DataPhilly Intro to SQL Workshop

October 15, 2020

Kayleigh Smoot

Link to Online Worksheet

https://www.jdoodle.com/ia/2QC

About the Instructor

Kayleigh Smoot

Kayleigh.a.smoot@gmail.com

https://www.linkedin.com/in/kayleigh-smoot/

What Will You Get Out of This Workshop?

A good understanding of basic SQL concepts

Ability to run different kinds of queries

What is SQL?

Structured Query Language

- Some people pronounce it “sequel”
- Others insist that “ess-cue-ell” is the only correct pronunciation

A language for accessing and updating databases

- Remains on Top 10 Tech Skills-in-Demand lists

Here is the First Table in Our "Database"

"Doctor" table

d_id	d_firstname	d_lastname	d_title	specialty
1	Bob	Samson	MD	General Internal
2	Ramesh	Hopkins	MD	General Internal
3	Jane	Adams	DO	Pediatrics
4	Sugirtha	Samson	MD	Toe Surgery

- **"Doctor" is the name of the table**
- **Each *row* is a record (a doctor)**
- **Each *value* in a *row* tells us something about the record (doctor)**
- **Each *value* in a *column* contains the same kind of information**

Tables in Our "Database"

"Doctor" table

d_id	d_firstname	d_lastname	d_title	specialty
1	Bob	Samson	MD	General Internal
2	Ramesh	Hopkins	MD	General Internal
3	Jane	Adams	DO	Pediatrics
4	Sugirtha	Samson	MD	Toe Surgery

"Patient" table

p_id	p_firstname	p_lastname	bdate
1	Mark	Jones	01/15/1990
2	Tom	Roberts	02/27/1985
3	Sue	Steinberg	03/07/1981
4	Becky	Jones	04/12/1985

"Appointment" table

a_id	a_date	p_id	d_id
1	02/15/2014 10:00:00	5	1
2	02/15/2014 10:00:00	3	2
3	02/15/2014 10:00:00	4	3
4	02/15/2014 11:00:00	2	1

What Does SQL Look Like?

- SQL contains two kinds of “languages” (statement types)
  - DDL is the Data Definition Language; it defines the *structure* of tables
    - CREATE TABLE -- creates a new database table
    - ALTER TABLE -- alters (changes) a database table
    - DROP TABLE -- deletes a database table
  - DML is the Data Manipulation Language; it defines and manipulates the *content* of tables
    - INSERT -- puts new data into the database
    - SELECT -- gets data from the database
    - UPDATE -- updates (changes) data in the database
    - DELETE -- removes data from the database

What Does SQL Look Like?

- SQL contains two kinds of “languages” (statement types)
  - DDL is the Data Definition Language; it defines the *structure* of tables
    - CREATE TABLE -- creates a new database table
    - ALTER TABLE -- alters (changes) a database table
    - DROP TABLE -- deletes a database table
  - DML is the Data Manipulation Language; it defines and manipulates the *content* of tables
    - INSERT -- puts new data into the database
    - SELECT -- gets data from the database
    - UPDATE -- updates (changes) data in the database
    - DELETE -- removes data from the database

SELECT statements

SELECT

- Syntax:
  SELECT *columns* FROM *table*;
  - *columns* is:

    a comma-separated list of column names, or

    * to indicate “all columns”
  - *table* is the name of the table

SELECT

- Syntax:
  SELECT *columns* FROM *table*;
  - *columns* is:

    a comma-separated list of column names, or

    * to indicate "all columns"

    Note: order of columns does not matter
  - *table* is the name of the table
- Let's take a look at the first name and last name of every doctor in the database

```
SELECT d_firstname, d_lastname FROM Doctor;
```

SELECT (practice)

Pull up the id, last name, first name, and birth date of every patient.

