Database Applications (15-415)

SQL-Part I

Lecture 7, January 31, 2016

Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

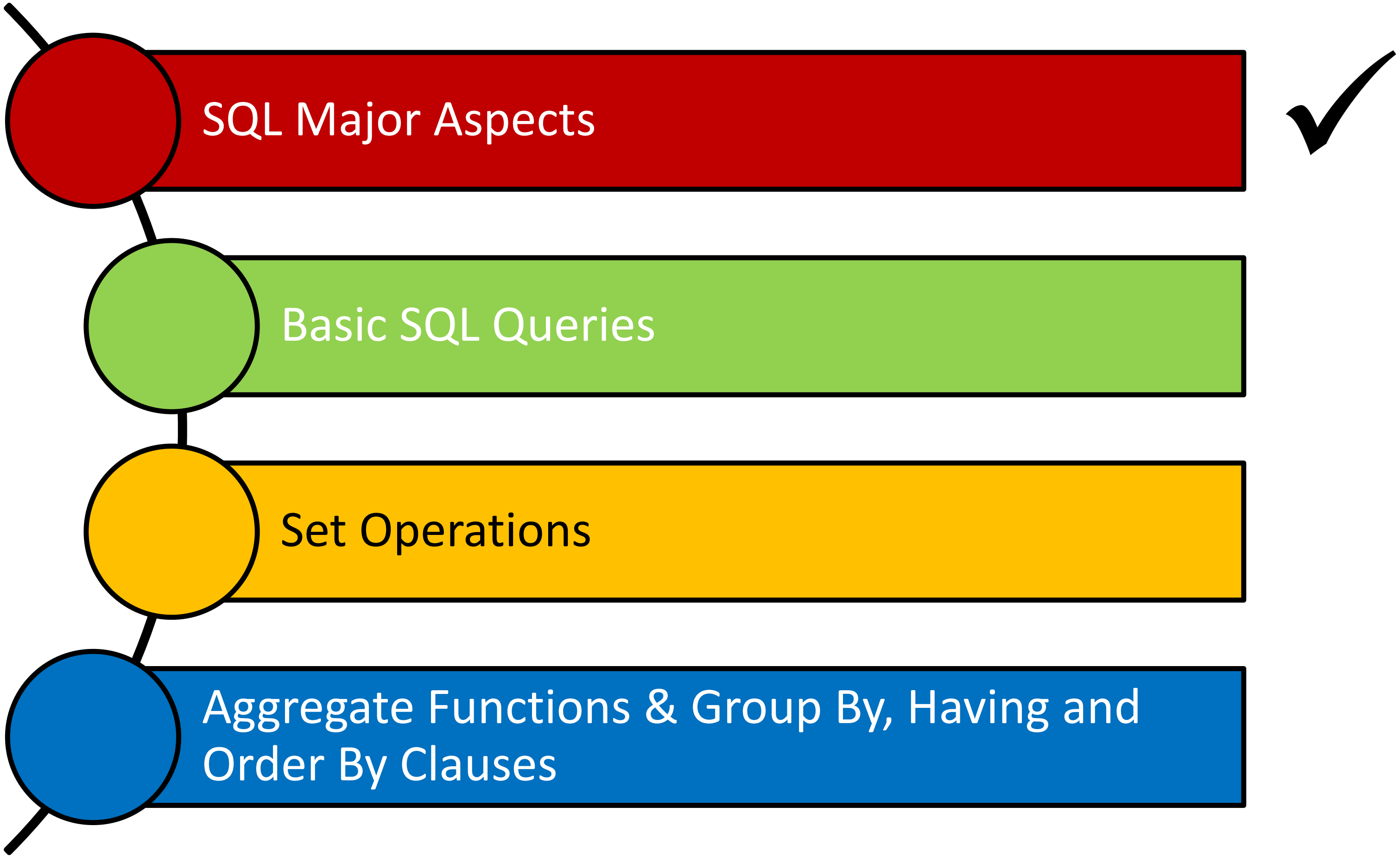
Today…

- Last Session:
  - Relational Calculus & Summary

- Today's Session:
  - Standard Query Language (SQL)- Part I

- Announcements:
  - PS2 is due on Sunday, Feb 07 by midnight
  - P1 will be out on Tuesday, Feb 02
  - We will practice on SQL during the upcoming recitation

