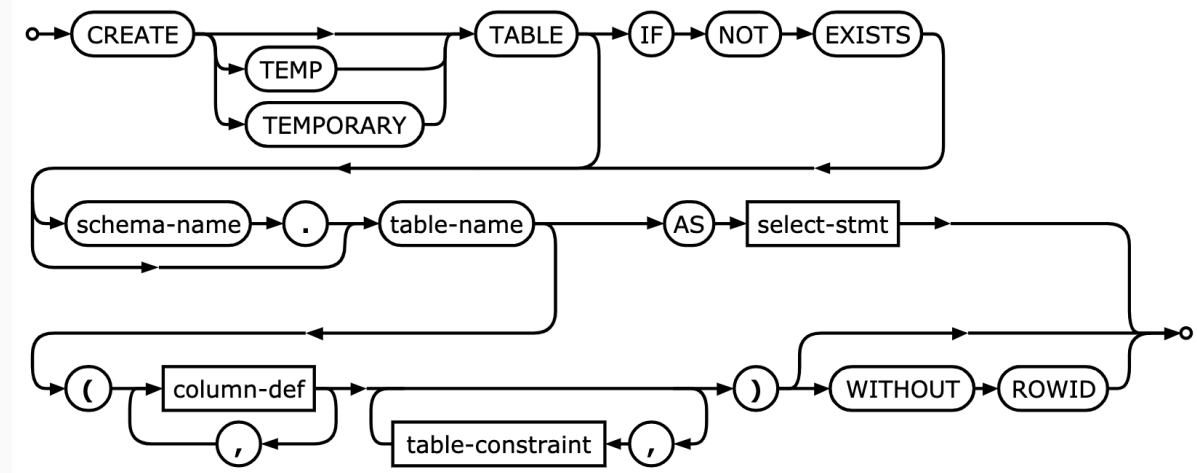
- **Data Types**
  - **NULL**
    - The value is a NULL value.
  - **INTEGER**
    - The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
  - **REAL**
    - The value is a floating-point value, stored as an 8-byte IEEE floating point number.
  -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

sqlite.org

SQLite Crash Course - 4

```
CREATE TABLE   table-name
(
   nameofcolumn1   datatype1,
   nameofcolumn2   datatype2,
   ...
);
```

- **CREATE** and **TABLE** are reserved words
- **table-name**
  - A valid identifier chosen to name the table
- **nameofcolumnX**
  - A valid identifier chosen to name column
- Don't forget the trailing semicolon



sqlite.org

SQLite Crash Course - 5

```
INSERT INTO table-name
  VALUES(value1, value2, ..., valueN);
```

```
INSERT INTO table-name
  (column1, column2, ..., columnN)
  VALUES(value1, value2, ..., valueN);
```

For clarity, column names should be listed in the order in which they appear in the table.

SQLite Crash Course - 6

```
SELECT    target-list
FROM      relation-list
WHERE     predicate;
```

- **`target-list`** is a list of one or more ***attributes*** $\mathbf{A_1, A_2, \ldots, A_n}$ of a relation in the specified **`relation-list`**
- **`relation-list`** lists the ***relations*** $\mathbf{R_1, R_2, \ldots, R_m}$ that will be considered in the evaluation of the query
- **`predicate`** is a simple or compound logical expression for comparing one or more attribute values

- Attributes = columns
- Relations = tables

sqlite.org

Hands-On Lab: Query an SQLite Table

Getting the Table into SQL - 1

- **Option 1**
  - Open a terminal window on **blackhawk**
  - Type **sqlite3** at the prompt
  - Type your **CREATE** statement(s) to create the table(s) you need
  - Type your **INSERT** statement(s) to populate the tables
  - Type your queries

Getting the Table into SQL - 2

- **Option 2**
  - Use your favorite editor to type up your SQL **CREATE** and **INSERT** statements in a text file
  - Open a terminal window on **blackhawk**
  - Type **sqlite3** at the prompt
  - Type **.read NameOfFile.txt**
  - Type your queries