SELECT (practice)

Pull up the id, last name, first name, and birth date of every patient.

```
SELECT p_id, p_lastname, p_firstname, bdate FROM Patient;
```

SELECT – Eliminating Duplication in Output

- Syntax:
  SELECT DISTINCT *column* FROM *table*;
  - DISTINCT keyword returns only distinct (or different) values for a column

SELECT – Eliminating Duplication in Output

- Syntax:
  SELECT DISTINCT *column* FROM *table*;
  - DISTINCT keyword returns only distinct (or different) values for a column

  First, select *all* doctor specialties from the Doctor table
  SELECT specialty FROM Doctor; #Shows the specialty for every doctor, regardless of duplicates

SELECT – Eliminating Duplication in Output

- What are the *different* specialties a doctor can have?

```
SELECT DISTINCT specialty FROM Doctor;
#Shows the different specialties for every doctor, not including duplicates
```

SELECT – Eliminating Duplication in Output (practice)

Pull a list of the *unique* last names of all patients.

SELECT – Eliminating Duplication in Output (practice)

Pull a list of the *unique* last names of all patients.

```
SELECT DISTINCT p_lastname FROM Patient;
```

SELECT – Limiting Rows

- Syntax:
  SELECT * FROM *table* LIMIT [*index*], *number*;
  - *number* is a number indicating how many rows to return
  - *index* is:
    - optional
    - the number of rows to skip

SELECT – Limiting Rows

- Syntax:
  SELECT * FROM *table* LIMIT [*index*], *number*;
  - *number* is a number indicating how many rows to return
  - *index* is:
    - optional
- Let's query for the id and date values for the first 3 rows in the Appointment table
  SELECT a_id, a_date FROM Appointment LIMIT 3;
- Select 3 patients, starting on row 3
  SELECT * FROM Patient LIMIT 2, 3;

SELECT – Summary

Command Desc.	Basic Syntax Structure	Example
SELECT Views all columns of a table	SELECT * FROM *table*;	SELECT * FROM Doctor;
SELECT Views one column of a table	SELECT *column* FROM *table*;	SELECT d_firstname FROM Doctor;
SELECT Views multiple columns of a table	SELECT *column, column, …* FROM *table*;	SELECT d_firstname, d_lastname FROM Doctor;
AS Assigns an alias to a column during display	SELECT *column* [AS] *alias* FROM *table*;	SELECT d_firstname as 'First Name' FROM Doctor;
DISTINCT Eliminates duplication in output	SELECT DISTINCT *column* FROM *table*;	SELECT DISTINCT specialty FROM Doctor;
LIMIT Limits the total rows in the output	SELECT * |*column* FROM *table* LIMIT #;	SELECT * FROM Doctor LIMIT 10;

Filtering Records & Sorting Data

SELECT – Using WHERE Clause

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition*;
  - *condition* is a condition to be satisfied

SELECT – Using WHERE Clause

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition*;
  - *condition* is a condition to be satisfied
- Let's take a look at the first name and last name of every doctor where the doctor id is less than 3
- SELECT d_firstname, d_lastname FROM Doctor WHERE d_id < 3;

SELECT – Using WHERE Clause (practice)

Find all patients with the last name “Jones”.

SELECT – Using WHERE Clause (practice)

Find all patients with the last name “Jones”.

```
SELECT * FROM Patient
 WHERE p_lastname = 'Jones’;
```

SELECT – Using WHERE Clause

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition*;
  - *condition* is a condition to be satisfied
- Find the doctors who *do not* have the title “MD”
  SELECT * FROM Doctor WHERE d_title != ‘MD’;

SELECT – Common Comparison Operators

=	Equals
!=	Not equals (<> can also be used)
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

SELECT – Using AND Operator with Multiple Conditions

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition1* AND *condition2*;
  - *condition1* and *condition2* are conditions to be satisfied
    - *both* must be true to satisfy WHERE clause

SELECT – Using AND Operator with Multiple Conditions

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition1* AND *condition2*;
  - *condition1* and *condition2* are conditions to be satisfied
    - *both* must be true to satisfy WHERE clause
- Let's see all of the patients with the first name "Becky" *and* the last name "Jones"

```
SELECT * FROM Patient
 WHERE p_firstname = 'Becky'
 AND p_lastname = 'Jones';
```

