DS 1300 - Introduction to SQL
Part 3 – Aggregation & other Topics

by Michael Hahsler
Based on slides for CS145 Introduction to Databases (Stanford)

DILBERT
BY SCOTT ADAMS
I ACCOMPLISHED NOTHING THIS WEEK BECAUSE I WAS IN A TRAINING CLASS.
DilbertCartoonist@gmail.com
I DIDN'T APPROVE ANY TRAINING EXPENSES.
A VENDOR PAID FOR IT.
YOU DIDN'T ASK FOR PERMISSION.
I'M PRO-ACTIVE AND EMPOWERED.
© 2013 Scott Adams, Inc. /Dist. by Universal Uclick
AND WHAT WAS THE NAME OF THIS ALLEGED CLASS?
ADVANCED SCRIPTING STRUCTURE FOR INTERNETWORK OPTIMIZATION OF SQL DATABASES.
THAT DOESN'T SOUND REAL.
I CAN'T DO MY JOB IF YOU DON'T TRUST ME!
DO YOU LIKE HOW I COMBINED AGGRESSIVENESS WITH MY BASELINE LEVEL OF USELESSNESS?
www.dilbert.com
7-28-13
I HAVE A GOOD FEELING ABOUT THIS.
YOU MIGHT NEED MORE AGGRESSIVENESS.

Lecture Overview

1. Aggregation & GROUP BY

2. Advanced SQL-izing (set operations, NULL, Outer Joins, etc.)

AGGREGATION, GROUP BY AND HAVING CLAUSE

Aggregation

```
SELECT COUNT(*)
FROM   Product
WHERE  year > 1995
```

```
SELECT AVG(price)
FROM   Product
WHERE  maker = ‘Toyota’
```

- SQL supports several **aggregation** operations:
  - SUM, COUNT, MIN, MAX, AVG

Except for COUNT, all aggregations apply to a single attribute!

Aggregation: COUNT

COUNT counts the number of tuples including duplicates.

```
SELECT COUNT(category)
FROM   Product
WHERE  year > 1995
```

Note: Same as COUNT(*)!

We probably want count the number of “different” categories:

```
SELECT COUNT(DISTINCT category)
FROM   Product
WHERE  year > 1995
```

More Examples

Purchase(product, date, price, quantity)

```
SELECT SUM(price * quantity)
FROM   Purchase
```

What do these mean?

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = ‘bagel’
```

Simple Aggregations

Purchase

Product	Date	Price	Quantity
bagel	10/21	1	20
banana	10/3	0.5	10
banana	10/10	1	10
bagel	10/25	1.50	20

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = ‘bagel’
```

50 (= 1*20 + 1.50*20)

Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after Oct 1, 2010, per product.

```
SELECT   product,
         SUM(price * quantity) AS TotalSales
FROM     Purchase
WHERE    date > ‘2000-10-01’
GROUP BY product
```

Note: Be very careful with dates! Use date/time related functions!

Let’s see what this means…

Grouping and Aggregation

Semantics of the query:

1. Compute the FROM and WHERE clauses

2. Group by the attributes in the GROUP BY

3. Compute the SELECT clause: grouped attributes and aggregates

1. Compute the FROM and WHERE clauses

```
SELECT   product, SUM(price*quantity) AS
TotalSales
FROM     Purchase
WHERE    date > ‘2000-10-01’
GROUP BY product
```

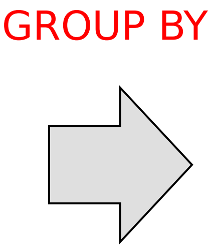
FROM

Product	Date	Price	Quantity
Bagel	2000-10-21	1	20
Bagel	2000-10-25	1.50	20
Banana	2000-10-03	0.5	10
Banana	2000-10-10	1	10

