Lecture SQL04
Basic SQL – Part I

UAH
CPE 353

Outline

- **SQL** Basics
  - Identifiers and Data Types
  - Creating a Table in **SQL**
  - Adding Rows to the Table
  - Getting the Table into **sqlite3**
- Mapping **RA** to **SQL**

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”

Data Types - 1

- Exact Numeric quantities
  - INTEGER, DECIMAL, NUMERIC, etc.
- Approximate Numeric quantities
  - FLOAT, REAL, etc.
- Logical quantities TRUE, FALSE, UNKNOWN
  - BOOLEAN

Data Types - 2

- Fixed length character sequences
  - CHARACTER(length)
  - CHAR(length)
- Variable length character sequences
  - CHARACTER VARYING
  - VARCHAR
  - TEXT
- Use ***single quotes*** for string quantities

Data Types - 3

- Dates
  - DATE
- Times
  - TIME
- Timestamp
  - TIMESTAMP

Data Types and SQLite

SQLite Type Affinity	Types
INTEGER	INT, INTEGER, TINYINT, SMALLINT, BIGINT, etc.
TEXT	TEXT, VARCHAR(X), CHARACTER(X), NCHAR(X),etc.
NONE	BLOB
REAL	REAL, DOUBLE, DOUBLE PRECISION, FLOAT
NUMERIC	NUMERIC, DECIMAL(X,Y), BOOLEAN, DATE, TIME

Additional details may be found at http://www.sqlite.org

Creating a Table in SQL - 1

```
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

Creating a Table in SQL - 2

```
CREATE TABLE customers
(
  uid INTEGER,
  lastname TEXT,
  firstname TEXT
);
```

Adding Rows to a Table - 1

```
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.

Adding Rows to a Table - 2

```
INSERT INTO customers (uid, lastname, firstname)
               VALUES (128, 'Smith', 'John');
INSERT INTO customers (uid, lastname, firstname)
               VALUES (324, 'Doe', 'John');


INSERT INTO customers VALUES (245, 'Jones', 'Mark');
INSERT INTO customers VALUES (756, 'Smith', 'Jane');

INSERT INTO customers (lastname, firstname, uid)
               VALUES ('Moore', 'Sara', 459);
INSERT INTO customers (lastname, firstname, uid)
               VALUES ('Parks', 'Ralph', 721);
```

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

Mapping RA to SQL

Generic SQL Query - 1

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

- `target-list` is a list of one or more ***attributes*** **A_1, A_2, ..., A_n** of a relation in the specified **`relation-list`**

Generic SQL Query - 2

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

- `relation-list` lists the ***relations*** $\mathbf{R_1, R_2, \ldots, R_m}$ that will be considered in the evaluation of the query

Generic SQL Query - 3

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

- **`predicate`** is a simple or compound logical expression for comparing one or more attribute values

Generic SQL Query - 4

- Three clauses: **`SELECT, FROM, WHERE`**

- **`SELECT`** maps to **RA** **projection****
- **`FROM`** maps to **RA** Cartesian product
- **`WHERE`** maps to **RA** selection operator

*** This can be confusing...*

Generic SQL Query - 5

```
SELECT      A1, A2, ..., An
FROM        R1, R2, ..., Rm
WHERE       P;
```

$$\Pi_{A1,\ A2,\ \ldots,\ An}\ \sigma_P(R_1\ X\ R_2\ X\ \ldots X\ R_m)$$

There are many modifiers that can be added to the above generic query

Relations

customers

UID	Last Name	First Name
128	Smith	John
324	Doe	John
245	Jones	Mark
756	Smith	Jane
459	Moore	Sara
721	Parks	Ralph

pets

UID	Pet Name	Type
128	Spot	Dog
324	Rex	Dog
756	Tiger	Cat
756	Fluffy	Cat
459	Tweety	Bird
721	Yippy	Dog
128	Rover	Dog
245	Stripes	Cat
324	Cupcake	Dog
459	Chewy	Dog

vets

UID
324
245

accounts

UID	Balance
128	0
756	45
459	0
721	10

VetClinic Example

VetClinic SQL Example - 1

.read FILENAME   Execute SQL in FILENAME

.dump Dump the database in an SQL text format

```
$ sqlite3
SQLite version 3.4.0
Enter ".help" for instructions
sqlite> .read vetclinic.txt
sqlite> .dump
BEGIN TRANSACTION;
CREATE TABLE customers
(
  uid INTEGER,
  lastname TEXT,
  firstname TEXT
);
INSERT INTO "customers" VALUES(128,'Smith','John');
INSERT INTO "customers" VALUES(245,'Jones','Mark');
INSERT INTO "customers" VALUES(324,'Doe','John');
INSERT INTO "customers" VALUES(459,'Moore','Sara');
INSERT INTO "customers" VALUES(721,'Parks','Ralph');
INSERT INTO "customers" VALUES(756,'Smith','Jane');
CREATE TABLE accounts
(
  uid INTEGER,
  balance DECIMAL
);
INSERT INTO "accounts" VALUES(128,0);
INSERT INTO "accounts" VALUES(459,0);
INSERT INTO "accounts" VALUES(721,10);
INSERT INTO "accounts" VALUES(756,45);
```