SELECT – Using OR Operator with Multiple Conditions

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition1* OR *condition2*;
  - *condition1* and *condition2* are conditions to be satisfied
  - *either* can be true to satisfy WHERE clause

SELECT – Using OR Operator with Multiple Conditions

- Syntax:
  SELECT *columns* FROM *table* WHERE *condition1* OR *condition2*;
  - *condition1* and *condition2* are conditions to be satisfied
  - *either* can be true to satisfy WHERE clause
- Let's see all of the patients with the last name "Jones" or "Roberts"

```
SELECT * FROM Patient
  WHERE p_lastname = 'Jones'
  OR p_lastname ='Roberts';
```

SELECT – Using IN Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] IN (*value1*, *value2*);
  - *column* is a column name
  - *value1* and *value2* are multiple values to compare *column* with
  - NOT is optional

SELECT – Using IN Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] IN (*value1*, *value2*);
  - *column* is a column name
  - *value1* and *value2* are multiple values to compare *column* with
  - NOT is optional
- Let's see all of the patients with the last name "Jones" or "Roberts"
  SELECT * FROM Patient WHERE p_lastname IN ('Jones', 'Roberts');

SELECT – Using BETWEEN…AND Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] BETWEEN *value1* AND *value2*;
  - *value1* and *value2* are values in a range (can be numbers, text, or dates)
  - NOT is optional

SELECT – Using BETWEEN…AND Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] BETWEEN *value1* AND *value2*;
  - *value1* and *value2* are values in a range (can be numbers, text, or dates)
  - NOT is optional
- List the doctor's appointments where the id is either 2,3,4, or 5
  SELECT * FROM Appointment
  WHERE a_id BETWEEN 2 AND 5;
  Note: the range values are included in the result

SELECT – Using BETWEEN…AND Operator (practice)

List all appointments that took place on 02/15/2016 between 10am and 11am.

Note: use the MySQL datetime format when comparing dates or times

MySQL comes with different data types for storing a date or a date/time value:

DATE -- format: YYYY-MM-DD

DATETIME -- format: YYYY-MM-DD HH:MI:SS

SELECT – Using BETWEEN…AND Operator (practice)

List all appointments taking place on 02/15/2016 between 10am and 11am

```
SELECT * FROM Appointment
WHERE a_date BETWEEN '2016-02-15 10:00:00' AND '2016-02-15 11:00:00';
```

SELECT – Using LIKE Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] LIKE *pattern*;
  - *pattern* is:
    - a specified pattern to search for
    - Note: LIKE is not case sensitive
  - NOT is optional

SELECT – Using LIKE Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] LIKE *pattern*;
  - *pattern* is:
    - a specified pattern to search for
    - Note: LIKE is not case sensitive
  - NOT is optional
- Search for patients with first names that begin with the letter “J”
  - SELECT * FROM Patient WHERE p_firstname LIKE 'J%’;
  - % is a wildcard character matching zero or more characters in the pattern

SELECT – Using LIKE Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* [NOT] LIKE *pattern*;
  - *pattern* is:
    - a specified pattern to search for
    - Note: LIKE is not case sensitive
  - NOT is optional
- Search for doctors with last names that begin with “Sa” and end with “an”
- SELECT * FROM Doctor WHERE d_lastname LIKE ’Sa_an’;
- _ means you can substitute any number of individual character(s)

SELECT – Using IS NULL Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* IS [NOT] NULL;
  - IS NULL searches for records where *column* contains a NULL (unknown) value
  - NOT is optional

SELECT – Using IS NULL Operator

- Syntax:
  SELECT *columns* FROM *table* WHERE *column* IS [NOT] NULL;
  - IS NULL searches for records where *column* contains a NULL (unknown) value
  - NOT is optional