2. Group by the attributes in the GROUP BY

```
SELECT   product, SUM(price*quantity) AS
TotalSales
FROM     Purchase
WHERE    date > ‘2000-10-01’
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	2000-10-21	1	20
Bagel	2000-10-25	1.50	20
Banana	2000-10-03	0.5	10
Banana	2000-10-10	1	10



Product	Date	Price	Quantity
Bagel	2000-10-21	1	20
	2000-10-25	1.50	20
Banana	2000-10-03	0.5	10
	2000-10-10	1	1

3. Compute the SELECT clause: grouped attributes and aggregates

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > ‘2000-10-01’
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	2000-10-21	1	20
	2000-10-25	1.50	20
Banana	2000-10-03	0.5	10
	2000-10-10	1	10

SELECT

Product	TotalSales
Bagel	50
Banana	15

Activity

Company(Cname, country)
Product(PName, price, category, manufacturer)
Purchase(id, product, buyer)

1) What do the next two queries calculate?

```
SELECT SUM(price) AS total, SUM(price) *1.08 AS totalPlusTax
   FROM Product pr
   JOIN Purchase p ON pr.PName = p.product
   WHERE  p.buyer = 'Joe Blow'
```

```
SELECT p.buyer, SUM(price) AS total, SUM(price) *1.08 AS totalPlusTax
   FROM Product pr
   JOIN Purchase p ON pr.PName = p.product
   GROUP BY p.buyer
   ORDER BY 1
```

2) Write a query to find the price of the most expensive product in each category.

HAVING Clause

Purchase(product, date, price, quantity)

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > ‘2005-10-01’
GROUP BY product
HAVING   SUM(quantity) > 100
```

Same query as before, except that we consider only products that have more than 100 buyers

HAVING clauses contains conditions on **aggregates**

*Whereas WHERE clauses condition on **individual tuples…***

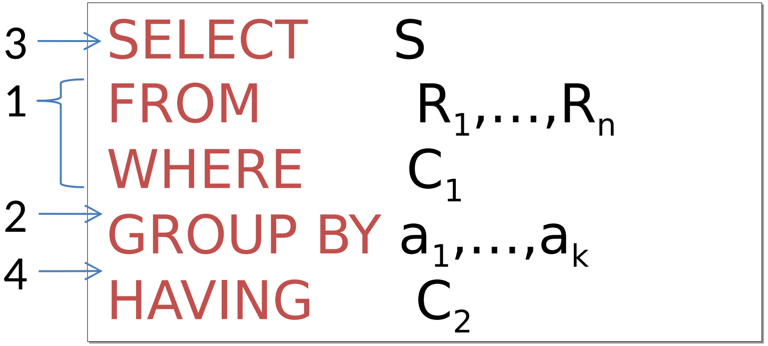
General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C_1
GROUP BY $a_1,\ldots,a_k$
HAVING C_2

- S = Can ONLY contain attributes $a_1,\ldots,a_k$ and/or aggregates over other attributes
- C_1 = is any condition on the attributes in $R_1,\ldots,R_n$
- C_2 = is any condition on the aggregate expressions

Why?

General form of Grouping and Aggregation



3 SELECT S
1 FROM $R_1,\ldots,R_n$
WHERE C_1
2 GROUP BY $a_1,\ldots,a_k$
4 HAVING C_2

Evaluation steps:

1. Evaluate FROM-WHERE: apply condition C_1 on the attributes in $R_1,\ldots,R_n$
2. GROUP BY the attributes $a_1,\ldots,a_k$
3. Compute aggregates in S and do projection (SELECT)
4. Apply condition C_2 to each group (may have aggregates)

Activity

Company(Cname, country)
Product(PName, price, manufacturer)
Purchase(id, product, buyer)

1) What does this query do?

```
SELECT p.buyer, SUM(price) AS total, COUNT(*) AS purchases
FROM Product pr
JOIN Purchase p ON pr.PName = p.product
GROUP BY p.buyer
HAVING purchases >2
ORDER BY 1
```

2) What products in the DB have a revenue of more then $10,000?

OTHER SQL TOPICS: SUBQUERIES, NULLS, CASTING, OUTER JOINS AND ADDING DATA

Subqueries

```
SELECT *
  FROM (SELECT product, COUNT(product) AS count
     FROM Purchase GROUP BY product)
  WHERE count > 2
```

```
SELECT *,  (SELECT count(*) FROM Product p1
     WHERE p1.category = p2.category) AS '# Prod. in Cat.'
   FROM Product p2
```

Subqueries can appear wherever a table or a value is needed.