Getting the Table into SQL - 3

ID - Integer	Name - Text	Price - Integer
1	Audi	52642
2	Mercedes	57127
3	Skoda	9000
4	Volvo	29000
5	Bentley	350000
6	Citroen	21000
7	Hummer	41400
8	Volkswagen	21600

https://zetcode.com/db/sqlitec/

Getting the Table into SQL - 4

```
CREATE TABLE Cars( ID INTEGER, Name TEXT, Price INTEGER);
INSERT INTO Cars VALUES(1, 'Audi', 52642);
INSERT INTO Cars VALUES(2, 'Mercedes', 57127);
INSERT INTO Cars VALUES(3, 'Skoda', 9000);
INSERT INTO Cars VALUES(4, 'Volvo', 29000);
INSERT INTO Cars VALUES(5, 'Bentley', 350000);
INSERT INTO Cars VALUES(6, 'Citroen', 21000);
INSERT INTO Cars VALUES(7, 'Hummer', 41400);
INSERT INTO Cars VALUES(8, 'Volkswagen', 21600);
```

https://zetcode.com/db/sqlitec/

Getting the Table into SQL - 5

```
-bash-4.2$ sqlite3 cars.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .read cars.txt
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE Cars( ID INTEGER, Name TEXT, Price INTEGER);
INSERT INTO "Cars" VALUES(1,'Audi',52642);
INSERT INTO "Cars" VALUES(2,'Mercedes',57127);
INSERT INTO "Cars" VALUES(3,'Skoda',9000);
INSERT INTO "Cars" VALUES(4,'Volvo',29000);
INSERT INTO "Cars" VALUES(5,'Bentley',350000);
INSERT INTO "Cars" VALUES(6,'Citroen',21000);
INSERT INTO "Cars" VALUES(7,'Hummer',41400);
INSERT INTO "Cars" VALUES(8,'Volkswagen',21600);
COMMIT;
sqlite> .quit
```

Getting the Table into SQL - 6

```
-bash-4.2$ sqlite3 cars.db
SQLite version 3.7.17 2013-05-20 00:56:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .dump
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE Cars( ID INTEGER, Name TEXT, Price INTEGER);
INSERT INTO "Cars" VALUES(1,'Audi',52642);
INSERT INTO "Cars" VALUES(2,'Mercedes',57127);
INSERT INTO "Cars" VALUES(3,'Skoda',9000);
INSERT INTO "Cars" VALUES(4,'Volvo',29000);
INSERT INTO "Cars" VALUES(5,'Bentley',350000);
INSERT INTO "Cars" VALUES(6,'Citroen',21000);
INSERT INTO "Cars" VALUES(7,'Hummer',41400);
INSERT INTO "Cars" VALUES(8,'Volkswagen',21600);
COMMIT;
sqlite>
```

Getting the Table into SQL - 7

```
sqlite> SELECT * FROM Cars WHERE Price <= 20000;
3|Skoda|9000
sqlite> SELECT * FROM Cars WHERE Name = 'Mercedes';
2|Mercedes|57127
sqlite> SELECT * FROM Cars WHERE Name >= 'Mercedes';
2|Mercedes|57127
3|Skoda|9000
4|Volvo|29000
8|Volkswagen|21600
sqlite>
```

Single Quotes Delimit String Literals in SQL

Getting the Table into SQL - 8

C to SQLite Interface

```
#include <sqlite3.h>
#include <stdio.h>
#include <string.h>

int main(void)
{
    sqlite3 *db;
    char *err_msg = 0;
    int rc = sqlite3_open("test.db", &db);

    if (rc != SQLITE_OK)
    {
        fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
        sqlite3_close(db);
        return 1;
    }

    char *sql = "DROP TABLE IF EXISTS Cars;"
                "CREATE TABLE Cars(Id INT, Name TEXT, Price INT);"
                "INSERT INTO Cars VALUES(1, 'Audi', 52642);"
                "INSERT INTO Cars VALUES(2, 'Mercedes', 57127);"
                "INSERT INTO Cars VALUES(3, 'Skoda', 9000);"
                "INSERT INTO Cars VALUES(4, 'Volvo', 29000);"
                "INSERT INTO Cars VALUES(5, 'Bentley', 350000);"
                "INSERT INTO Cars VALUES(6, 'Citroen', 21000);"
                "INSERT INTO Cars VALUES(7, 'Hummer', 41400);"
                "INSERT INTO Cars VALUES(8, 'Volkswagen', 21600);";
```