- Find the doctors with an unknown title
  SELECT * FROM Doctor WHERE d_title IS NULL;

  - #Shows doctors where title is NULL

- Compare with this:
  SELECT * FROM Doctor WHERE d_title IS NULL OR d_title = ‘’;
  #Shows doctors where title is NULL or empty

SELECT – Using ORDER BY Clause

- Syntax:
  SELECT *columns* FROM *table* ORDER BY *column* [ASC | DESC];
  - orders rows by *column*
  - ASC | DESC is optional

SELECT – Using ORDER BY Clause

- Syntax:
  SELECT *columns* FROM *table* ORDER BY *column* [ASC | DESC];
  - orders rows by *column*
  - ASC | DESC is optional
- Let's select the first name and last name of all patients, ordering them by last name
- SELECT p_firstname, p_lastname FROM Patient ORDER BY p_lastname;

SELECT – Using ORDER BY Clause

- Syntax:
  SELECT *columns* FROM *table* ORDER BY *column* [ASC | DESC];
  - orders rows by *column*
  - ASC | DESC is optional
- Now order them by last name, then first name
  SELECT p_firstname, p_lastname FROM Patient ORDER BY p_lastname ASC, p_firstname ASC;

Filtering Records & Sorting Data Summary

Command Desc.	Basic Syntax Structure	Example
WHERE Specifies a search condition	SELECT *columns* FROM *table* WHERE *condition*;	SELECT d_firstname, d_lastname FROM Doctor WHERE d_id < 3;
AND Combines two conditions together – record must match both conditions	SELECT *columns* FROM *table* WHERE *condition1* AND *condition2*;	SELECT * FROM Patient WHERE p_firstname = 'Becky' AND p_lastname = 'Jones';
OR Requires a record to match only one of the search conditions	SELECT *columns* FROM *table* WHERE *condition1* OR *condition2*;	SELECT * FROM Patient WHERE p_lastname = 'Jones' OR p_lastname ='Roberts';
[NOT] IN Searches for records matching on of the items in the list	SELECT *columns* FROM *table* WHERE *column* [NOT] IN (*value1*, *value2*, …);	SELECT * FROM Patient WHERE p_lastname IN ('Jones', 'Roberts');
[NOT] BETWEEN … AND Searches for records in a specified range of values	SELECT *columns* FROM *table* WHERE *column* [NOT] BETWEEN *value1* AND *value2*;	SELECT * FROM Appointment WHERE a_id BETWEEN 2 AND 5;

Filtering Records & Sorting Data Summary

Command Desc.	Basic Syntax Structure	Example
[NOT] LIKE Searches for records matching a search pattern -- used with wildcard character % represents any number of characters	SELECT *columns* FROM *table* WHERE *column* [NOT] LIKE *pattern*;	SELECT * FROM Patient WHERE p_firstname LIKE 'J%';
[NOT] LIKE Searches for records matching a search pattern – used with wildcard character _ represents character(s) in the indicated position	SELECT *columns* FROM *table* WHERE *column* [NOT] LIKE *pattern*;	SELECT * FROM Doctor WHERE d_lastname LIKE 'Sa_an';
IS [NOT] NULL Searches for records with a NULL or NOT NULL value in the indicated column	SELECT *columns* FROM *table* WHERE *column* IS [NOT] NULL;	SELECT * FROM Doctor WHERE d_title IS NULL;
ORDER BY Specifies the display order of query results	SELECT *columns* FROM *table* ORDER BY *column* [ASC | DESC];	SELECT p_firstname, p_lastname FROM Patient ORDER BY p_lastname, p_firstname;

Joining Data From Multiple Tables

JOIN

- A JOIN lets you collect information from two or more tables and present it as a single table
- A JOIN needs a way to uniquely identify each row in a table
  - A *primary key* is a column, or group of columns, whose values uniquely identify each row