NULL VALUES & OTHER DETAILS

NULL Values

- Whenever we do not have a value, we can use NULL

- Can mean many things:
  - Value does not exists
  - Value exists but is unknown (n/a, not available)
  - Value not applicable

- The schema specifies for each attribute if it can be null (*nullable* attribute) or not with NOT NULL

NULL Values and Operators

For numerical operations:

– If x = NULL then 4*(3-x)/7 is also NULL

For boolean operations, in SQL there are three values:

FALSE	**=**	**0**
TRUE	**=**	**1**
UNKNOWN		

If x= NULL then x='Joe' is UNKNOWN

Note: comparison in SQL is a single '='

SQLite does not have a boolean datatype. It uses Integer instead!
Try:

- SELECT 2>1
- SELECT 2>NULL
- SELECT 1+NULL

Null Values in the WHERE Clause

```
SELECT *
FROM   Person
WHERE (age < 25)
  AND (height > 6 AND weight > 190)
```

Will not return age=20, height=NULL, weight=200

Since NULL > 6 is UNKNOWN!

NULL Values in WHERE Clauses

Unexpected behavior:

```
SELECT *
FROM   Person
WHERE  age < 25 OR age >= 25
```

Should return all persons, but
persons with NULL as age are not included!

You can use CASE with IS NULL, ISNULL(), IFNULL() or COALESCE() to handle NULL values.

CASTing Data Types

SQL is a typed language. I.e., values and columns have a data type.

```
SELECT 3/2
SELECT 3.0/2
SELECT 3/2.0
SELECT CAST(3 AS DOUBLE)/2
```

```
1
1.5
1.5
1.5
```

Typecasting rules are similar to other typed languages like C++.

RECAP: Inner Joins

Inner joins select all rows from both tables as long as there is a match between the columns in both tables. Inner joins are the default in SQL.

Example: What stores sell what products?

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
  JOIN Purchase ON Product.name =
                 Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

Both equivalent:
Both INNER JOINS!

Inner Joins + NULLS = Lost data?

```
Product(name, category)
Purchase(prodName, store)
```

```
SELECT Product.name, Purchase.store
FROM   Product
  JOIN Purchase ON Product.name =
                  Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

However: Products that were never sold in any store (with no Purchase tuple) will be lost!

Outer Joins

An **outer join** returns also tuples from the joined relations that do not have a corresponding tuple in the other relations (filled with NULL values).

Left outer joins in SQL:

```
SELECT Product.name, Purchase.store
FROM   Product
  LEFT OUTER JOIN Purchase ON
      Product.name = Purchase.prodName
```

Now we'll get products even if they didn't sell

INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM   Product
  INNER JOIN Purchase
    ON Product.name = Purchase.prodName
```

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

LEFT OUTER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

```
SELECT Product.name, Purchase.store
FROM   Product
  LEFT OUTER JOIN Purchase
    ON Product.name = Purchase.prodName
```

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Other Outer Joins

- Left outer join:
  - Include the left tuple even if there’s no match
- Right outer join:
  - Include the right tuple even if there’s no match
- Full outer join:
  - Include the both left and right tuples even if there’s no match

SQLite currently only supports LEFT OUTER JOIN, but you can easily just change the order of the tables in the query.

Adding Data

```
INSERT INTO TABLE_NAME
    [(column1, column2, column3,...columnN)]
VALUES (value1, value2, value3,...valueN);
```

Note: column names are optional.

```
INSERT INTO Product
VALUES ('Gizmo', 19, 'Gadgets', 'GWorks')
```