Outline



- SQL Major Aspects ✓
- Basic SQL Queries
- Set Operations
- Aggregate Functions & Group By, Having and Order By Clauses

SQL Major Aspects

- A major strength of the relational model is that it supports simple and powerful *querying* of data

- Structured Query Language (SQL) is the most widely used commercial relational database language

- SQL has several aspects to it:
  1. Data Manipulation Language (DML)
     - It allows users to pose queries and insert, delete and modify <u>rows</u>
  2. Data Definition Language (DDL)
     - It allows users to create, delete, and modify <u>tables and views</u>

SQL Major Aspects

- SQL has several aspects to it:

3. Triggers and Advanced Integrity Constraints
   - It supports “triggers”, which are actions executed by the DBMS whenever changes to the database meet conditions specified in triggers

4. Embedded and Dynamic Language
   - Embedded SQL allows SQL code to be called from a *host language* (e.g., Java)
   - Dynamic SQL allows SQL queries to be constructed and executed at run-time

SQL Major Aspects

- SQL has several aspects to it:

  3. Triggers and Advanced Integrity Constraints
     - It supports “triggers”, which are actions executed by the DBMS whenever changes to the database meet conditions specified in triggers

  4. Embedded and Dynamic Language
     - Embedded SQL allows SQL code to be called from a *host language* (e.g., Java)
     - Dynamic SQL allows SQL queries to be constructed and executed at run-time

Sample programs will be discussed and coded in recitations!

SQL Major Aspects

- SQL has several aspects to it:

5. Remote Database Access
    - It allows connecting client programs to remote database servers

6. Transaction Management
    - It allows users to explicitly control aspects of how a transaction is to be executed (*later in the semester*)

7. Security
    - It provides mechanisms to control users' accesses to data objects (e.g., tables and views)

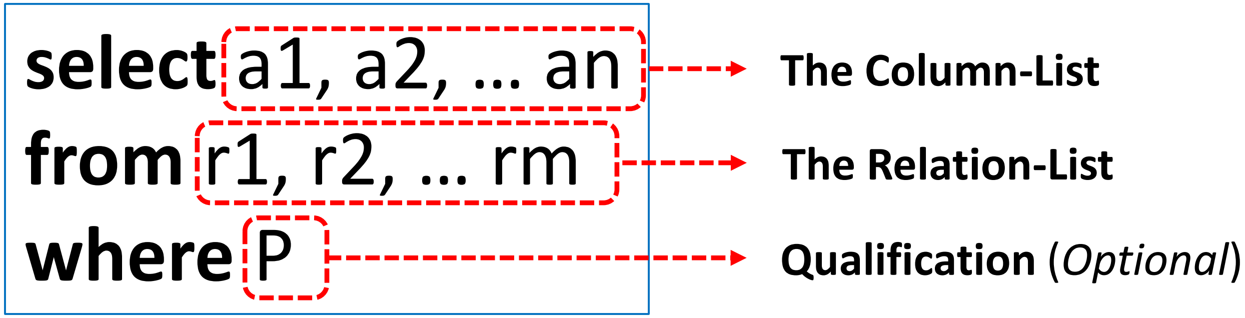
And others…

Outline

- SQL Major Aspects
- Basic SQL Queries ✓
- Set Operations
- Aggregate Functions & Group By, Having and Order By Clauses

Basic SQL Queries

- The basic form of an SQL query is as follows:



```
select a1, a2, ... an
from r1, r2, ... rm
where P
```

