Lecture SQL05
Basic SQL – Part II

UAH
CPE 353

Outline

- Last Time - Review
  - Projection, Selection, Cartesian Product, and Natural Join
- More Mapping **RA** to **SQL**
  - Set operations
  - Aggregate operations
  - Integrity constraints
- Tuple Variable
- NULL Values
- Nested Subqueries

Review - Creating a Table in SQL

```
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

Review - Adding Rows to a Table

```
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.

Review - Generic SQL Query

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

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

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

Review - Generic SQL Query

- 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...*

Review - Generic SQL Query

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

$$\Pi_{A1,\ A2,\ \ldots,\ An}\ \sigma_{P}(R_1 \bowtie R_2)$$

Set Operations

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 SQL Example - 11

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>
sqlite> SELECT * FROM customers NATURAL JOIN pets WHERE type='Dog';
128|Smith|John|Spot|Dog
128|Smith|John|Rover|Dog
324|Doe|John|Rex|Dog
324|Doe|John|Cupcake|Dog
459|Moore|Sara|Chewy|Dog
721|Parks|Ralph|Yippy|Dog
sqlite>
```

VetClinic SQL Example - 12

Natural Join

```
sqlite> SELECT lastname, firstname, petname, type  FROM customers NATURAL JOIN
pets WHERE type='Dog';
Smith|John|Spot|Dog
Smith|John|Rover|Dog
Doe|John|Rex|Dog
Doe|John|Cupcake|Dog
Moore|Sara|Chewy|Dog
Parks|Ralph|Yippy|Dog
sqlite>
sqlite> SELECT lastname, firstname, petname  FROM customers NATURAL JOIN pets
WHERE type='Dog';
Smith|John|Spot
Smith|John|Rover
Doe|John|Rex
Doe|John|Cupcake
Moore|Sara|Chewy
Parks|Ralph|Yippy
sqlite>
```

VetClinic SQL Example - 13

Union

```
sqlite> SELECT petname  FROM customers NATURAL JOIN pets WHERE type='Dog'
   ...> UNION
   ...> SELECT petname  FROM customers NATURAL JOIN pets WHERE type='Cat';
Chewy
Cupcake
Fluffy
Rex
Rover
Spot
Stripes
Tiger
Yippy
sqlite>
sqlite> SELECT * FROM customers WHERE lastname='Smith' UNION SELECT * FROM
customers WHERE firstname='John';
128|Smith|John
324|Doe|John
756|Smith|Jane
sqlite>
```

VetClinic SQL Example - 14

Intersection

```
sqlite> SELECT petname  FROM customers NATURAL JOIN pets WHERE type='Dog'
   ...> INTERSECT
   ...> SELECT petname  FROM customers NATURAL JOIN pets WHERE type='Cat';
sqlite>
sqlite>
sqlite> SELECT * FROM customers WHERE lastname='Smith' INTERSECT SELECT * FROM
customers WHERE firstname='John';
128|Smith|John
sqlite>
```

VetClinic SQL Example - 15

Difference

```
qlite> SELECT * FROM customers WHERE lastname='Smith';
128|Smith|John
756|Smith|Jane
sqlite> SELECT * FROM customers WHERE firstname='John';
128|Smith|John
324|Doe|John
sqlite>
sqlite> SELECT * FROM customers WHERE lastname='Smith' EXCEPT SELECT * FROM
customers WHERE firstname='John';
756|Smith|Jane
sqlite>
sqlite>
```

R_1 EXCEPT R_2 ==> Rows from R_1 that do not appear in R_2

VetClinic SQL Example - 16

Delete Row

```
sqlite> SELECT * FROM customers;
128|Smith|John
324|Doe|John
245|Jones|Mark
756|Smith|Jane
459|Moore|Sara
721|Parks|Ralph
sqlite> SELECT * FROM customers WHERE lastname='Smith';
128|Smith|John
756|Smith|Jane
sqlite> DELETE FROM customers WHERE lastname='Smith' and firstname='Jane';
sqlite> SELECT * FROM customers;
128|Smith|John
324|Doe|John
245|Jones|Mark
459|Moore|Sara
721|Parks|Ralph
sqlite>
```

VetClinic SQL Example - 17

Update Row

```
sqlite> SELECT * FROM customers;
128|Smith|John
324|Doe|John
245|Jones|Mark
756|Smith|Jane
459|Moore|Sara
721|Parks|Ralph
sqlite> UPDATE customers SET lastname='Simpson' WHERE lastname='Doe';
sqlite> .dump
BEGIN TRANSACTION;
CREATE TABLE customers (uid INTEGER, lastname TEXT, firstname TEXT);
INSERT INTO "customers" VALUES(128,'Smith','John');
INSERT INTO "customers" VALUES(324,'Simpson','John');
INSERT INTO "customers" VALUES(245,'Jones','Mark');
INSERT INTO "customers" VALUES(756,'Smith','Jane');
INSERT INTO "customers" VALUES(459,'Moore','Sara');
INSERT INTO "customers" VALUES(721,'Parks','Ralph');
COMMIT;
sqlite>
```

Aggregate Operations

Aggregate Operations

- Produces a single value result not a relation
- Cannot mix non-aggregate row-by-row and aggregate expressions in a SELECT clause
- Cannot nest aggregate operations
- Cannot express these operations in terms of Relational Algebra

VetClinic SQL Example - 18

Count Rows

```
sqlite> SELECT * FROM pets WHERE type='Dog';
128|Spot|Dog
324|Rex|Dog
721|Yippy|Dog
128|Rover|Dog
324|Cupcake|Dog
459|Chewy|Dog
sqlite> SELECT COUNT(*) FROM pets WHERE type='Dog';
6
sqlite>
```

Returns a non-negative number

VetClinic SQL Example - 19

Count Rows