VetClinic SQL Example - 2

```
CREATE TABLE vets
(
  uid INTEGER
);
INSERT INTO "vets" VALUES(245);
INSERT INTO "vets" VALUES(324);
CREATE TABLE pets
(
  uid INTEGER,
  petname TEXT,
  type TEXT
);
INSERT INTO "pets" VALUES(128,'Spot','Dog');
INSERT INTO "pets" VALUES(324,'Rex','Dog');
INSERT INTO "pets" VALUES(756,'Tiger','Cat');
INSERT INTO "pets" VALUES(756,'Fluffy','Cat');
INSERT INTO "pets" VALUES(459,'Tweety','Bird');
INSERT INTO "pets" VALUES(721,'Yippy','Dog');
INSERT INTO "pets" VALUES(128,'Rover','Dog');
INSERT INTO "pets" VALUES(245,'Stripes','Cat');
INSERT INTO "pets" VALUES(324,'Cupcake','Dog');
INSERT INTO "pets" VALUES(459,'Chewy','Dog');
COMMIT;
sqlite>
```

VetClinic SQL Example - 3

```
sqlite> .dump accounts
BEGIN TRANSACTION;
CREATE TABLE accounts
(
  uid INTEGER,
  balance DECIMAL
);
INSERT INTO "accounts" VALUES(128,0);
INSERT INTO "accounts" VALUES(756,45);
INSERT INTO "accounts" VALUES(459,0);
INSERT INTO "accounts" VALUES(721,10);
COMMIT;
sqlite>
```

.dump ?Table? Dump the database in an SQL text format

VetClinic SQL Example - 4

Projection Operation

```
sqlite> SELECT uid FROM accounts;
128
756
459
721
sqlite> SELECT uid, balance FROM accounts;
128|0
756|45
459|0
721|10
sqlite> SELECT * FROM accounts;
128|0
756|45
459|0
721|10
sqlite> SELECT balance FROM accounts;
0
45
0
10
sqlite> SELECT DISTINCT balance FROM accounts;
0
45
10
sqlite>
```

VetClinic SQL Example - 5

Selection Operation

```
sqlite> SELECT *  FROM accounts WHERE balance>0;
756|45
721|10
sqlite> SELECT uid, balance  FROM accounts WHERE balance>0;
756|45
721|10
sqlite> SELECT uid  FROM accounts WHERE balance>0;
756
721
sqlite> SELECT petname FROM pets WHERE type='Dog';
Spot
Rex
Yippy
Rover
Cupcake
Chewy
sqlite>
sqlite> SELECT petname FROM pets WHERE type=Dog;
SQL error: no such column: Dog
sqlite>
```

VetClinic SQL Example - 6

Cartesian Product

```
sqlite> SELECT * FROM vets, accounts;
324|128|0
324|756|45
324|459|0
324|721|10
245|128|0
245|756|45
245|459|0
245|721|10
sqlite> SELECT * FROM vets, customers;
324|128|Smith|John
324|324|Doe|John
324|245|Jones|Mark
324|756|Smith|Jane
324|459|Moore|Sara
324|721|Parks|Ralph
245|128|Smith|John
245|324|Doe|John
245|245|Jones|Mark
245|756|Smith|Jane
245|459|Moore|Sara
245|721|Parks|Ralph
sqlite>
```

VetClinic SQL Example - 7

Cartesian Product

```
sqlite> SELECT * FROM accounts, customers;
128|0|128|Smith|John
128|0|324|Doe|John
128|0|245|Jones|Mark
128|0|756|Smith|Jane
128|0|459|Moore|Sara
128|0|721|Parks|Ralph
756|45|128|Smith|John
756|45|324|Doe|John
756|45|245|Jones|Mark
756|45|756|Smith|Jane
756|45|459|Moore|Sara
756|45|721|Parks|Ralph
459|0|128|Smith|John
459|0|324|Doe|John
459|0|245|Jones|Mark
459|0|756|Smith|Jane
459|0|459|Moore|Sara
459|0|721|Parks|Ralph
721|10|128|Smith|John
721|10|324|Doe|John
721|10|245|Jones|Mark
721|10|756|Smith|Jane
721|10|459|Moore|Sara
721|10|721|Parks|Ralph
sqlite>
```

VetClinic SQL Example - 8

Cartesian Product

```
sqlite> SELECT * FROM vets, customers WHERE vets.uid=customers.uid;
324|324|Doe|John
245|245|Jones|Mark
sqlite> SELECT * FROM vets, pets WHERE vets.uid = pets.uid;
324|324|Rex|Dog
324|324|Cupcake|Dog
245|245|Stripes|Cat
sqlite>
```

VetClinic SQL Example - 9

Projection, Selection, and Cartesian Product and Nested Queries

```
sqlite> SELECT petname FROM pets WHERE
   ...>  uid = (SELECT uid FROM customers WHERE lastname = 'Smith' AND
firstname = 'Jane');
Tiger
Fluffy
sqlite>
```

VetClinic SQL Example - 10

Natural Join

```
sqlite> SELECT * FROM customers NATURAL JOIN pets;
128|Smith|John|Spot|Dog
128|Smith|John|Rover|Dog
324|Doe|John|Rex|Dog
324|Doe|John|Cupcake|Dog
245|Jones|Mark|Stripes|Cat
756|Smith|Jane|Tiger|Cat
756|Smith|Jane|Fluffy|Cat
459|Moore|Sara|Tweety|Bird
459|Moore|Sara|Chewy|Dog
721|Parks|Ralph|Yippy|Dog
sqlite>
```