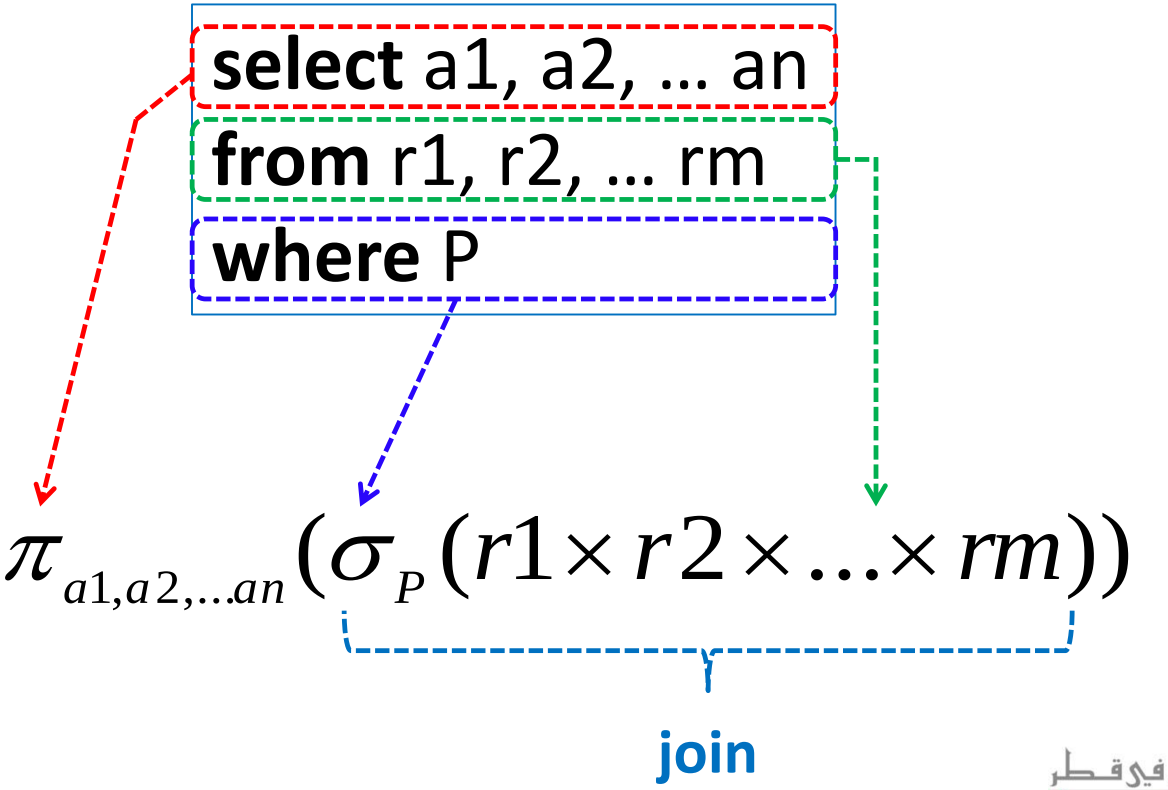
The Column-List

The Relation-List

Qualification (*Optional*)

Equivalence to Relational Algebra

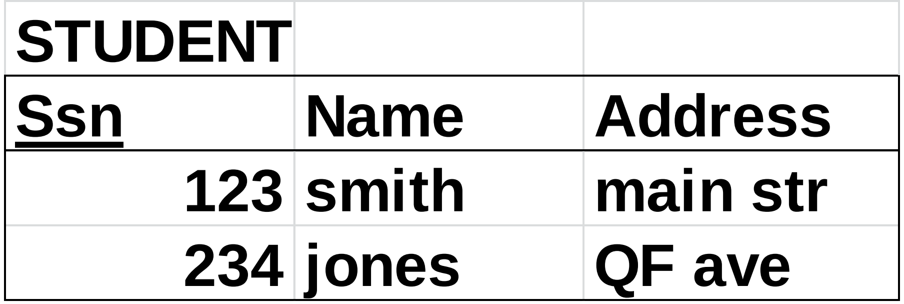
- The basic form of an SQL query is as follows:



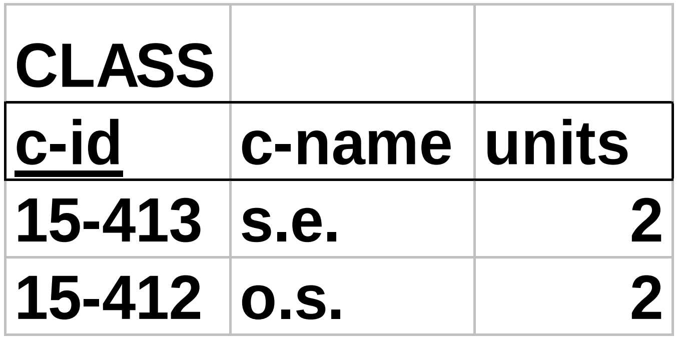
select a1, a2, … an
from r1, r2, … rm
where P

$$\pi_{a1,a2,\ldots an}(\underbrace{\sigma_P(r1\times r2\times\ldots\times rm)}_{\textbf{join}})$$