Primary Keys

“Doctor” table

d_id	d_firstname	d_lastname	d_title	specialty
1	Bob	Samson	MD	General Internal
2	Ramesh	Hopkins	MD	General Internal
3	Jane	Adams	DO	Pediatrics
4	Sugirtha	Samson	MD	Toe Surgery

- The *primary key* for the Doctor table is ‘d_id’
  - It can be used to *uniquely* identify each row (doctor)

Primary Keys

"Doctor" table

d_id	d_firstname	d_lastname	d_title	specialty
1	Bob	Samson	MD	General Internal
2	Ramesh	Hopkins	MD	General Internal
3	Jane	Adams	DO	Pediatrics
4	Sugirtha	Samson	MD	Toe Surgery

- The *primary key* for the Doctor table is 'd_id'
  - It can be used to *uniquely* identify each row (doctor)
- We could also use the combination of 'd_firstname' and 'd_lastname' as the (composite) *primary key* for the Doctor table
  - No two doctors have the same first name and last name!

Primary Keys

"Patient" table

p_id	p_firstname	p_lastname	bdate
1	Mark	Jones	01/15/1990
2	Tom	Roberts	02/27/1985
3	Sue	Steinberg	03/07/1981
4	Becky	Jones	04/12/1985

- The *primary key* for the Patient table is 'p_id'
  - It can be used to *uniquely* identify each row (patient)

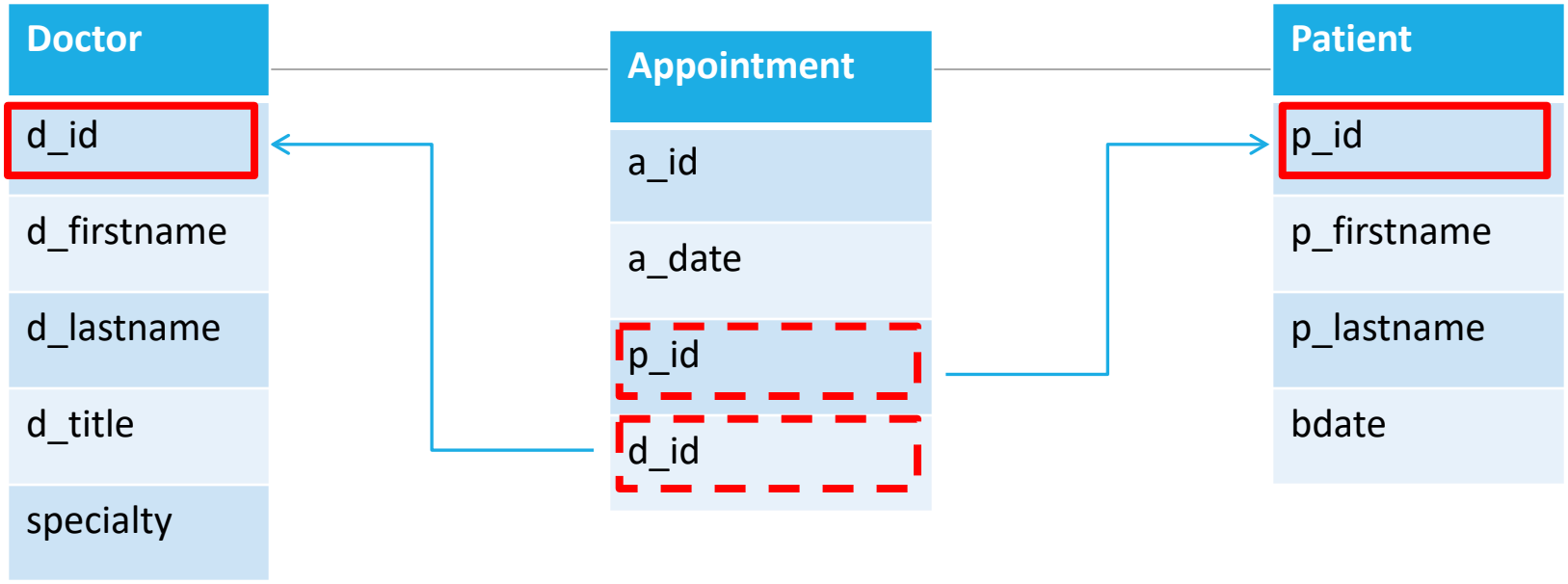
Primary Keys

"Patient" table

p_id	p_firstname	p_lastname	bdate
1	Mark	Jones	01/15/1990
2	Tom	Roberts	02/27/1985
3	Sue	Steinberg	03/07/1981
4	Becky	Jones	04/12/1985

