SQL

Complete Summary

Instructor:

Deepak Poonia

MTech, IISc Bangalore

GATE CSE AIR 53; AIR 67;
AIR 107; AIR 206; AIR 256

DBMS Complete Course:

https://www.goclasses.in/courses/Database-Management-Systems

NOTE :

Complete Discrete Mathematics & Complete Engineering

Mathematics Courses, by GO Classes, are **FREE** for ALL learners.

Visit here to watch : https://www.goclasses.in/s/store/

SignUp/Login on Goclasses website for free and start learning.

GO CLASSES

FREE

COMPLETE COURSE

Engineering Mathematics

GATE CS IT

Freely Available Features

FREE

- All Video Lectures
- Quizzes
- Homeworks, Practice Sets
- Annotated Notes
- Toppers' Hand Written Notes
- GATE PYQs Video Solutions

SACHIN MITTAL SIR

www.goclasses.in

/ GoClasses

GO CLASSES

FREE

COMPLETE COURSE

Discrete Mathematics

GATE CS IT

Freely Available Features

FREE

- All Video Lectures
- Quizzes, Homeworks
- Annotated Notes
- Toppers' Hand Written Notes
- Summary Lectures
- GATE PYQs Video Solutions

DEEPAK POONIA SIR

www.goclasses.in

/ GoClasses

GO CLASSES

FREE

COMPLETE COURSE

Linear Algebra

GATE DA

Freely Available Features

FREE

- All Video Lectures
- Quizzes
- Homeworks, Practice Sets
- Annotated Notes
- Toppers' Hand Written Notes
- GATE PYQs Video Solutions

SACHIN MITTAL SIR

www.goclasses.in

/ GoClasses

We are on Telegram. **Contact us** for any help.

Link in the Description!!

Join GO Classes **Doubt Discussion** Telegram Group :

Username:

@GATECSE_GOCLASSES

We are on Telegram. **Contact us** for any help.

Join GO Classes Telegram Channel, Username: **@GOCLASSES_CSE**

Join GO Classes **Doubt Discussion** Telegram Group :

Username: **@GATECSE_Goclasses**

(Any doubt related to Goclasses Courses can also be asked here.)

Join GATEOverflow Doubt Discussion Telegram Group :

Username: **@GateOverflow_CSE**

GO CLASSES


GO CLASSES
GO CLASSES
GO CLASSES
GO CLASSES
GO CLASSES
GO CLASSES

SQL

Complete Summary

The Database Language SQL

- SQL: Structured Query Language
  - Pronounced "S-Q-L" or "sequel"
  - The standard query language supported by most DBMS

Structured Query Language (SQL) is the most commonly used relational database query language.

NOTE:

SQL is Declarative, Not procedural.

Only specify What we want ;
Not How to get it

Defines Step by step Procedure about How to get info.

Let me give you a real-world example: I need a cup of tea.

Task

Procedural:

1. Go to kitchen
2. Get sugar, milk, and tea,
3. Mix them, and heat over the fire till it boils
4. Put that in a cup and bring it to me

Declarative:

1. Get me a cup of tea.

SQL is Declarative, Not procedural.

Declarative programming is where you say what you want without having to say how to do it.
With procedural programming, you have to specify exact steps to get the result.

SQL is declarative, because the queries don't specify steps to produce the result.

In a procedural language, you define the whole process and provide the steps how to do it.

In a declarative language, you just set the command or order, and let it be on the system how to complete that order. You just need your result without digging into how it should be done.

NOTE:

SQL is Declarative, Not procedural.

SQL is a **declarative query** language

We ask **what we want** and the DBMS is going to deliver!

We will Revisit this point shortly.

Next Topic:

Set
Vs
Multiset (Bag)

- We have seen that sets are unordered collections of items, which do not contain duplicates.
- A *bag* is an *unordered* collection of items that *may* contain duplicates.
- Bags are sometimes called *multisets*.

Set : Unordered, No Duplicates

Bag (multiset) : Unordered, Duplicates allowed.

Set $\{1, 2, 2\} = \{1, 2\} = \{2, 1\}$

Bag $\{1, 2, 2\} \neq \{1, 2\}$

$\{1, 2, 2\}$: a Bag, not a set → 3 items

Note:

Every Set is also a Multiset(bag), But converse is Not true.

Bags: $\{1,2\} = \{2,1\}$

In a Bag, frequency of elements matter.

$\{1,1,2\} \neq \{1,2,2\} \neq \{1,2\} \neq \{1,1,2\}$

Bag $\{1,1,1,2,3,3,3,3\}$ ⟶ 8 items

Example 5.1: The relation in Fig. 5.1 is a bag of tuples. In it, the tuple $(1, 2)$ appears three times and the tuple $(3, 4)$ appears once. If Fig. 5.1 were a set-valued relation, we would have to eliminate two occurrences of the tuple $(1, 2)$. In a bag-valued relation, we *do* allow multiple occurrences of the same tuple, but like sets, the order of tuples does not matter. □

NOT a Set of Tuples

A	B
1	2
3	4
1	2
1	2

a bag of Tuples

4 tuples

Figure 5.1: A bag

Next Topic:

SQL

Very Important NOTE

Structured Query Language (SQL):

SQL treats Table(relation) as a Multiset(Bag). Hence, Duplicate Rows are allowed.

SQL allows the same tuple to appear more than once in a relation.

Set versus bag semantics

- Set
  - No duplicates
  - Relational model and algebra use set semantics
- Bag
  - Duplicates allowed
  - Number of duplicates is significant
  - SQL uses bag semantics by default

Before proceeding, we must point out an *important distinction* between the practical SQL model and the formal relational model discussed in Chapter 5: SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values. Hence, in general, an **SQL** table is not a *set of tuples*, because a set does not allow two identical members; rather, it is a **multiset** (sometimes called a *bag*) of tuples. Some SQL relations are *constrained to be sets* because a key constraint has been declared or because the DISTINCT option has been used with the SELECT statement (described later in this section). We should be aware of this distinction as we discuss the examples.

Next Topic:

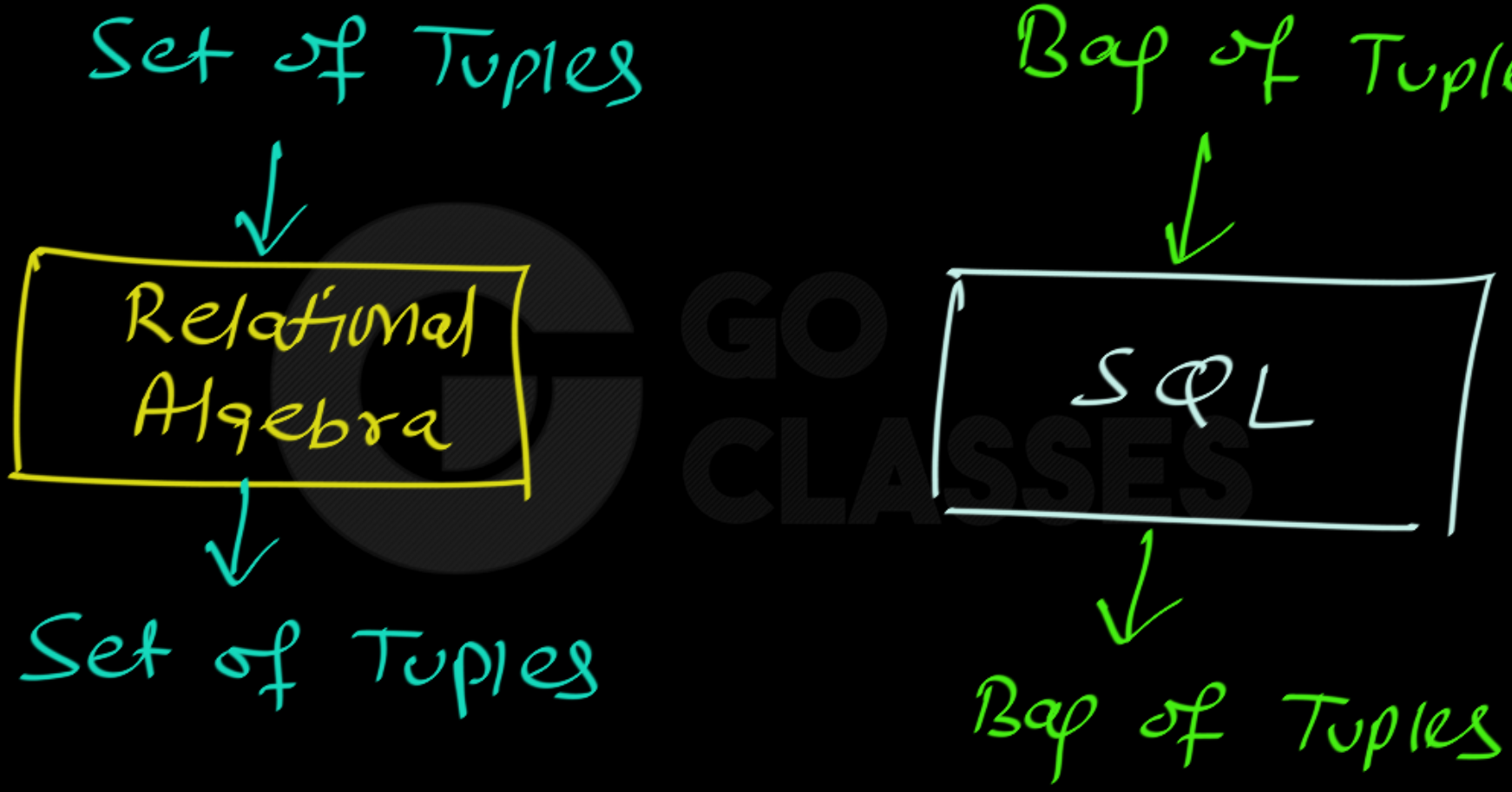
Relational Algebra

Vs

SQL

1. Relational algebra uses Set-semantics, whereas SQL uses Bag-semantics.

i.e. Relational algebra treats Table(relation) as a Set of Tuples (NO Duplication of Tuples)
SQL treats Table(relation) as a Bag of Tuples (Duplication of Tuples allowed)


Set of Tuples
Relational Algebra
Set of Tuples
Bag of Tuples
SQL
Bag of Tuples

2. There are no NULL values in relational algebra.

But SQL table my have Null values.

PS: Outer joins are not considered part of the classical relational algebra.

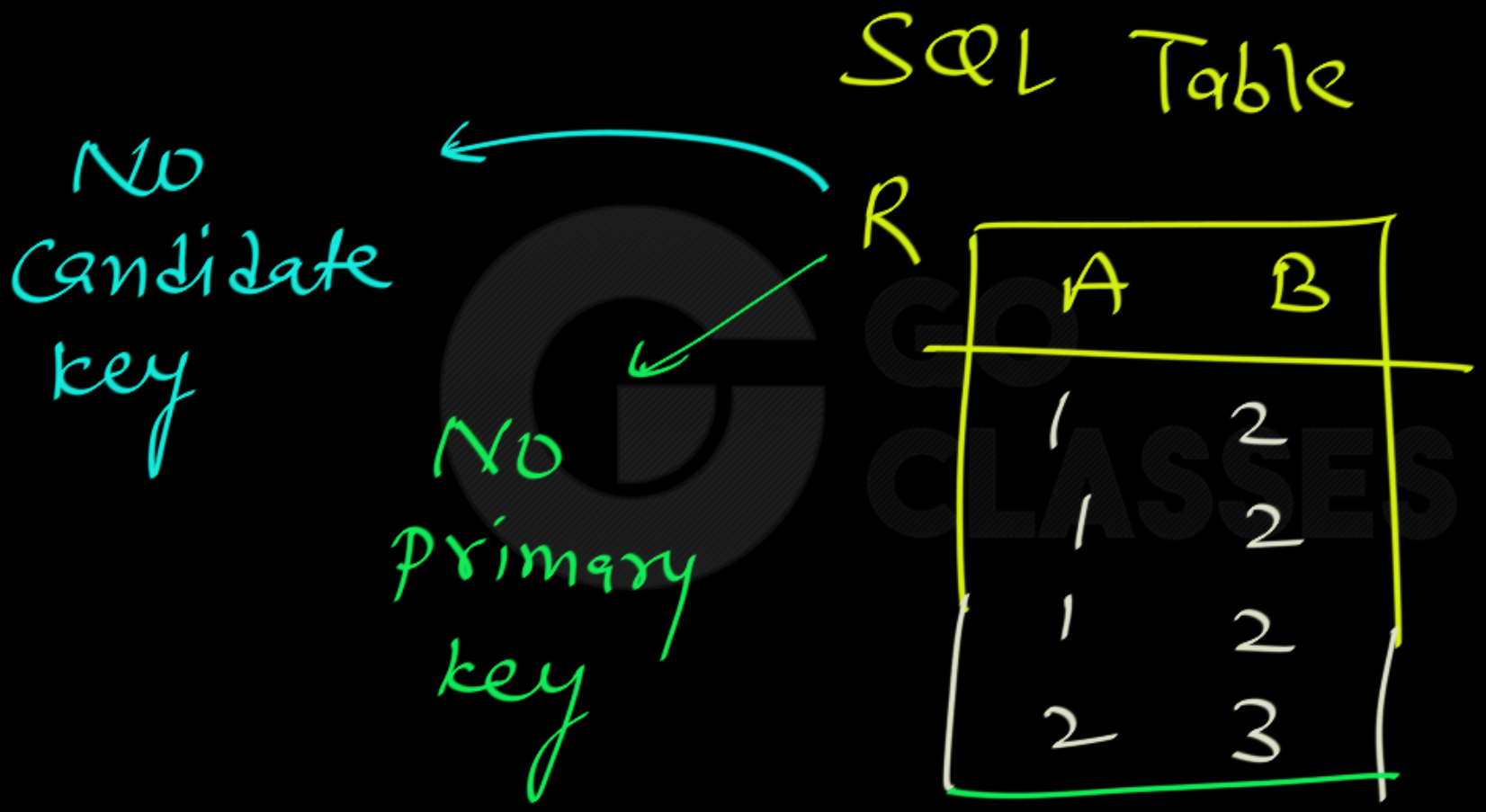
course.ccs.neu.edu/cs3200sp18s3/ssl/lectures/lecture_03_sql_1.pdf

lecture_03_sql_1 19 / 98 118%

*There are actually no concept of NULL in relational algebra

SQL: Part 1 (DML, Relational Algebra)

February 6, 2018


SQL Table
No candidate key
No Primary key
R
A B
1 2
1 2
1 2
2 3

3. In general, an SQL table is not required to have a key... i.e. SQL table may not have a primary key or candidate key.

Relational Algebra table MUST have a Primary Key, Candidate key(s).