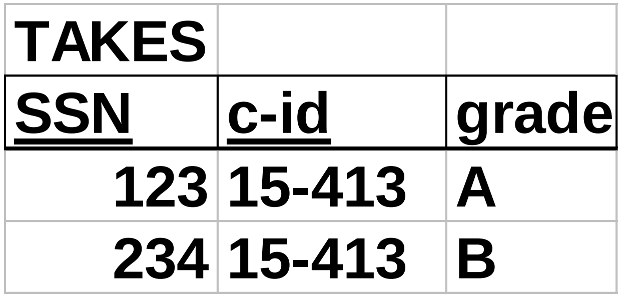
Reminder: Our Mini-U DB



STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	QF ave



CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2



TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

The WHERE Clause

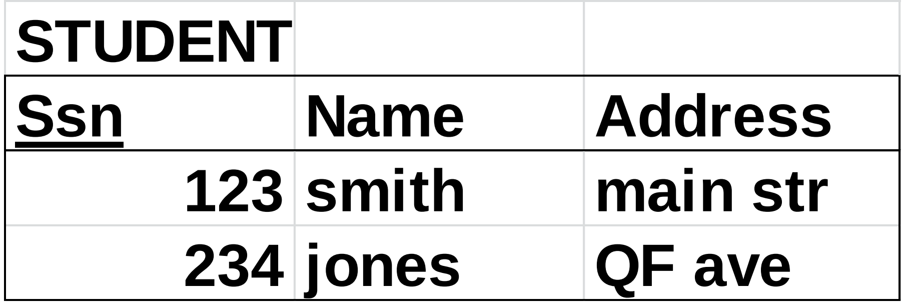
- Find the ssn(s) of everybody called “smith”

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	QF ave

```
select ssn
from student
where name=‘smith’
```

The WHERE Clause

- Find ssn(s) of all “smith”s on “main”



STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	QF ave

```
select ssn
from student
where address=‘main’ and
    name = ‘smith’
```

The WHERE Clause

- Boolean operators (**and, or, not**)
- Comparison operators (<, ≤, >, ≥, =, ≠)
- And more…

What About Strings?

- Find student ssn(s) who live on “main” (st or str or street – i.e., “main st” or “main str” or “main street”)

```
select ssn
from student
where address like ‘main%’
```

%: Variable-length do not care (i.e., stands for 0 or more arbitrary characters)
_: Single-character do not care (i.e., stands for any 1 character)

Another Example on *Pattern Matching*

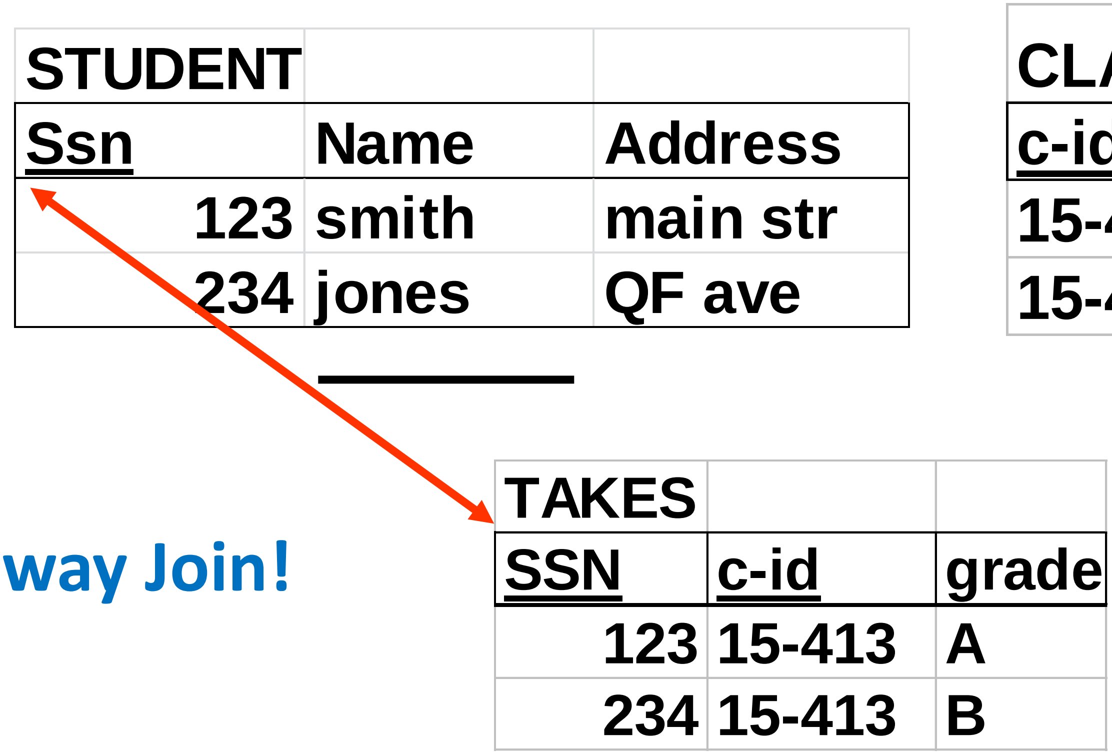
- Find the ages of sailors whose names begin and end with B and have at least 3 characters

