

## CS460: Intro to Database Systems

# Class 8: SQL, The Query Language – Part I

Instructor: Manos Athanassoulis

<https://bu-disc.github.io/CS460/>

# Today's course

**intuitive** way to ask **queries**

unlike *procedural languages* (C/C++, java)  
[which specify **how** to solve a problem (or answer a question)]

SQL is a **declarative query** language  
[we ask **what we want** and the DBMS is going to deliver]

# Introduction to SQL

SQL is a relational **query language**

supports **simple** yet **powerful** *querying* of data

It has two parts:

DDL: Data Definition Language (define and modify schema)

(we discussed about that in Relational Model)

CREATE TABLE

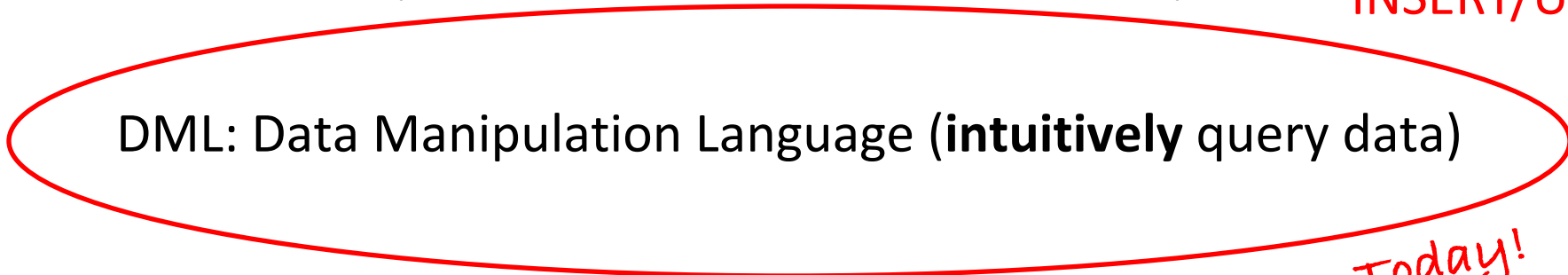


INSERT/UPDATE/DELETE



DML: Data Manipulation Language (**intuitively** query data)

Today!



# Reiterate some terminology

Relation (or table)

Students ← name

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

← schema

← data  
(instance)

Row (or tuple)

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

Column (or attribute)

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

# Reiterate some terminology

Primary Key (PK)

<u>sid</u>	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

The PK of a relation is the column (or the group of columns) that can uniquely define a row.

In other words:

Two rows **cannot** have the same PK.

# The simplest SQL query

“find all contents of a table”

in this example: “Find all info for all students”

```
SELECT *  
FROM Students S
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53777	White	white@cs	19	4.0

to find just names and logins, replace the first line:

```
SELECT S.name, S.login
```

# Show specific columns

“find name and login for all students”

```
SELECT S.name, S.login  
FROM Students S
```

name	login
Jones	jones@cs
Smith	smith@ee
White	white@cs

this is called: “**project** name and login from table Students”

# Show specific rows

“find all 18 year old students”

```
SELECT *  
  FROM Students S  
 WHERE S.age=18
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

this is called: “**select** students with age 18.”



# Querying Multiple Relations

can specify a join over two tables as follows:

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid AND E.grade='B'
```

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

result =

S.name	E.cid
Jones	History105

# Basic SQL Query

SELECT	[DISTINCT]	target-list
FROM		relation-list
WHERE		qualification

relation-list : a list of relations

target-list : a list of attributes of tables in *relation-list*

qualification : comparisons using AND, OR and NOT

comparisons are: <attr> <op> <const> or <attr1> <op> <attr2>, where *op* is:

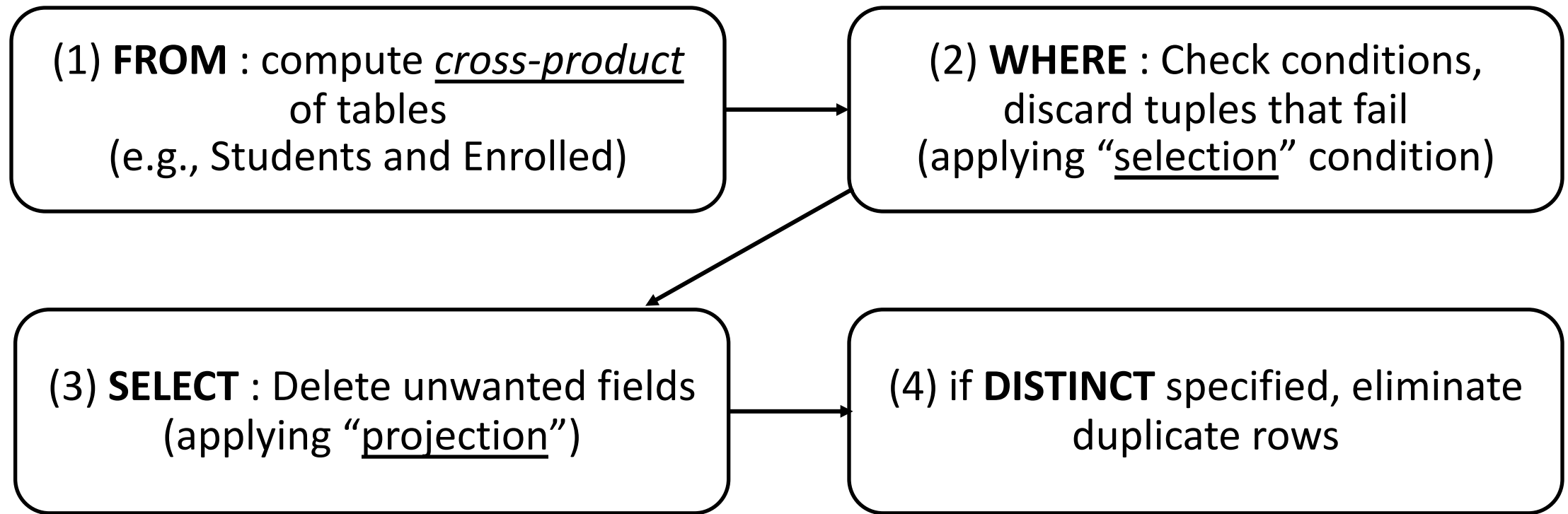
<, >, =, ≤, ≥, ≠

DISTINCT: *optional*, removes duplicates

By default SQL SELECT does not eliminate duplicates! (“multiset”)

# Query Semantics

Conceptually, a SQL query can be computed:



probably the least efficient way to compute a query!

**Query Optimization** finds the *same answer* more efficiently

# Remember the query and the data

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid AND E.grade='B'
```

sid	cid	grade
53831	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2

# Step 1 – Cross Product

Combine with cross-product all tables of the **FROM** clause.

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

```
SELECT S.name, E.cid  
FROM Students S, Enrolled E  
WHERE S.sid=E.sid AND E.grade='B'
```

## Step 2 - Discard tuples that fail predicate

Make sure the **WHERE** clause is true!

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'
```

# Step 3 - Discard Unwanted Columns

Show only what is on the **SELECT** clause.

S.sid	S.name	S.login	S.age	S.gpa	E.sid	E.cid	E.grade
53666	Jones	jones@cs	18	3.4	53831	Carnatic101	C
53666	Jones	jones@cs	18	3.4	53832	Reggae203	B
53666	Jones	jones@cs	18	3.4	53650	Topology112	A
53666	Jones	jones@cs	18	3.4	53666	History105	B
53688	Smith	smith@ee	18	3.2	53831	Carnatic101	C
53688	Smith	smith@ee	18	3.2	53831	Reggae203	B
53688	Smith	smith@ee	18	3.2	53650	Topology112	A
53688	Smith	smith@ee	18	3.2	53666	History105	B

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='B'
```

# Now the Details...

We will use these  
instances of relations  
in our examples.

## *Reserves*

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/16
95	103	11/12/16

## *Sailors*

<u>sid</u>	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

## *Boats*

<u>bid</u>	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red



# Another Join Query

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid AND bid=103
```

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/16
22	dustin	7	45.0	95	103	11/12/16
31	lubber	8	55.5	22	101	10/10/16
31	lubber	8	55.5	95	103	11/12/16
95	Bob	3	63.5	22	101	10/10/16
95	<b>Bob</b>	3	63.5	95	103	11/12/16

# Range Variables

can associate “range variables” with the tables in the FROM clause

a shorthand, like the rename operator from relational algebra

saves writing, makes queries easier to understand

“FROM Sailors, Reserves”

“FROM Sailors **S**, Reserves **R**”

needed when ambiguity could arise

for example, if same table used multiple times in same FROM (called a “self-join”)

“FROM Sailors **s1**, Sailors **s2**”

# Range Variables