- The *primary key* for the Patient table is 'p_id'
  - It can be used to *uniquely* identify each row (patient)
- We could also use the combination of 'p_firstname' and 'p_lastname' as the (composite) *primary key* for the Patient table
  - No two patients have the same first name and last name!

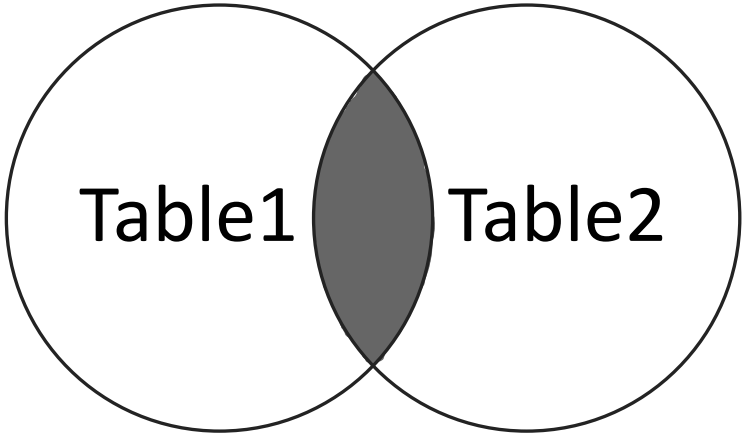
Foreign Keys



- Each appointment has a reference to a doctor ‘d_id’ and a patient ‘p_id’
  - These are *foreign key* values because they point to the *primary key* values of other tables

Inner Join

- An INNER JOIN returns all rows from *both* tables where there is a match



Inner Join

- Syntax:
  SELECT *t1.columns, t2.columns* FROM *table1* [AS] *t1*
  [INNER] JOIN *table2* [AS] *t2* ON *t1.column* = *t2.column*;
  - INNER is optional and implied in MySQL
  - *t1 is an example of a table alias*:
    - an alternate name for the table
    - Note: alias cannot have spaces
    - AS is optional
    - ***Very useful in JOINS!***

Inner Join

- Syntax:
  SELECT *t1.columns, t2.columns* FROM *table1* [AS] *t1*
  [INNER] JOIN *table2* [AS] *t2* ON *t1.column* = *t2.column*;
  - INNER is optional and implied in MySQL
- List each doctor with an appointment along with the appointment date & time
  SELECT d.d_lastname, a.a_date FROM Doctor d
  JOIN Appointment a ON d.d_id = a.d_id;
  - #Only doctors *with* appointments listed

Inner Join (practice)

- List each doctor with an appointment, along with the date & time AND patient name.

Inner Join (practice)

- List each doctor with an appointment, along with the date & time AND patient name.
  - SELECT d.d_lastname, a.a_date, p.p_lastname FROM Doctor d JOIN Appointment a ON d.d_id = a.d_id JOIN Patient p ON a.p_id = p.p_id;

Group (Aggregate) Functions

Group Functions

- Group functions return a single value, calculated from values in a column
  - COUNT(*column*)
  - AVG(*column*)
  - MAX(*column*)
  - MIN(*column*)
  - SUM(*column*)
  - STD(*column*)

Group Functions

- How many doctors are there?
  SELECT COUNT(*) FROM Doctor;

Group Functions

- Can you find the birthdates of the oldest and youngest patients?
  SELECT MIN(bdate), MAX(bdate) FROM Patient;