Adding Data

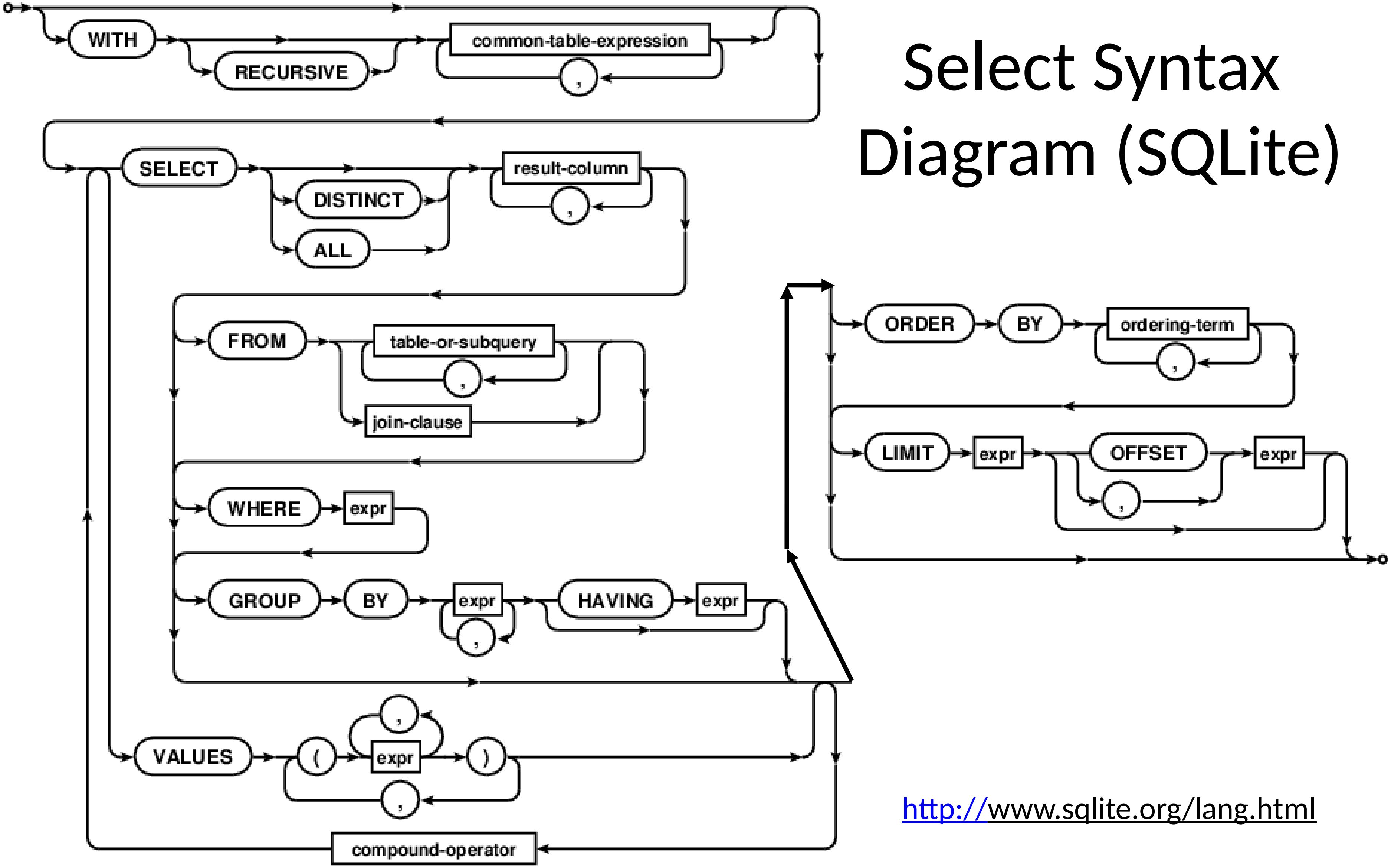
The data can also come from an existing table.

```
INSERT INTO first_table_name [(column1, column2, ... columnN)]
   SELECT column1, column2, ...columnN
   FROM second_table_name
   [WHERE condition];
```

Removing a Table

```
DROP TABLE database_name.table_name
```

Select Syntax Diagram (SQLite)



http://www.sqlite.org/lang.html

Activity

Review (http://www.tutorialspoint.com/sqlite/):

- Transaction control
- Views
- Indexes
- Date & Time