```
SELECT sname  
FROM Sailors,Reserves  
WHERE Sailors.sid=Reserves.sid AND bid=103
```

can be  
rewritten using  
range variables as:

```
SELECT S.sname  
FROM Sailors S, Reserves R  
WHERE S.sid=R.sid AND bid=103
```

# Range Variables

an example requiring range variables (self-join)

```
SELECT s1.sname, s1.age, s2.sname, s2.age  
FROM Sailors s1, Sailors s2  
WHERE s1.age > s2.age
```

another one: “\*” if you don’t want a projection:

```
SELECT *  
FROM Sailors S  
WHERE S.age > 20
```

# Find sailors who have reserved at least one boat

```
SELECT  S.sid  
FROM    Sailors S, Reserves R  
WHERE   S.sid=R.sid
```

does DISTINCT makes a difference?



what is the effect of replacing *S.sid* by *S.sname* in the SELECT clause?

Would adding DISTINCT to this variant of the query make a difference?



# Expressions

Can use arithmetic expressions in SELECT clause  
(plus other operations we'll discuss later)

Use **AS** to provide column names

```
SELECT S.age, S.age-5 AS age1, 2*S.age AS age2
FROM   Sailors S
WHERE  S.sname = 'dustin'
```

Can also have expressions in WHERE clause:

```
SELECT S1.sname AS name1, S2.sname AS name2
FROM   Sailors S1, Sailors S2
WHERE  2*S1.rating = S2.rating - 1
```

# String operations

SQL also supports some string operations

“LIKE” is used for string matching.

```
SELECT  S.age, age1=S.age-5, 2*S.age AS age2
FROM    Sailors S
WHERE   S.sname LIKE 'B_%B'
```

'\_' stands for any one character

'%' stands for 0 or more arbitrary characters

# More Operations

SQL queries produce new tables

If the results of two queries are **union-compatible**  
(same number and types of columns)  
then we can apply logical operations



UNION

INTERSECTION

SET DIFFERENCE (called EXCEPT or MINUS)



Find sids of sailors who have reserved a red or a green boat

**UNION**: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries)

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND
(B.color='red' OR B.color='green')
```

VS.

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND B.color='red'
UNION
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid=B.bid AND
      B.color='green'
```

Find sids of sailors who have reserved a red and a green boat

If we simply replace **OR** by **AND** in the previous query, we get the wrong answer. (Why?)

Instead, could use a self-join:

```
SELECT R1.sid
FROM Boats B1, Reserves R1,
      Boats B2, Reserves R2
WHERE R1.sid=R2.sid
      AND R1.bid=B1.bid
      AND R2.bid=B2.bid
      AND (B1.color='red' AND B2.color='green')
```

## AND Continued...

**INTERSECT**: discussed in the book. Can be used to compute the intersection of any two *union-compatible* sets of tuples

Also in text: **EXCEPT**  
(sometimes called MINUS)

Included in the SQL/92 standard, but some systems do not support them

Key field!

```
SELECT S.sid
FROM Sailors S, Boats B,
      Reserves R
WHERE S.sid=R.sid
      AND R.bid=B.bid
      AND B.color='red'
```

**INTERSECT**

```
SELECT S.sid
FROM Sailors S, Boats B,
      Reserves R
WHERE S.sid=R.sid
      AND R.bid=B.bid
      AND B.color='green'
```

# Your turn ...



1. Find (the names of) all sailors who are over 50 years old
2. Find (the names of) all boats that have been reserved at least once
3. Find all sailors who have not reserved a red boat (hint: use "EXCEPT")
4. Find all pairs of same-color boats
5. Find all pairs of sailors in which the older sailor has a lower rating

**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

1. Find (the names of) all sailors who are over 50 years old



**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

2. Find (the names of) all boats that have been reserved at least once



**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

3. Find all sailors who have not reserved a red boat



**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

4. Find all pairs of same-color boats





**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

5. Find all pairs of sailors in which the older sailor has a lower rating



# Nested Queries

powerful feature of SQL:

WHERE clause can itself contain an SQL query!

Actually, so can FROM and HAVING clauses.

*Names of sailors who have reserved boat #103*