[10]In general, an SQL table is not required to have a key, although in most cases there will be one.

NOTE: An SQL table with a key is restricted to being a set, since the key value must be distinct in each tuple.

Relational Algebra

1. Works on pure Relational model.
2. No Duplicate Tuples
3. No Null Values
4. Key : mandatory

SQL

1. Based on Relational model But NOT Exactly.
2. Bag of Tuples
3. Null values allowed
4. Key may not be there.

Next Topic:

SQL

Basic SQL Query

SQL Query

- Basic form (there are many many more bells and whistles)

```
SELECT <attributes>
FROM   <one or more relations>
WHERE  <conditions>
```

Call this a **SFW** query.

Basic SQL Query:

SELECT A_1, A_2, A_3, A_4 → Projection of attributes

FROM R_1, R_2, R_3 // $R_1 \times R_2 \times R_3$

WHERE Condition → for Tuples Selection

SQL Query:

```
SELECT  A, B
FROM    R, S, T
WHERE   C > 5
```

Conceptual Evaluation of Basic SQL Query:

1. $R \times S \times T$
2. $(R \times S \times T)_{C>5}$
3. Project attributes A, B only

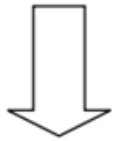
Simple SQL Query: Projection

Product

Projection is the operation of producing an output table with tuples that have a subset of their prior attributes

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT Pname, Price, Manufacturer
FROM   Product
WHERE  Category = 'Gadgets'
```



PName	Price	Manufacturer
Gizmo	$19.99	GizmoWorks
Powergizmo	$29.99	GizmoWorks

Note:

If we want to project ALL attributes then use

```
SELECT *
FROM R, S
WHERE Condition
```

→ all attributes

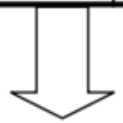
Simple SQL Query: Selection

Product

Selection is the operation of filtering a relation's tuples on some condition

```
SELECT *
FROM   Product
WHERE  Category = 'Gadgets'
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi



PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Show specific rows

"Find all 18 year old students"

students

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

Show specific rows

"Find all 18 year old students"

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

Students

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."

Show specific rows

"Find all 18 year old students"

alias / Tuple variable / Range variable / Renaming

```
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."

$\pi_{gid} Member$

Q: Find Number of tuples in the output of both queries.

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov

```
SELECT gid
FROM Member;
```

Q: Find Number of tuples in the output of both queries.

Relational Algebra Query ← (1) π_{gid} Member

Projection

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov

o/p:

gid
dps
gov
abc

3 Tuples

Q:

Find Number of tuples in the output of both queries.

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov

```
SELECT gid
FROM Member;
```

o/p:

gid
dps
gov
abc
gov
abc
gov

6 Tuples

Set versus bag example

Member

uid	*gid*
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

$\pi_{gid} Member$

gid
dps
gov
abc
…

```
SELECT gid
FROM Member;
```

gid
dps
gov
abc
gov
abc
gov
…

To get DISTINCT tuples in output:

Use the keyword DISTINCT in the SELECT clause, meaning that only distinct tuples should remain in the result.

The basic form of an SQL query is as follows:

```
SELECT    DISTINCT  select-list
FROM      from-list
WHERE     qualification
```

Set versus bag example


```
SELECT DISTINCT gid
FROM Member;
```

gid
dps
gov
abc

3 Tuples

Member

uid	gid
142	dps
123	gov
857	abc
857	gov
456	abc
456	gov
…	…

$\pi_{gid} Member$

gid
dps
gov
abc
…

```
SELECT gid
FROM Member;
```

gid
dps
gov
abc
gov
abc
gov
…

NOTE:

The SELECT and FROM clauses are mandatory in all SQL queries.

The WHERE clause is optional.

Show specific columns

"Find name and login for all students"

Students

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

Show specific columns

"Find name and login for all students"

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

Alias

Students S

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

This is called: "**Project** name and login from table Students"

NOTE:

SQL is Case-Insensitive.

1. SELECT = Select = SELeCt } Keywords
2. FROm = from = fRom
3. Students = Students = STUdENTs } Table
4. age = Age = AGE } attributes

NOTE:

SQL is Case-Insensitive... BUT String name is Case-Sensitive & written in Single Quote.

```
SELECT *
FROM   R
WHERE  Sname = 'Ram'
```

→ 1 Tuple

R

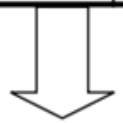
Sname
Ram
Raju
ram

Simple SQL Query: Selection

Selection is the operation of filtering a relation's tuples on some condition

```
SELECT *
FROM   Product
WHERE  Category = 'Gadgets'
```

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi



PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Case Insensitivity

SQL is *case insensitive*, meaning that it treats upper- and lower-case letters as the same letter. For example, although we have chosen to write keywords like `FROM` in capitals, it is equally proper to write this keyword as `From` or `from`, or even `FrOm`. Names of attributes, relations, aliases, and so on are similarly case insensitive. Only inside quotes does SQL make a distinction between upper- and lower-case letters. Thus, `'FROM'` and `'from'` are different character strings. Of course, neither is the keyword `FROM`.

Basic queries: SFW statement

- SELECT $A_1, A_2, \ldots, A_n$
  FROM $R_1, R_2, \ldots, R_m$
  WHERE $condition;$
- Also called an SPJ (select-project-join) query
- Corresponds to (but not really equivalent to) relational algebra query:

$$\pi_{A_1,A_2,\ldots,A_n}\left(\sigma_{condition}(R_1 \times R_2 \times \cdots \times R_m)\right)$$

WHERE keyword for selection of Tuples

SELECT keyword

FROM keyword

The basic form of an SQL query is as follows:

```
SELECT    [ DISTINCT ] select-list
FROM      from-list
WHERE     qualification
```

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following *conceptual evaluation strategy*:
  - Compute the cross-product of the *relation-list*.
  - Select tuples (rows) if they satisfy *qualifications*.
  - Select attributes (columns) in the *target-list*.
  - If DISTINCT is specified, eliminate duplicate rows.

Remember the query and the data

Example 1:

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

Enrolled

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

Students

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
② 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

(2)

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

Example 2:

Details...

Reserves

sid	bid	day
22	101	10/10/96
95	103	11/12/96

Sailors

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

Boats

bid	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
```

Example 2:

Another Join Query

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

o/p:

Sname
Bob

1 Tuple

Details...

Reserves

sid	bid	day
~~22~~	~~101~~	~~10/10/96~~
95	103	11/12/96

Sailors

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

Boats

	bname	color
101	[illegible]	
102	[illegible]	
[illegible]	[illegible]pper	gre[illegible]
[illegible]04	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	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
95	Bob	3	63.5	22	101	10/10/96
95	**Bob**	3	63.5	95	103	11/12/96

Q:

Write a SQL query to output table R as it is. (Write an SQL query to Read table R)

Q:

Write a SQL query to output table R as it is. (Write an SQL query to Read table R)

A: R ✗

B: FROM R ✗

C: SELECT * FROM R ✓

D: SELECT DISTINCT * FROM R ✗

Example: reading a table

- `SELECT * FROM User;`
  - Single-table query, so no cross product here
  - `WHERE` clause is optional
  - `*` is a short hand for "all columns"

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
```

A Trick for Reading and Writing Queries

It is generally easist to examine a select-from-where query by first looking at the `FROM` clause, to learn which relations are involved in the query. Then, move to the `WHERE` clause, to learn what it is about tuples that is important to the query. Finally, look at the `SELECT` clause to see what the output is. The same order — from, then where, then select — is often useful when writing queries of your own, as well.

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'
```

Enrolled

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

Students

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 relation names

possibly with a *range-variable* after each name

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

qualification : Comparisons combined using AND, OR and NOT.