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

```
select S.age
from Sailors S
where S.sname like ‘B_%B’
```

The FROM Clause

- Find the names of students taking 15-415



STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	QF ave

CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

2-way Join!

The FROM Clause

- Find the names of students taking 15-415

```
select Name
from STUDENT, TAKES
where   ???
```

The FROM Clause

- Find the names of students taking 15-415

```
select Name
from STUDENT, TAKES
where   STUDENT.ssn = TAKES.ssn
        and TAKES.c-id = ‘15-415’
```

Renaming: Tuple Variables

- Find the names of students taking 15-415

```
select Name
from STUDENT as S, TAKES as T
where   S.ssn = T.ssn
        and T.c-id = “15-415”
```

Optional!

Renaming: Self-Joins

- Find Tom’s grandparent(s)

PC	
<u>p-id</u>	**c-id**
Mary	**Tom**
Peter	**Mary**
John	**Tom**

PC	
<u>p-id</u>	**c-id**
Mary	**Tom**
Peter	**Mary**
John	**Tom**

select gp.p-id
from PC **as** gp, PC
where gp.c-id= PC.p-id
  and PC.c-id = ‘Tom’

More on Self-Joins

- Find names and increments for the ratings of persons who have sailed two different boats on the same day

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

Reserves		
Sid	**Bid**	**Day**
22	101	10/10/2013
22	102	10/10/2013

More on Self-Joins

- Find names and increments for the ratings of persons who have sailed two different boats on the same day

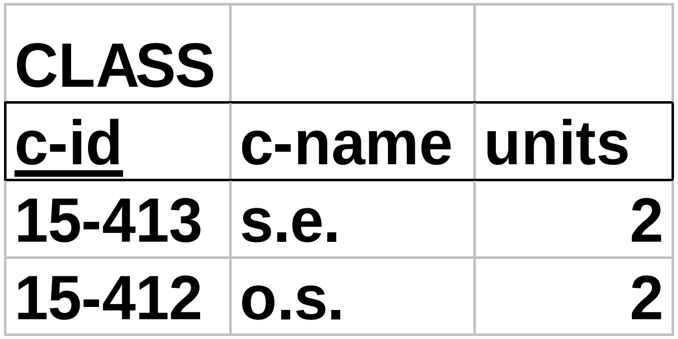
Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

Reserves		
Sid	**Bid**	**Day**
22	101	10/10/2013
22	102	10/10/2013

```
select S.sname, S.rating+1 as rating
from Sailors S, Reserves R1, Reserves R2
where S.sid = R1.sid and S.sid = R2.sid
  and  R1.day = R2.day and R1.bid != R2.bid
```