```
sqlite> SELECT uid FROM pets;
128
324
756
756
459
721
128
245
324
459
sqlite>
sqlite> SELECT COUNT(uid) FROM pets;
10
sqlite>
sqlite> SELECT COUNT(DISTINCT uid) FROM pets;
6
sqlite>
```

Returns a non-negative number

VetClinic SQL Example - 20

Sum, Average

```
sqlite> SELECT balance FROM accounts;
0
45
0
10
sqlite> SELECT SUM(balance) FROM accounts;
55
sqlite>
sqlite> SELECT AVG(balance) FROM accounts;
13.75
sqlite>
sqlite> SELECT AVG(DISTINCT balance) FROM accounts;
18.3333333333333
sqlite>
```

VetClinic SQL Example - 21

Min, Max

```
sqlite> SELECT balance FROM accounts;
0
45
0
10
sqlite> SELECT MIN(balance) FROM accounts;
0
sqlite> SELECT MAX(balance) FROM accounts;
45
sqlite>
sqlite> SELECT MIN(balance), MAX(balance) FROM accounts;
0|45
sqlite>
```

Use with numbers, characters, and data-time

VetClinic SQL Example - 22

Min, Max

```
sqlite> SELECT MIN(lastname), MAX(lastname) FROM customers;
Doe|Smith
sqlite> SELECT MIN(firstname), MAX(firstname) FROM customers;
Jane|Sara
sqlite>
```

Use with numbers, characters, and data-time

Integrity Constraints

VetClinic SQL Example - 23

Attribute Check

```
sqlite>
sqlite> CREATE TABLE dummy
   ...> (
   ...>   uid INTEGER CHECK( uid > 0 ),
   ...>   name TEXT
   ...> );
sqlite> INSERT INTO dummy VALUES(123, 'HORSE');
sqlite> INSERT INTO dummy VALUES(-1, 'COW');
SQL error: constraint failed
sqlite>
```

Constrain the value of an attribute

VetClinic SQL Example - 24

Tuple Check

```
sqlite> CREATE TABLE dummy
   ...> (
   ...>   uid INTEGER,
   ...>   studentname TEXT,
   ...>   previousgrade INTEGER,
   ...>   currentgrade INTEGER,
   ...>   CHECK(currentgrade >= previousgrade)
   ...> );
sqlite> INSERT INTO dummy VALUES(123, 'Homer Simpson', 10, 11);
sqlite> INSERT INTO dummy VALUES(234, 'Bart Simpson', 7, 6);
SQL error: constraint failed
sqlite>
```

Constrain multiple attributes

Tuple Variables

VetClinic SQL Example - 25

Tuple Variables

```
sqlite> SELECT * FROM vets AS V, customers AS C WHERE V.uid=C.uid;
324|324|Doe|John
245|245|Jones|Mark
sqlite>
```

NULL Values

VetClinic SQL Example - 26

NULL Values

```
$ sqlite3
SQLite version 3.4.0
Enter ".help" for instructions
sqlite> CREATE TABLE dummy
   ...> ( uid INTEGER, name TEXT );
sqlite> INSERT INTO dummy VALUES(123, 'Homer');
sqlite> INSERT INTO dummy VALUES(456, 'Marge');
sqlite> INSERT INTO dummy VALUES(NULL, 'Bart');
sqlite> SELECT * FROM dummy;
123|Homer
456|Marge
|Bart
sqlite> SELECT SUM(uid) FROM dummy;
579
sqlite>
sqlite> SELECT COUNT(*) FROM dummy;
3
sqlite> SELECT COUNT(uid) FROM dummy;
2
sqlite>
```

Aggregate operators other than **COUNT(*)** ignore NULL values

VetClinic SQL Example - 27

NULL Values

```
sqlite> SELECT * FROM dummy WHERE uid IS NULL;
|Bart
sqlite> SELECT * FROM dummy WHERE uid IS NOT NULL;
123|Homer
456|Marge
sqlite>
```

Aggregate operators other than **COUNT(*)** ignore NULL values

Integrity Constraints and NULL Values

```
sqlite> CREATE TABLE dummy
   ...> (
   ...>   uid INTEGER,
   ...>   name TEXT,
   ...>   CHECK( uid IS NOT NULL AND uid > 0)
   ...> );
sqlite> INSERT INTO dummy VALUES(123, 'Homer Simpson');
sqlite> INSERT INTO dummy VALUES(000, 'Bart Simpson');
SQL error: constraint failed
sqlite> INSERT INTO dummy VALUES(NULL, 'Marge Simpson');
SQL error: constraint failed
sqlite> .dump
BEGIN TRANSACTION;
CREATE TABLE dummy
(
  uid INTEGER,
  name TEXT,
  CHECK( uid IS NOT NULL AND uid > 0)
);
INSERT INTO "dummy" VALUES(123,'Homer Simpson');
COMMIT;
sqlite>
```

Nested Subqueries

VetClinic SQL Example - 28

Nested Queries

```
sqlite> SELECT * FROM customers WHERE uid IN
   ...>     (SELECT uid FROM accounts WHERE balance>0);
756|Smith|Jane
721|Parks|Ralph
sqlite>
sqlite> SELECT * FROM customers WHERE uid NOT IN
   ...>     (SELECT uid FROM accounts WHERE balance>0);
128|Smith|John
324|Doe|John
245|Jones|Mark
459|Moore|Sara
sqlite>
sqlite> SELECT * FROM customers WHERE lastname IN ('Jones', 'Parks');
245|Jones|Mark
721|Parks|Ralph
sqlite>
```