Comparisons are Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$

DISTINCT: optional keyword indicating that the answer should not contain duplicates.

In SQL SELECT, the default is that duplicates are *not* eliminated! (Result is called a "multiset")

Next Topic:

SQL

An Important Point (small Detour)

SQL

How we understand things..

Vs

How they actually happen..

Q:

We are saying that "SQL is Declarative, Not procedural."

But we are also providing the "Evaluation Order" for SQL query, like we do for Relational Algebra..

Isn't it Contradictory??

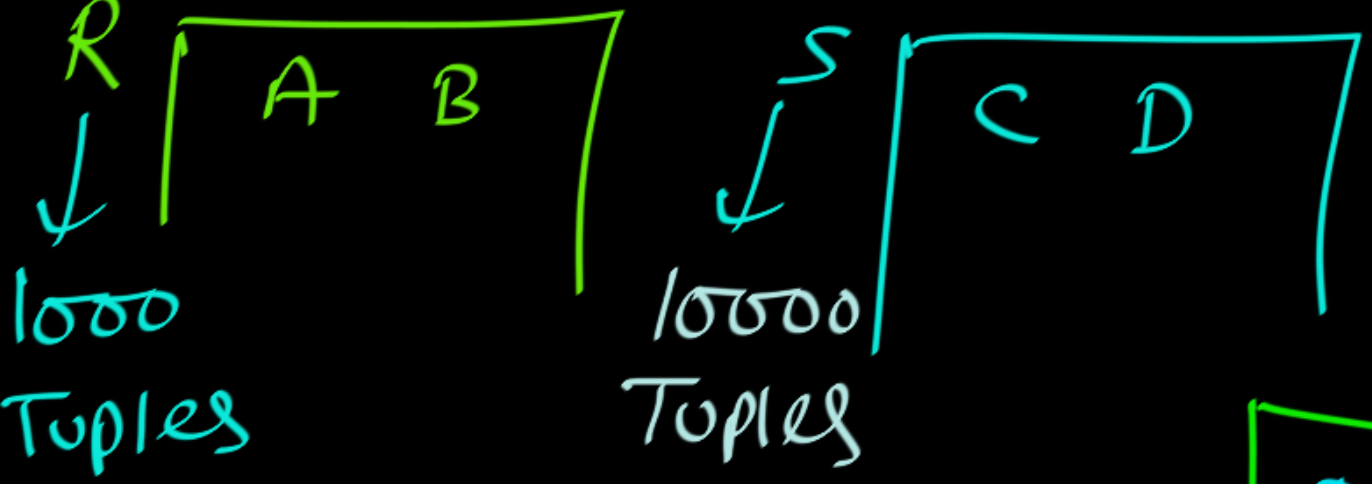
SQL : Declarative

SQL Evaluation order

1. FROM
2. WHERE
3. SELECT

"Just a Conceptual Evaluation" for our understanding of Query

DBMS will NOT Execute Like this.



2 Tuples in S with D > 6

3 Tuples in R with B = 5

```
SELECT *
FROM R, S
WHERE B = 5 and D > 6
```

DUMB DBMS:

1. R x S (10^7 Tuples)
2. B = 5 and D > 6
3. Project all attributes

The Evaluation of SQL query that we study is a "Conceptual Evaluation"...
NOT how actually the query would be evaluated by RDBMS.
DBMS will evaluate the query in different & much efficient ways.

5.2 THE FORM OF A BASIC SQL QUERY

This section presents the syntax of a simple SQL query and explains its meaning through a *conceptual evaluation strategy*. A conceptual evaluation strategy is a way to evaluate the query that is intended to be easy to understand rather than efficient. A DBMS would typically execute a query in a different and more efficient way.

The basic form of an SQL query is as follows:

```
SELECT  [DISTINCT] select-list
FROM    from-list
WHERE   qualification
```

Query Semantics

Conceptually, a SQL query can be computed:

1. **FROM** : compute *cross-product* of tables (e.g., Students and Enrolled).
2. **WHERE** : Check conditions, discard tuples that fail. (called "selection").
3. **SELECT** : Delete unwanted fields. (called "projection").
4. If **DISTINCT** specified, eliminate duplicate rows.

Probably the least efficient way to compute a query!

Query Optimization helps us find more efficient strategies to get the *same answer*.

HW 1:

Number of tuples in the output ?

```
SELECT actor, birth, movie
FROM Role, Person
WHERE actor = name and birth > 1940;
```

Role

actor	movie	persona
Ben Affleck	Argo	Tony Mendez
Alan Arkin	Argo	Lester Siegel
Ben Affleck	The Company Men	Bobby Walker
Tommy Lee Jones	The Company Men	Gene McClary

Person

name	birth	city
Ben Affleck	1972	Berkeley
Alan Arkin	1934	New York
Tommy Lee Jones	1946	San Saba

HW 2:

Number of tuples in the output ?

```
SELECT award, actor, persona, Role.movie
FROM Honours, Role
WHERE category = 'actor' AND winner = actor
      AND Honours.movie = Role.movie
```

Honours

movie	award	category	winner
Lincoln	Critic's Choice	actor	Daniel Day-Lewis
Argo	Critic's Choice	director	Ben Affleck
Lincoln	SAG	supporting actor	Tommy Lee Jones
Lincoln	Critic's Choice	screenplay	Tony Kushner
War Horse	BMI Flim	music	John Williams

Role

actor	movie	persona
Ben Affleck	Argo	Tony Mendez
Tommy Lee Jones	Lincoln	Thaddeus Stevens
Daniel Day-Lewis	The Boxer	Danny Flynn
Daniel Day-Lewis	Lincoln	Abraham Lincoln

2-RELATION SELECT-FROM-WHERE


```
SELECT award, actor, persona, Role.movie
FROM Honours, Role
WHERE category = 'actor' AND winner = actor
      AND Honours.movie = Role.movie
```

Honours

movie	award	category	winner
Lincoln	Critic's Choice	actor	Daniel Day-Lewis
Argo	Critic's Choice	director	Ben Affleck
Lincoln	SAG	supporting actor	Tommy Lee Jones
Lincoln	Critic's Choice	screenplay	Tony Kushner
War Horse	BMI Flim	music	John Williams

Role

actor	movie	persona
Ben Affleck	Argo	Tony Mendez
Tommy Lee Jones	Lincoln	Thaddeus Stevens
Daniel Day-Lewis	The Boxer	Danny Flynn
Daniel Day-Lewis	Lincoln	Abraham Lincoln

Honours.movie	award	category	winner	actor	Role.movie	persona	
Lincoln	Critic's Choice	actor	Daniel Day-Lewis	Ben Affleck	Argo	Tony Mendez	×
Lincoln	Critic's Choice	actor	Daniel Day-Lewis	Tommy Lee Jones	Lincoln	Thaddeus Stevens	×
Lincoln	Critic's Choice	actor	Daniel Day-Lewis	Daniel Day-Lewis	The Boxer	Danny Flynn	×
Lincoln	Critic's Choice	actor	Daniel Day-Lewis	Daniel Day-Lewis	Lincoln	Abraham Lincoln	☑
Argo	Critic's Choice	director	Ben Affleck	Ben Affleck	Argo	Tony Mendez	×
Argo	Critic's Choice	director	Ben Affleck	Tommy Lee Jones	Lincoln	Thaddeus Stevens	×
Argo	Critic's Choice	director	Ben Affleck	Daniel Day-Lewis	The Boxer	Danny Flynn	×
Argo	Critic's Choice	director	Ben Affleck	Daniel Day-Lewis	Lincoln	Abraham Lincoln	×
Lincoln	SAG	supporting actor	Tommy Lee Jones	Ben Affleck	Argo	Tony Mendez	×
Lincoln	SAG	supporting actor	Tommy Lee Jones	Tommy Lee Jones	Lincoln	Thaddeus Stevens	×
Lincoln	SAG	supporting actor	Tommy Lee Jones	Daniel Day-Lewis	The Boxer	Danny Flynn	×
...							