Renaming: Theta Joins

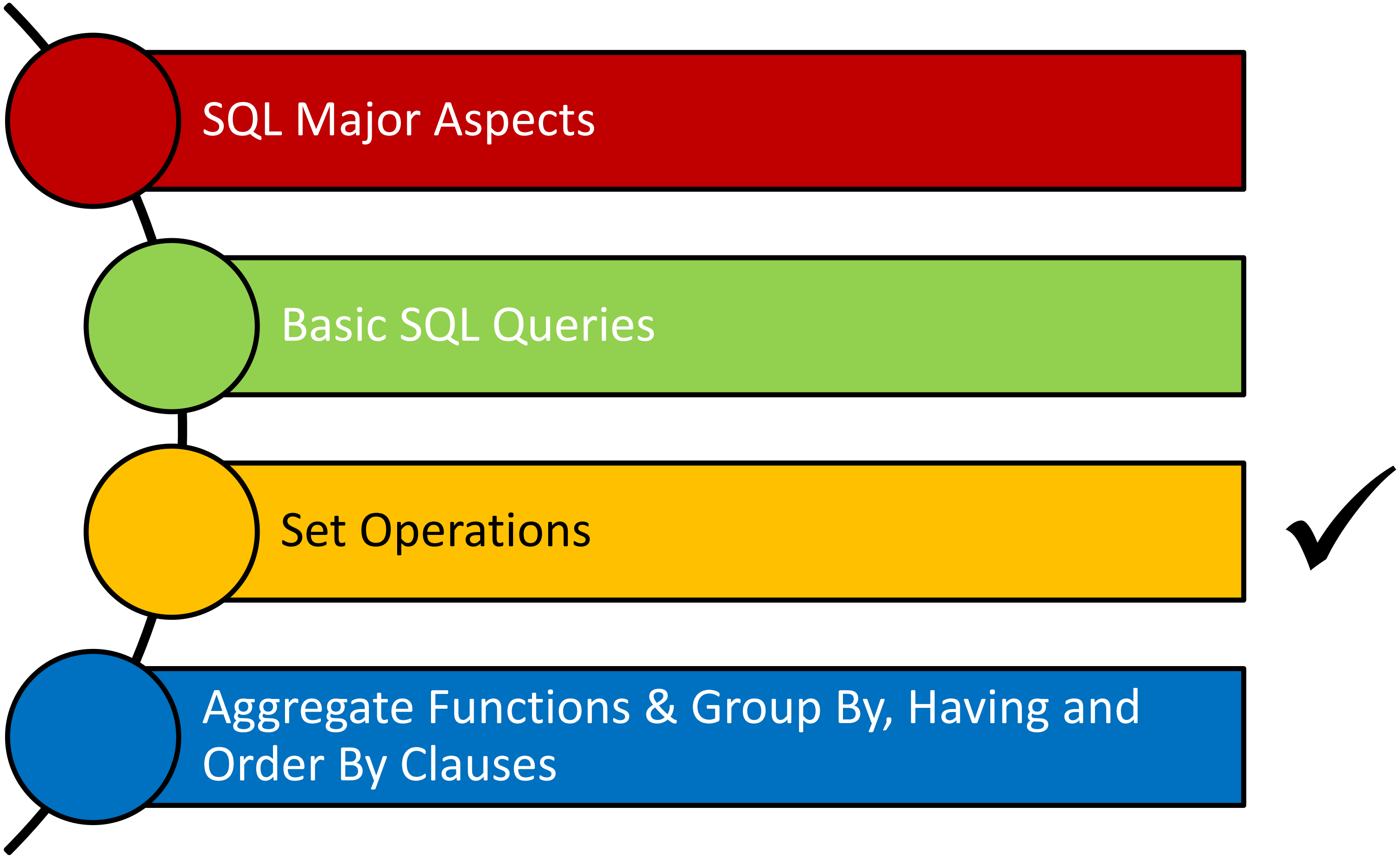
- Find course names with more units than 15-415



CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2

```
select c1.c-name
from class as c1, class as c2
where c1.units > c2.units
   and  c2.c-id = ‘15-415’
```

Outline



- SQL Major Aspects
- Basic SQL Queries
- Set Operations ✓
- Aggregate Functions & Group By, Having and Order By Clauses