```
SELECT  S.sname
FROM    Sailors S
WHERE   S.sid IN (SELECT R.sid
                  FROM    Reserves R
                  WHERE   R.bid=103)
```

# Nested Queries

to find sailors who have *not* reserved #103, use **NOT IN**.

To understand semantics of nested queries:

*think* of a *nested loops* evaluation


*for each Sailors tuple*

*check the qualification by computing the subquery*

# Nested Queries with Correlation

*Find names of sailors who have reserved boat #103*

```
SELECT  S.sname
FROM    Sailors S
WHERE   EXISTS (SELECT  *
                  FROM    Reserves R
                  WHERE R.bid=103 AND S.sid=R.sid)
```



**EXISTS** is another set operator, like **IN** (also **NOT EXISTS**)

If **EXISTS UNIQUE** is used, and \* is replaced by *R.bid*, finds sailors with at most one reservation for boat #103.

UNIQUE checks for duplicate tuples in a subquery;

Subquery must be recomputed for each Sailors tuple.

Think of subquery as a function call that runs a query!

# More on Set-Comparison Operators

We've already seen IN, EXISTS and UNIQUE. Can also use **NOT IN**, **NOT EXISTS** and **NOT UNIQUE**.

Also available: *op ANY, op ALL*

Find sailors whose rating is greater than that of some sailor called Horatio:

```
SELECT  *
FROM    Sailors S
WHERE   S.rating > ANY (SELECT  S2.rating
                        FROM    Sailors S2
                        WHERE   S2.sname='Horatio')
```

## Rewriting INTERSECT Queries Using IN

*Find sids of sailors who have reserved both a red and a green boat*

```
SELECT  R.sid
FROM    Boats B, Reserves R
WHERE   R.bid=B.bid
        AND B.color='red'
        AND R.sid IN (SELECT R2.sid
                        FROM    Boats B2, Reserves R2
                        WHERE   R2.bid=B2.bid
                        AND     B2.color='green')
```

Similarly, **EXCEPT** queries can be re-written using **NOT IN**.

How would you change this to find *names* (not *sids*) of Sailors who've reserved both red and green boats?



# Query #3 revisited ...

- Find all sailors who have not reserved a red boat  
(this time, without using “EXCEPT”)

**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

# Answer ...

3. Find all sailors who have not reserved a red boat

**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)



# Another Correct Answer ...

3. Find all sailors who have not reserved a red boat

**Reserves** (sid, bid, day)

**Sailors** (sid, sname, rating, age)

**Boats** (bid, bname, color)

# Division in SQL

Find sailors who have reserved all boats.

*Sailors S for which ...*

```

SELECT S.sname
FROM   Sailors S
WHERE  NOT EXISTS (SELECT B.bid
                   FROM   Boats B
                   WHERE  NOT EXISTS (SELECT R.bid
                                     FROM   Reserves R
                                     WHERE  R.bid=B.bid
                                     a Reserves tuple AND R.sid=S.sid))
showing S reserved B

```

*there is no boat B without ...*

# Aggregate Operators

Significant extension of relational algebra.

```
SELECT COUNT (*)  
FROM Sailors S
```

```
SELECT AVG (S.age)  
FROM Sailors S  
WHERE S.rating=10
```

```
SELECT COUNT (DISTINCT S.rating)  
FROM Sailors S  
WHERE S.sname='Bob'
```

COUNT (\*)  
COUNT ( [DISTINCT] A)  
SUM ( [DISTINCT] A)  
AVG ( [DISTINCT] A)  
MAX (A)  
MIN (A)

*single column*

# Aggregate Operators

COUNT (\*)  
COUNT ( [DISTINCT] A)  
SUM ( [DISTINCT] A)  
AVG ( [DISTINCT] A)  
MAX (A)  
MIN (A)

*single column*

```
SELECT  S.sname  
FROM    sailors S  
WHERE   S.rating = (SELECT  MAX(S2.rating)  
                   FROM    sailors S2)
```

```
SELECT  AVG (DISTINCT S.age)  
FROM    sailors S  
WHERE   S.rating=10
```



# Find name and age of the oldest sailor(s)

The first query is incorrect!

```
SELECT S.sname, MAX (S.age)  
FROM Sailors S
```

Third query equivalent to second query

allowed in SQL/92 standard, but not  
supported in some systems.

```
SELECT S.sname, S.age  
FROM Sailors S  
WHERE S.age =  
      (SELECT MAX (S2.age)  
       FROM   Sailors S2)
```

```
SELECT S.sname, S.age  
FROM   Sailors S  
WHERE  (SELECT MAX (S2.age)  
        FROM   Sailors S2)  
        = S.age
```