Next Topic:

SQL

Writing & Understanding SQL Queries

Writing & Understanding SQL Query:

Step 1: Look at the given database & find out the relations we need to get the desired information.

Step2:

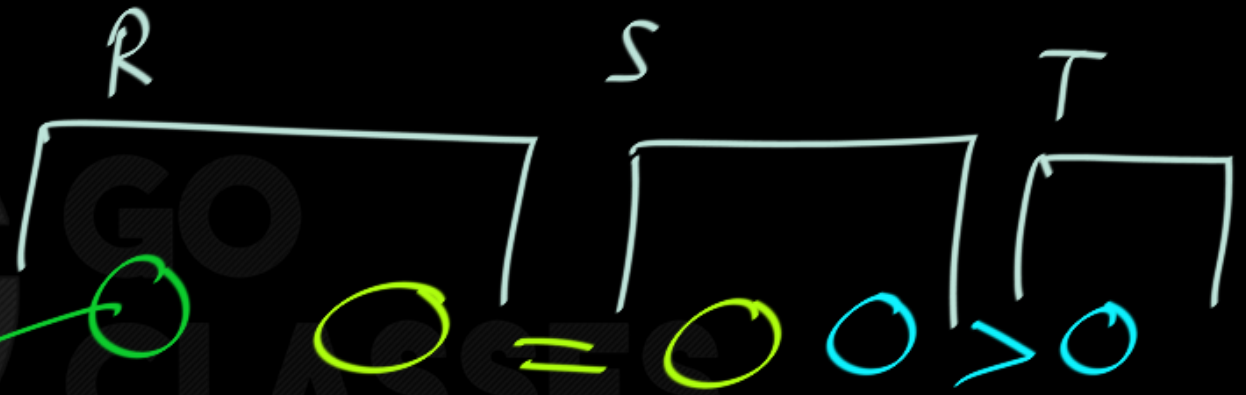
Draw tables side by side & think about the Idea...Do the mapping.

Step3: Implement that Idea.

Query:

SELECT - -
FROM R, S, T
WHERE -

Idea:



projected in o/p

Q 15: Find the names and ages of all sailors.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

Q 15: Find the names and ages of all sailors.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

```
SELECT  S.sname, S.age
FROM    sailors S
```

Q 15: Find the names and ages of all sailors.

```
SELECT sname, age
FROM Sailors
```

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 5.1 An Instance $S3$ of Sailors

10 Tuples in o/p

The answer to this query with and without the keyword DISTINCT on instance $S3$ of Sailors is shown in Figures 5.4 and 5.5. The only difference is that the tuple for Horatio appears twice if DISTINCT is omitted; this is because there are two sailors called Horatio and age 35.

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Art	25.5
Bob	63.5

Figure 5.4 Answer to Q15 with DISTINCT

sname	*age*
Dustin	45.0
Brutus	33.0
Lubber	55.5
Andy	25.5
Rusty	35.0
Horatio	35.0
Zorba	16.0
Horatio	35.0
Art	25.5
Bob	63.5

Figure 5.5 Answer to Q15 without DISTINCT

Q 11: Find all sailors with a rating above 7.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

(Q11) Find all sailors with a rating above 7.

```
SELECT  S.sid, S.sname, S.rating, S.age
FROM    Sailors AS S
WHERE   S.rating > 7
```

≡ SELECT *

This query uses the optional keyword **AS** to introduce a range variable. Incidentally,

(Q1) Find the names of sailors who have reserved a boat number 103.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

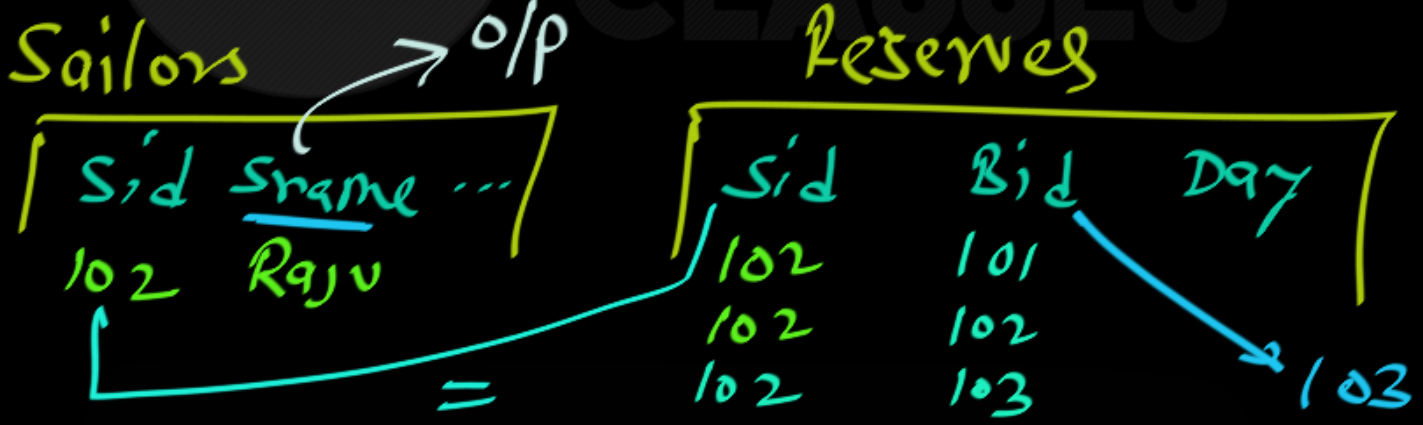
(Q1) Find the names of sailors who have reserved a boat number 103.

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Sailors

Reserves

IDEA :



```
SELECT   S.Sname
FROM     Sailors S, Reserves R
WHERE    S.sid = R.sid and   R.bid = 103
```

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

An equivalent way to write this query is:

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

Q : Find sailor Sid's who have reserved at least one boat.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

Q: Find sailor Sid's who have reserved at least one boat.