Getting the Table into SQL - 9

C to SQLite Interface

```
    printf("sql = %s\n", sql);

    rc = sqlite3_exec(db, sql, 0, 0, &err_msg);

    if (rc != SQLITE_OK )
    {
        fprintf(stderr, "SQL error: %s\n", err_msg);
        sqlite3_free(err_msg);
        sqlite3_close(db);
        return 1;
    }
    sqlite3_close(db);
    return 0;
}
-bash-4.2$
```

Executes sequence of commands pointed to by **sql**

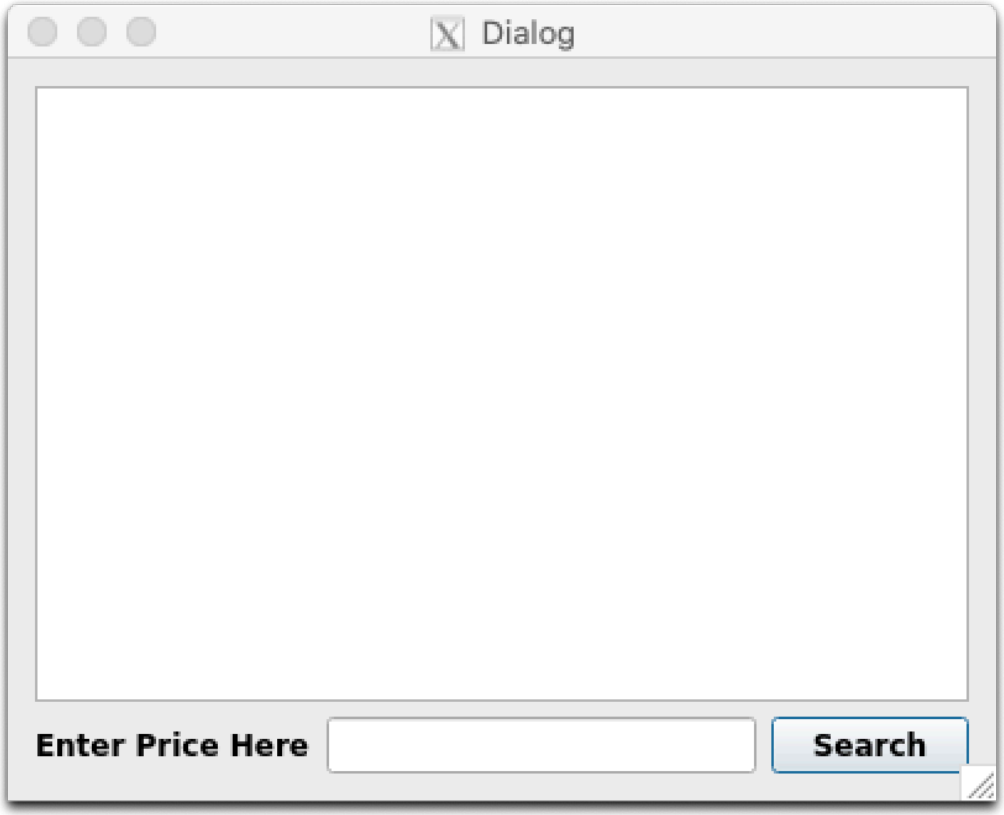
Getting the Table into SQL - 10

```
-bash-4.2$ ./a.out
sql = DROP TABLE IF EXISTS Cars;CREATE TABLE Cars(Id INT,
Name TEXT, Price INT);INSERT INTO Cars VALUES(1, 'Audi',
52642);INSERT INTO Cars VALUES(2, 'Mercedes',
57127);INSERT INTO Cars VALUES(3, 'Skoda', 9000);INSERT
INTO Cars VALUES(4, 'Volvo', 29000);INSERT INTO Cars
VALUES(5, 'Bentley', 350000);INSERT INTO Cars VALUES(6,
'Citroen', 21000);INSERT INTO Cars VALUES(7, 'Hummer',
41400);INSERT INTO Cars VALUES(8, 'Volkswagen', 21600);
-bash-4.2$
```