Set Operations

- Find ssn(s) of students taking both 15-415 and 15-413

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

~~**select** ssn~~

~~**from** takes~~

~~**where** c-id='15-415' **and**~~
~~c-id='15-413'~~

Set Operations

- Find ssn(s) of students taking both 15-415 and 15-413

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

```
(select ssn from takes where c-id="15-415" )
intersect
(select ssn from takes where c-id="15-413" )
```

Other operations: **union , except**

Set Operations

- Find ssn(s) of students taking 15-415 <u>or</u> 15-413

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

```
(select ssn from takes where c-id=“15-415” )
union
(select ssn from takes where c-id=“15-413” )
```

Set Operations

- Find ssn(s) of students taking 15-415 but not 15-413

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

```
(select ssn from takes where c-id=“15-415” )
except
(select ssn from takes where c-id=“15-413” )
```

Another Example on Set Operations

- Find the names of sailors who have reserved both a red and a green boat

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

Reserves		
Sid	**Bid**	**Day**
22	101	10/10/2013
22	102	10/11/2013

Boats		
Bid	**Bname**	**Color**
101	Interlake	Red
102	Clipper	Green

Another Example on Set Operations

- Find the names of sailors who have reserved both a red and a green boat

```
(select S.sname from Sailors S, Reserves R, Boats B
where S.sid = R.sid and R.bid = B.bid and B.color = ‘green’)
intersect
(select S2.sname from Sailors S2, Reserves R2, Boats B2
where S2.sid = R2.sid and R2.bid = B2.bid and B2.color = ‘red’)
```

The query contains a “subtle bug” which arises because we are using *sname* to identify Sailors, and “sname” is not a key for Sailors!

We can compute the names of such Sailors using a NESTED query (*which we cover next lecture!*)

Outline

- SQL Major Aspects
- Basic SQL Queries
- Set Operations
- Aggregate Functions & Group By, Having and Order By Clauses ✓

Aggregate Functions

- Find average grade, across all students

SSN	c-id	grade
123	15-413	4
234	15-413	3

```
select ??
from takes
```

Aggregate Functions

- Find average grade, across all students

SSN	c-id	grade
123	15-413	4
234	15-413	3

```
select avg(grade)
from takes
```

Other functions: Count ([Distinct] A), Sum ([Distinct] A), Max (A), Min (A), assuming column A

Aggregate Functions

- Find total number of enrollments

SSN	c-id	grade
123	15-413	4
234	15-413	3

```
select count(*)
from takes
```

Aggregate Functions

- Find total number of students in 15-415

SSN	c-id	grade
123	15-413	4
234	15-413	3

```
select count(*)
from takes
where c-id=‘15-415’
```

Aggregate Functions

- Find the name and age of the oldest sailor

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

~~**select** S.sname, **max** (S.age)~~
~~**from** Sailors S~~

This query is illegal in SQL- If the "select" clause uses an aggregate function, it must use ONLY aggregate function unless the query contains a "group by" clause!

The GROUP BY and HAVING Clauses

- Find the age of the youngest sailor for each rating level

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

- In general, we do not know how many rating levels exist, and what the rating values for these levels are!

- Suppose we know that rating values go from 1 to 10; we can write 10 queries that look like this (!):

For $i = 1, 2, \ldots, 10$:

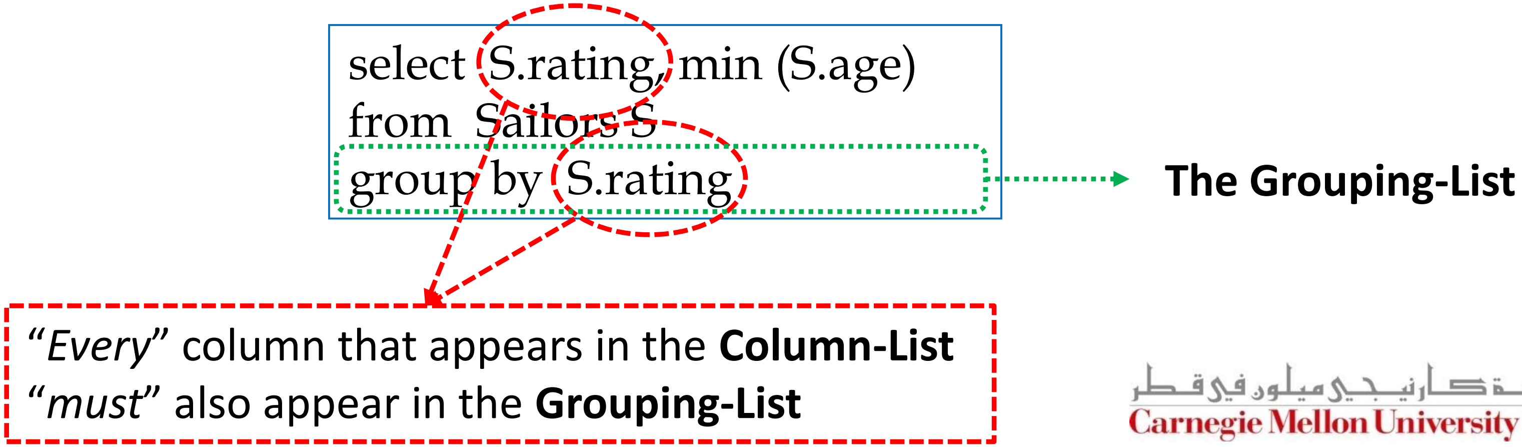
```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

The GROUP BY and HAVING Clauses

- Find the age of the youngest sailor for each rating level

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

- Using the GROUP BY clause, we can write this query as follows:



```
select S.rating, min (S.age)
from  Sailors S
group by S.rating
```

The Grouping-List

"*Every*" column that appears in the **Column-List** "*must*" also appear in the **Grouping-List**

The GROUP BY and HAVING Clauses

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 sailors

Sailors			
Sid	**Sname**	**Rating**	**age**
22	Dustin	7	45.0
29	Brutus	1	33.0

```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

The GROUP BY and HAVING Clauses

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 sailors

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

rating	minage
3	25.5
7	35.0
8	25.5

The GROUP BY and HAVING Clauses

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 sailors, and with every sailor under 60

```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1 AND EVERY (S.age <=60)
```

The GROUP BY and HAVING Clauses

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 sailors, and with every sailor under 60

rating	age
7	45.0
1	33.0
8	55.5
8	25.5
10	35.0
7	35.0
10	16.0
9	35.0
3	25.5
3	63.5
3	25.5

rating	age
1	33.0
3	25.5
3	63.5
3	25.5
7	45.0
7	35.0
8	55.5
8	25.5
9	35.0
10	35.0

rating	minage
7	35.0
8	25.5

What would be the result if we change EVERY to ANY in “HAVING COUNT (*) > 1 AND EVERY (S.age <=60)”?

The GROUP BY and HAVING Clauses

- Find age of the youngest sailor with age ≥ 18, for each rating level with at least 2 sailors between 18 and 60

```
SELECT  S.rating,  MIN (S.age) AS minage
FROM  Sailors S
WHERE  S.age >= 18 AND S.age <= 60
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

Will this give the same result as the previous query which uses the EVERY clause?

Will this give the same result as the previous query which uses the ANY clause?

The ORDER BY Clause

- Find student records, sorted in name order

select *

from student

~~**where** ??~~

The ORDER BY Clause

- Find student records, sorted in name order

```
select *
from student
order by name asc
```

asc is the default

The ORDER BY Clause

- Find student records, sorted in name order; break ties by reverse ssn

```
select *
from student
order by name, ssn desc
```

More Examples

- Find the total number of students in each course

SSN	c-id	grade
123	15-413	4
234	15-413	3

select count(*)
from takes
~~**where** ???~~

More Examples

- Find the total number of students in each course

SSN	c-id	grade
123	15-413	4
234	15-413	3

c-id	count
15-413	2

```
select c-id, count(*)
from takes
group by c-id
```

More Examples

- Find total number of students in each course, and sort by count, in decreasing order

SSN	c-id	grade
123	15-413	4
234	15-413	3

c-id	pop
15-413	2

```
select c-id, count(*) as pop
from takes
group by c-id
order by pop desc
```

Concluding Remarks

- SQL was an important factor in the early acceptance of the relational model
    - It is more natural than earlier procedural query languages

- SQL is relationally complete; in fact, significantly more expressive power than relational algebra

- Even queries that can be expressed in relational algebra can often be expressed more naturally in SQL

Next Class

SQL- Part II