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

```
SELECT Sid
FROM Reserves
```

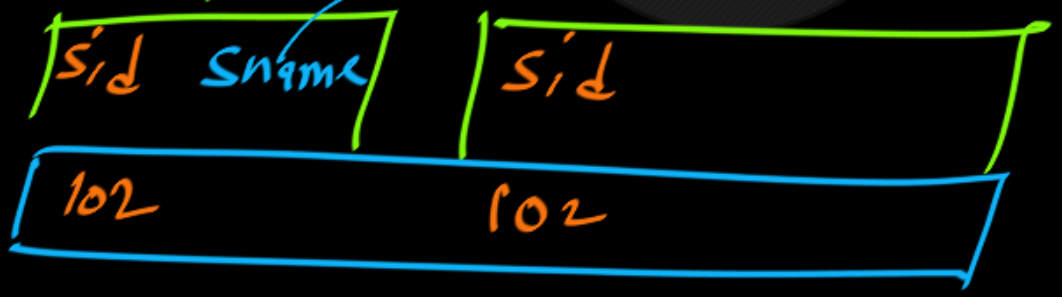
Q : Find sailor names who have reserved at least one boat.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

Q: Find sailor names who have reserved at least one boat.

Sailors

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)




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

(Q2) Find the names of sailors who have reserved a red boat.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

(Q2) Find the names of sailors who have reserved a red boat.

Sailors

Boats

- Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
- Boats(*bid:* integer, *bname:* string, *color:* string)
- Reserves(*sid:* integer, *bid:* integer, *day:* date)

Idea:

Sailors → o/p

Sailors		Reserves		Boats	
Sid	Sname	Sid	Bid	Bid	Color
102		102	2	2	Red

(Q2) Find the names of sailors who have reserved a red boat.

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

This query contains a join of three tables followed by a selection on the color of boats. The join with Sailors allows us to find the name of the sailor who, according to Reserves tuple R, has reserved a red boat described by tuple B.

Q: Output of the following Query ??

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

A. Sailors who reserved all red boats.
B. Sailors who reserved at least one red boat.
C. Sailors who reserved no red boats.

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Q: Output of the following Query ??

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

This Condition is Checked for Every Tuple independently.

A. Sailors who reserved all red boats.

B. Sailors who reserved at least one red boat.

C. Sailors who reserved no red boats.

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Q: What is the output of the following SQL query ?

```
SELECT  x.sname, x.age
FROM Sailors x, Sailors y
WHERE  x.age > y.age
```

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

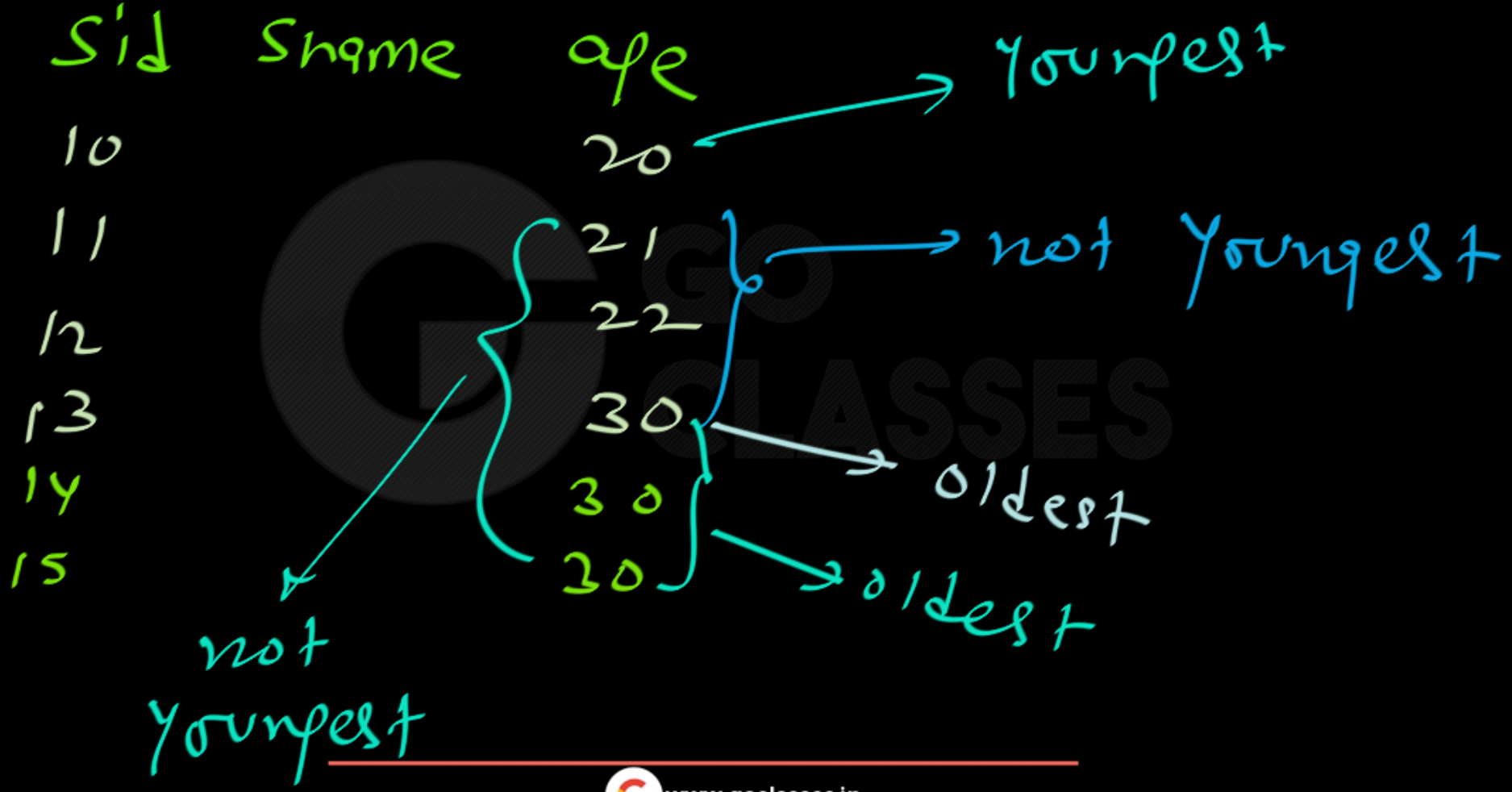
```
SELECT  x.sname, x.age
FROM Sailors x, Sailors y
WHERE  x.age > y.age
```

A. Name & Age of Youngest Sailors
B. Name & Age of Oldest Sailors
C. Name & Age of Sailors who are Not Youngest
D. Name & Age of Sailors who are Not Oldest

```
SELECT  x.sname, x.age
FROM Sailors x, Sailors y
WHERE  x.age > y.age
```

o/p : name & age of those sailors who are older than at least one person.

