SQL Workshop Session 2

Thursday 18 July 2013 General Assembly

Roadmap

- Where are we
- Review
 - Homework and material
- Conceptual path of query writing
- Concepts of database organization
- Joins
- Subqueries
- More detailed conditions

So Far:

- We have tables, we write queries, we get another table back
- Oriented to basics as we showed that
 - We can do more with WHERE, GROUP BY, ORDER
 BY
 - Important, useful, boring

Next: Combine Tables, More Complicated Analysis

- JOIN one table with another
 - E.g., match records from one table with records from another
- Nest queries with one another
 - Use "subqueries" to generate values used in queries, or to run particular queries for each record in the returned table

Environment Stuff

- Learn about the databases and tables available to use
- Create new databases and tables
- Load new data
- Revise existing tables, remove or revise data
- Crucial, but boring

Deeper manipulations

- More about
 - Operators (greater than, less than, IN, etc)
 - Logical expressions (AND, OR)
 - Using functions to filter, group, reduce data
- Available functions

Exercise Review

Joins

A basic query:

```
SELECT
student_id AS Student,
AVG(score) AS Average,
COUNT(score) AS "# of Tests"
FROM
sampdb.score
GROUP BY
student_id
ORDER BY
Average DESC
```

 Returns each student's average

	Student	Average	# of Tests	
▶	1	48.0000	5	
	27	45.7500	4	
	5	42.8333	6	
	18	41.3333	6	
	17	41.2000	5	
	2	40.4000	5	
	11	39.8333	6	

 But can we get the student's name rather than their id?

Yes

```
SELECT
  st.name AS
                 Name,
  scr.student_id AS Id,
  AVG(scr.score) AS Average,
  COUNT(scr.score) AS "# Tests"
FROM
  sampdb.score scr
INNER JOIN
  sampdb.student st
ON
  scr.student_id = st.student_id
GROUP BY
  scr.student_id
ORDER BY
  Average DESC
```

	Name	Id	Average	# Tests
\triangleright	Megan	1	48.0000	5
	Carter	27	45.7500	4
	Abby	5	42.8333	6
	Max	18	41.3333	6
	Will	17	41.2000	5
	Incenh	2	40 4000	5

Breaking down the query

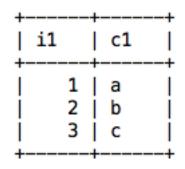
```
SELECT
                Name.
 st.name AS
 scr.student id AS Id,
 AVG(scr.score) AS Average,
 COUNT(scr.score) AS "# Tests"
FROM
 sampdb.score scr
INNER JOIN
  sampdb.student st
ON
  scr.student id = st.student id
GROUP BY
 scr.student id
ORDER BY
 Average DESC
```

- A join combines rows from different tables
- The "ON" clause specifies conditions for combining records
 - Here, records with the same student_id value are combined into a single record
 - The SELECT clause still specifies which fields are displayed from that combined record
- INNER JOIN specifies which table to join data from
 - The addition of "scr" and "st" in the specification of tables provides alias values for reference by the other clauses
 - Without these the query wouldn't know which table it should find a field in

Break Down Joins Still More

Two Really Simple Tables

• T1



• t2

<u>-</u>		+	+
i2		c2	I
!	2	+ c	+ !
		b a	
+		+	+

Inner Join: All Rows Matched

- A simple inner join matches each row in one table with each row in the other
- SELECT * FROM join_sample.t1 INNER JOIN join_sample.t2;

- So joining a 3 row table with a 3 row table produces a 9 row table
- Obviously we don't want all those rows

i1	c1	i2	c2
1	a	2	c
j 2	b	2	c
3	C	2	c
1	a	3	b
2	b	3	b
3	C	3	b
1	a	4	a
2	b	4	a
3	C	4	a
+	+	+	++

Limiting Inner Join Results

```
i1
                                                c1
                                                       i2
                                                            | c2
SELECT
FROM
  join_sample.t1
INNER JOIN
  join_sample.t2
WHERE
  join_sample.t1.i1 = join_sample.t2.i2
                                           i1
                                                 c1
                                                        i2
                                                              c2
```

Left Join

 Get all rows from the "left" table, each with that row from the "right" table that matches on the specified fields

```
SELECT
  *
FROM
  join_sample.t1
LEFT JOIN
  join_sample.t2
  ON
    join_sample.t1.i1 = join_sample.t2.i2
;
```

i1	c1	
1 2	a b	NULL

Subqueries

Simple but clumsy

```
SELECT au id, city
 FROM books.authors
 WHERE city IN
    'New York',
    'San Francisco',
    'Hamburg',
    'Berkeley'
  );
```

- Uses the WHERE field IN [list] pattern that we have seen before
- Works, but must type four city names
 - List might be longer
 - List might be wrong
 - List might change
- How did we get those names in the first place?

Databases produce lists

SELECT city FROM books.publishers;

city New York San Francisco Hamburg Berkeley Now we have the right list, but we still have to type it.

Better to subquery

SELECT au_id, city

FROM books.authors

WHERE city IN

(SELECT city FROM books.publishers);

- We let the database figure out the list
- If the data supporting the list changes – the query still works
- And we can expand the logic used to specify the list . . .

SELECT au_id, au_fname, au_Iname, state

FROM authors

WHERE state IN

(SELECT state

FROM authors

WHERE au_id = 'A04');

 Now we can introduce logic into our specification of the WHERE clause list

Subqueries inject query results into queries

SELECT au_id, city

FROM books.authors

WHERE city IN

(SELECT city FROM books.publishers);

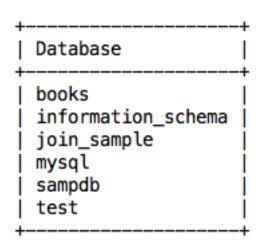
Exploring Data Available

Exploring databases and tables

SHOW DATABASES

SHOW TABLES IN books

Dull but important





Finding fields

DESCRIBE books.author

- 'Type' tells the database how to handle data in a field
 - For example, numbers, strings and dates all sort in different ways
 - Null, Key and Default set controls on values entered into the table

Field	Туре	Null	Key	Default	Extra
au_id au_fname au_lname phone address city state zip	char(3) varchar(15) varchar(15) varchar(12) varchar(20) varchar(15) char(2)	NO NO NO YES YES YES YES	PRI	NULL NULL NULL NULL NULL NULL NULL NULL	

Setting and Changing These

- We will look at how to creating and loading these things
 - These values are set at table creation
 - They can be altered, but it takes some trouble