SQL Injection

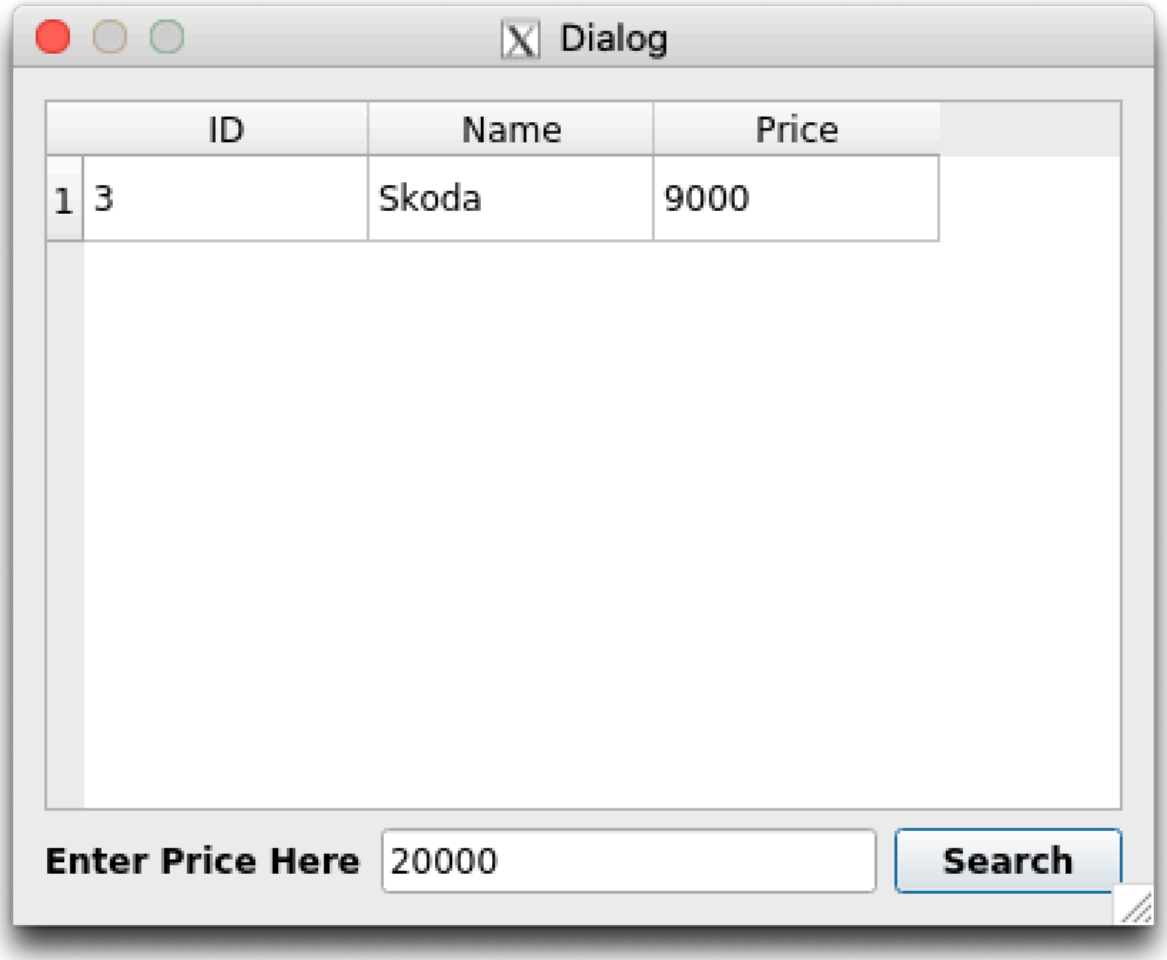
Vulnerable Qt-SQL Program - 1

- GUI provides user ability to search for cars where price is less than amount entered