Grouping Rows – Using GROUP BY

- Syntax:
  SELECT *column1,*
  AGGREGATE_FUNCTION(*column2*) FROM *table*
  GROUP BY *column1;*
  - *column1* is a column to group or “collapse”
  - *column2* is a column to aggregate

Grouping Rows – Using GROUP BY

- Syntax:
  SELECT *column1,*
  AGGREGATE_FUNCTION(*column2*) FROM *table*
  GROUP BY *column1;*
  - *column1* is a column to group or “collapse”
  - *column2* is a column to aggregate
- Calculate the total number of appointments per doctor

```
SELECT d.d_id, COUNT(*) AS 'Num of Appts'
FROM Doctor d
JOIN Appointment a ON d.d_id = a.d_id
GROUP BY d.d_id; #Counts the number of appointments for each d_id
```

Grouping Rows – Using GROUP BY

- What time is each doctor's last appointment on 02/15/2016?

```
SELECT d.d_id, MAX(TIME(a.a_date)) AS 'Last Appt.' FROM Doctor d
JOIN Appointment a ON d.d_id = a.d_id
WHERE DATE(a.a_date) = '2016-02-15'
GROUP BY d.d_id;
```

Restricting Aggregated Output – Using HAVING

- Syntax:
  SELECT *column1, AGGREGATE_FUNCTION*(*column2*) FROM *table*
  GROUP BY *column1*
  HAVING *AGGREGATE_FUNCTION*(*column2*) *condition*;
  - *condition* is a condition to be satisfied by the aggregated value
- Which doctors have more than 2 appointments?

```
SELECT d.d_id, COUNT(*) AS 'Num of Appts' FROM Doctor d
JOIN Appointment a ON d.d_id = a.d_id
GROUP BY d.d_id
HAVING COUNT(*) > 2;
```

Restricting Aggregated Output – Using HAVING

- Which specialties have more than 2 doctors?

```
SELECT specialty, count(d_id) UniqueDocs FROM
Doctor
GROUP BY specialty
HAVING UniqueDocs > 1;
```

Functions & Grouping Data Summary

Command Desc.	Basic Syntax Structure	Example
Single-Row (Scalar) Functions return a single value, based on an input	SELECT *FUNCTION*(*input*) FROM *table*;	SELECT CONCAT(d_firstname, ' ', d_lastname, ', ', d_title) FROM Doctor;
Group (Aggregate) Functions return a single value, calculated from values in a column	SELECT *column1*, *AGGREGATE_FUNCTION*(*column2*) FROM *table* GROUP BY *column1*;	SELECT COUNT(d_id) FROM Doctor;
GROUP BY Groups or “collapses” rows based on one or more columns	SELECT *column1*, *AGGREGATE_FUNCTION*(*column2*) FROM *table* GROUP BY *column1*;	SELECT d.d_id, COUNT(*) AS 'Num of Appts’ FROM Doctor d JOIN Appointment a ON d.d_id = a.d_id GROUP BY d.d_id;
HAVING Specifies a search condition for an aggregated value	SELECT *column1*, *AGGREGATE_FUNCTION*(*column2*) FROM *table* GROUP BY *column1* HAVING *AGGREGATE_FUNCTION*(*column2*) *condition*;	SELECT d.d_id, COUNT(*) AS 'Num of Appts’ FROM Doctor d JOIN Appointment a ON d.d_id = a.d_id GROUP BY d_id HAVING COUNT(*) > 2;

Further Training

- Online resources
  - Codecademy
  - DataCamp
  - Udemy
  - SQLZOO - http://sqlzoo.net/
  - https://www.w3resource.com/sql-exercises/
  - SQL-EX - http://www.sql-ex.com/
  - MySQLTutorial - http://www.mysqltutorial.org/
  - https://www.listendata.com/2019/06/free-sql-download-to-practice-queries.html

Database Options

- AWS
- Microsoft Azure
- Snowflake