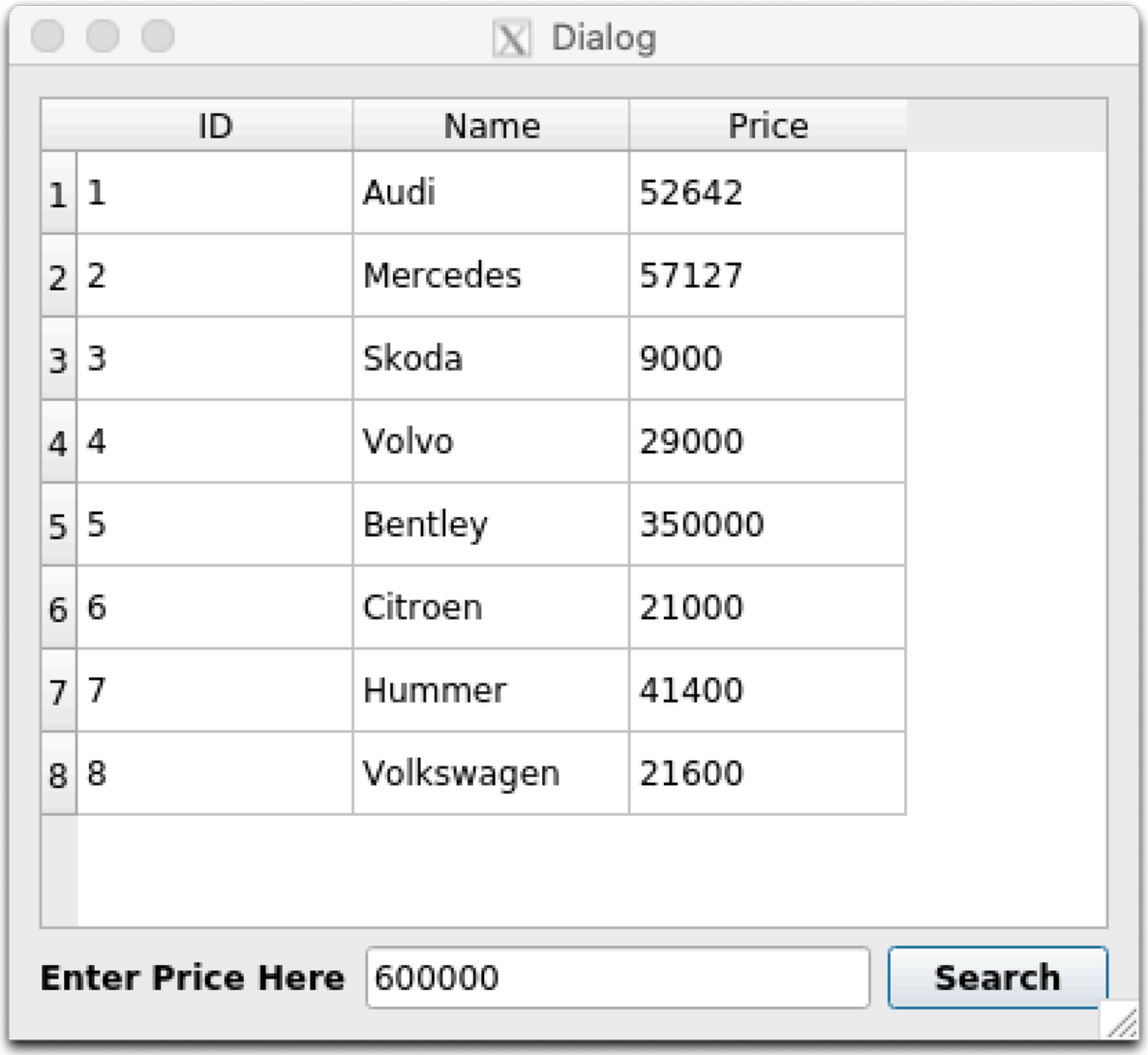
Vulnerable Qt-SQL Program - 2

- Example: Intended use of application



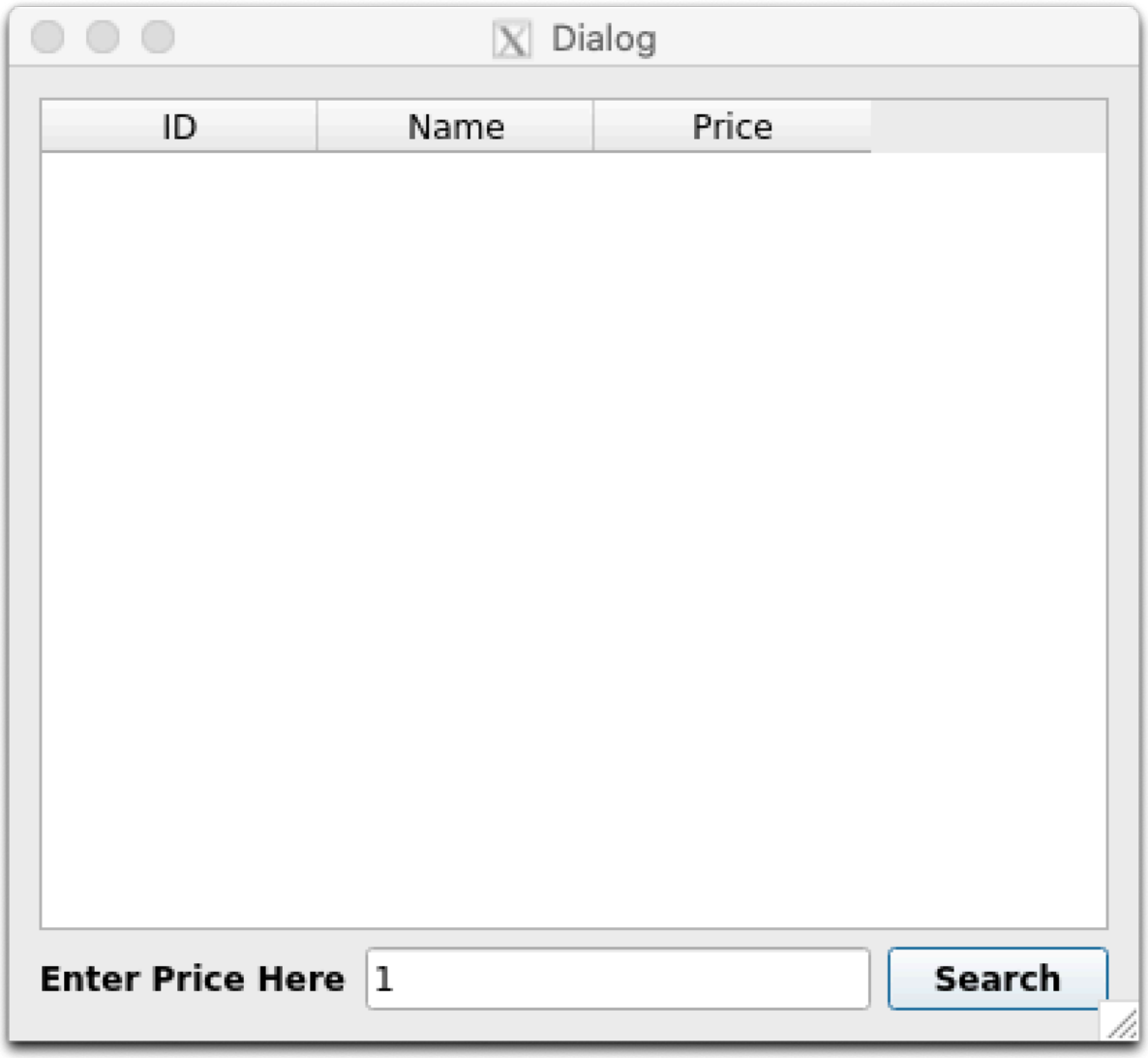
Vulnerable Qt-SQL Program - 3

- Example: Show all cars < $600,000



Vulnerable Qt-SQL Program - 3

- Example: Show all cars < $1



Vulnerable Qt-SQL Program - 4

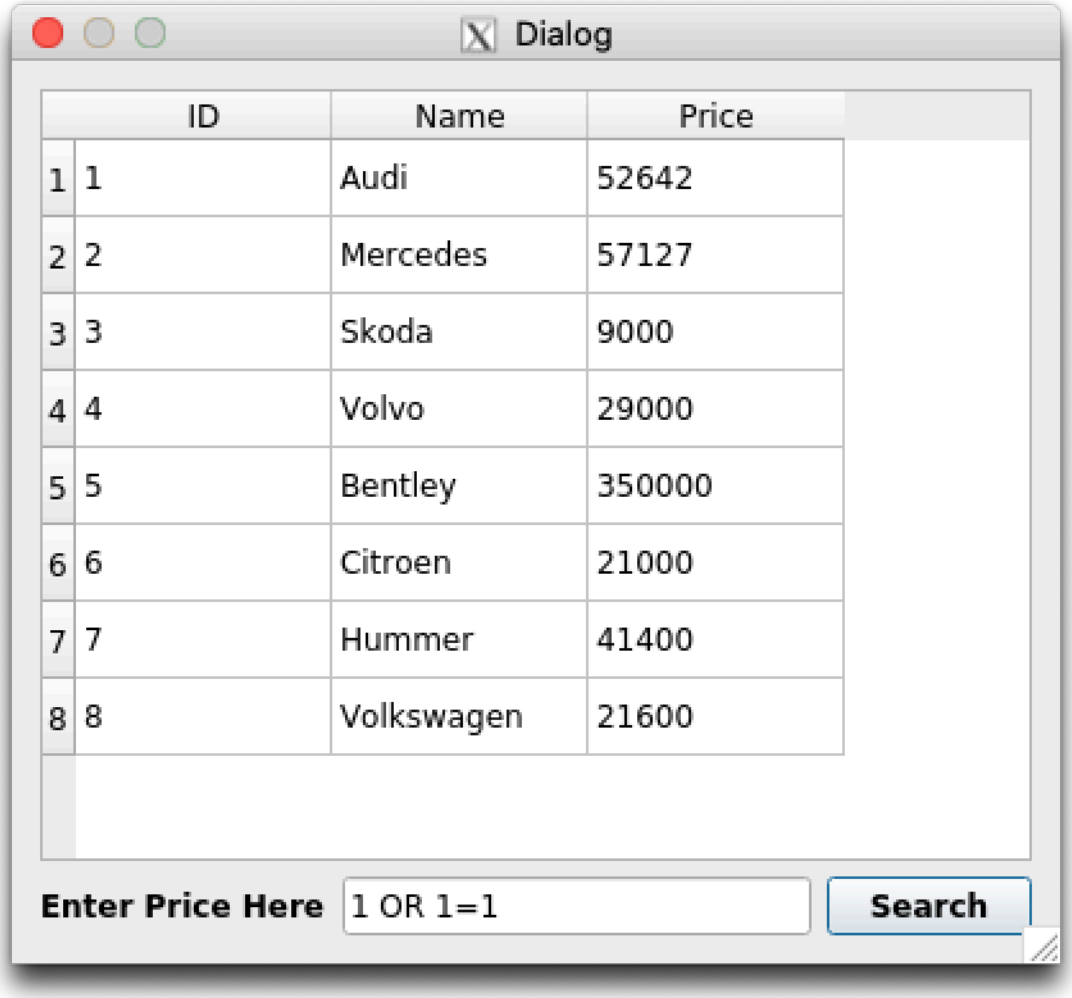
Qt to SQLite Interface

```
#include "dialog.h"
#include "ui_dialog.h"

Dialog::Dialog(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::Dialog)
{
    ui->setupUi(this);

    db = QSqlDatabase::addDatabase("QSQLITE");
    db.setDatabaseName("cars.db");
    if (!db.open())
        exit(1);

    connect(ui->pushButton, SIGNAL(clicked()), this, SLOT(searchDB()));
}

Dialog::~Dialog()
{
    delete ui;
}

void Dialog::searchDB()
{
    QString s = QString("SELECT * FROM cars WHERE Price < %1").arg(ui->lineEdit->text());
    QSqlQueryModel* model = new QSqlQueryModel;
    model->setQuery(s);
    ui->tableView->setModel(model);
}
```

SQL Injection Demo

SQL Injection Demo

Why does this input reveal the entire **Cars** table?




```
SELECT * FROM cars WHERE Price < 1 OR 1=1
```

Complete Quiz-Lab04