A. Name & Age of Youngest Sailors
B. Name & Age of Oldest Sailors
C. Name & Age of Sailors who are Not Youngest
D. Name & Age of Sailors who are Not Oldest

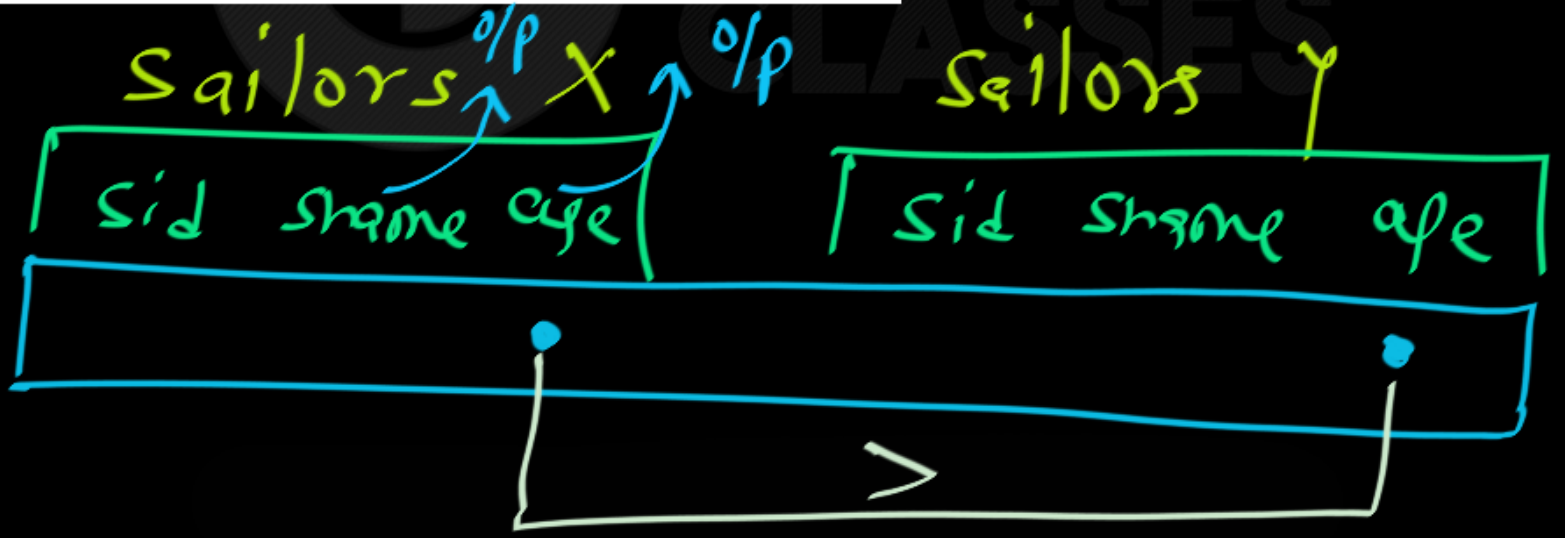

Sid
Sname
age
10
11
12
13
14
15
20
21
22
30
30
30
youngest
not youngest
oldest
oldest
not youngest

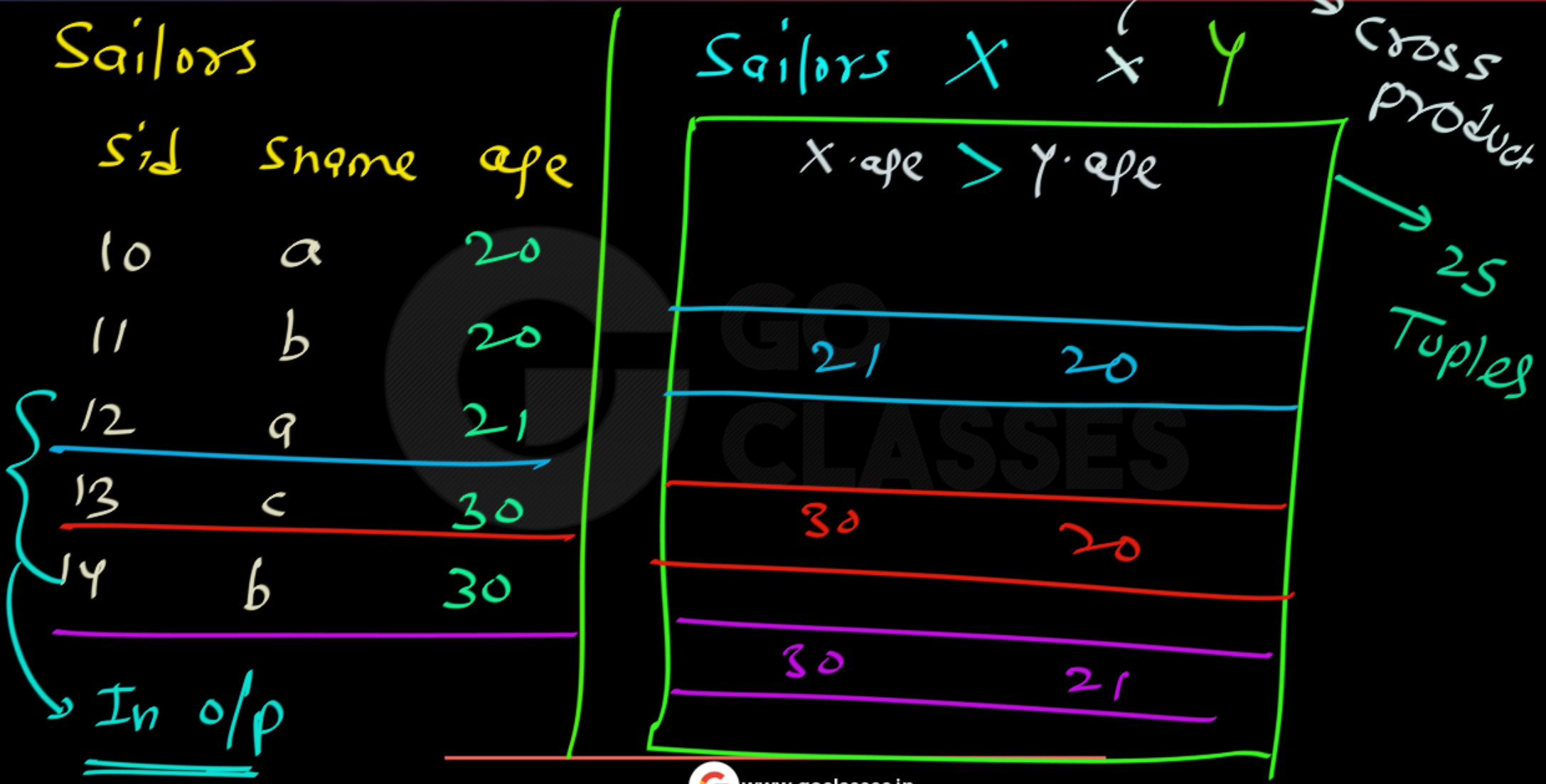
Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

```
SELECT  x.sname, x.age
FROM Sailors x, Sailors y
WHERE  x.age > y.age
```

→ Cross product

IDEA:



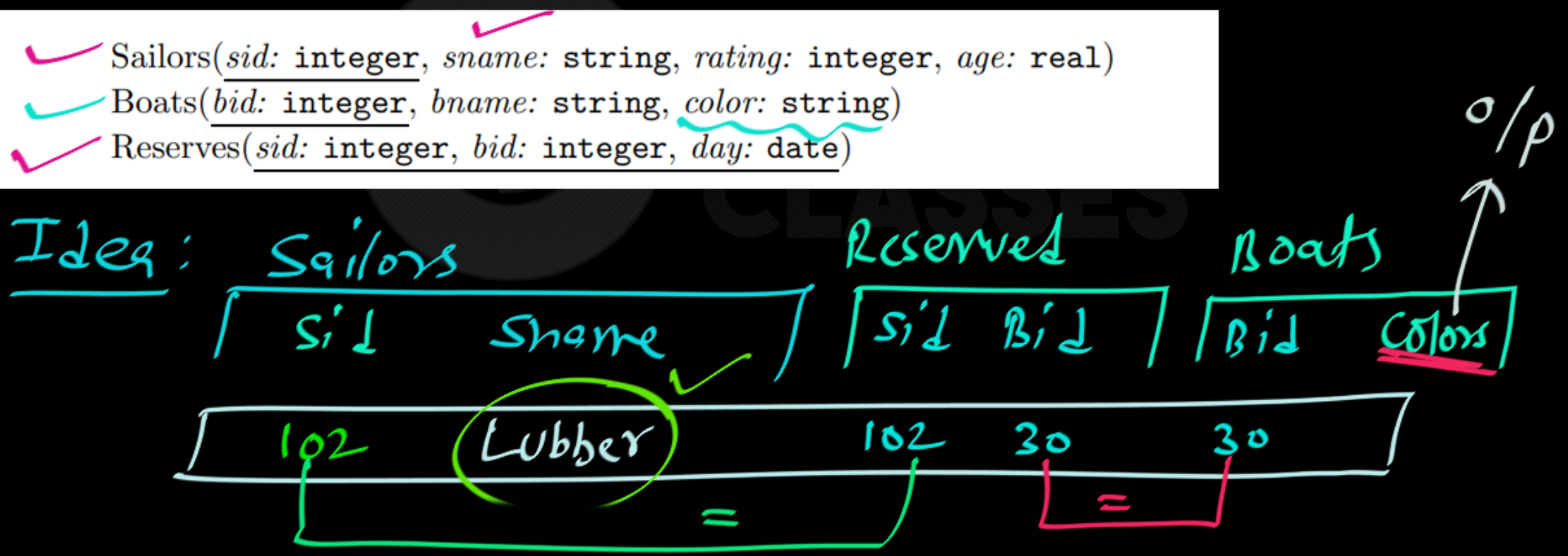

Sailors
sid sname age
10 a 20
11 b 20
12 a 21
13 c 30
14 b 30
In o/p
Sailors X X Y
Cross product
X.age > Y.age
25 Tuples
21 20
30 20
30 21

(Q3) Find the colors of boats reserved by Lubber.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

(Q3) Find the colors of boats reserved by Lubber.

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)



(Q3) Find the colors of boats reserved by Lubber.

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

This query is very similar to the previous one. Notice that in general there may be more than one sailor called Lubber (since *sname* is not a key for Sailors); this query is still correct in that it will return the colors of boats reserved by *some* Lubber, if there are several sailors called Lubber.

Next Topic:

SQL

Keywords : Union, Intersect, Except

SQL keywords : → Set Semantics

keywords : | Purpose

union | Set Union

intersect | Set intersection

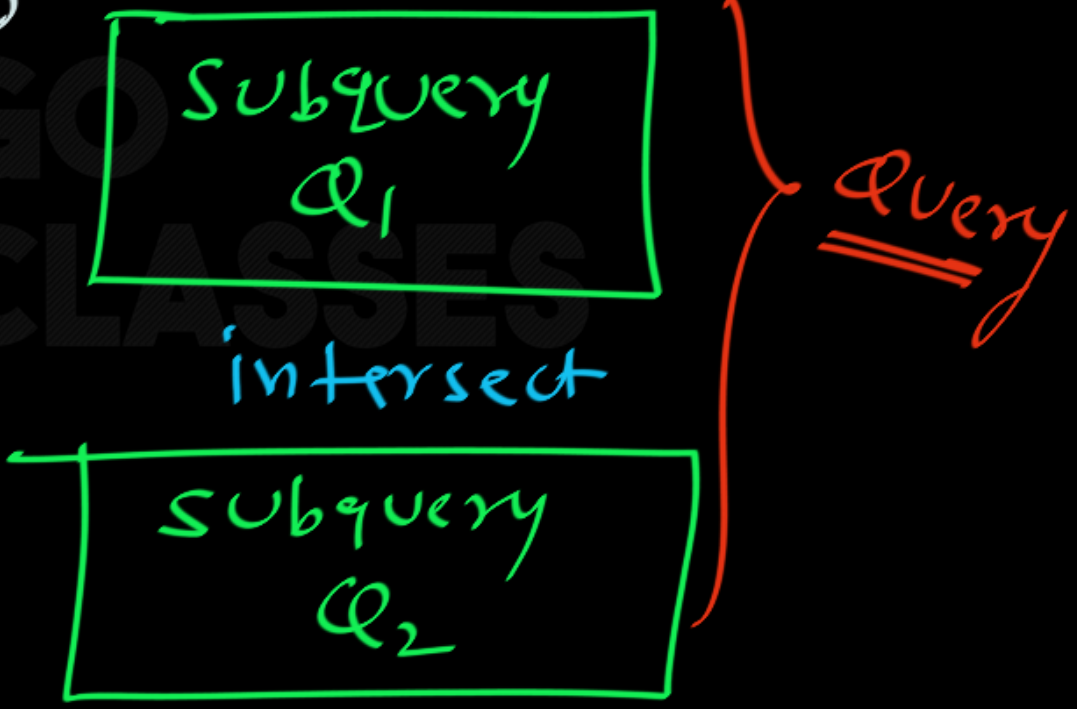
Except / minus | Set set Difference

Except → Standard SQL keyword

minus → oracle DBMS

Query format using union, intersect, Except:

① R union S



Note: To use Union, intersect, Except; Result of subquery Q_1 & Q_2 must be union_compatible.

Union Compatible :

R, S are union Compatible :

(1) Number of attributes in R, S Same.

&

(2) Domain(a) = Domain(d)
Domain(b) = Domain(e)
Domain(c) = Domain(f)

Note that `UNION`, `INTERSECT`, and `EXCEPT` can be used on *any* two tables that are union-compatible, that is, have the same number of columns and the columns, taken in order, have the same types.

- UNION, EXCEPT, INTERSECT
  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
    - Duplicates in result are also eliminated (for UNION)
  - Exactly like set ∪, −, and ∩ in relational algebra

SQL has directly incorporated some of the set operations from mathematical *set theory*, which are also part of relational algebra (see Chapter 8). There are set union (**UNION**), set difference (**EXCEPT**),[11] and set intersection (**INTERSECT**) operations. The relations resulting from these set operations are sets of tuples; that is, *duplicate tuples are eliminated from the result*. These set operations apply only to *type-compatible relations*, so we must make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations. The next example illustrates the use of UNION.

---

[10] In general, an SQL table is not required to have a key, although in most cases there will be one.

[11] In some systems, the keyword MINUS is used for the set difference operation instead of EXCEPT.

(a)

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

R union S

→ Invalid SQL Query

(a)

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

(SELECT * FROM R) ≡ R

(SELECT * FROM S) ≡ S

(a)

R

A
a1
a2
~~a2~~
a3

S

A
a1
a2
a4
a5

Ignore Duplicates in Input Tables

(SELECT * FROM R)
union
(SELECT * FROM S)

Remove Duplicates in o/p

o/p:

A
a_1
a_2
a_3
a_4
a_5

5 Tuples

(a)

R

A
a1
a2
~~a2~~
a3

S

A
a1
a2
a4
a5

Ignore Duplicates in Input Table

(SELECT * FROM R)

Intersect

(SELECT * FROM S)

use set semantics

o/p

2 Tuples

A
a1
a2

R - S

(a)

R

A
a1
a2
~~a2~~
a3

S

A
a1
a2
a4
a5

Ignore Duplicates in Input Table

```
( SELECT *
  FROM   R )
Except
( SELECT *
  FROM   S )
```

use set semantics

o/p

A
a_3

1 Tuples

Next Topic:

SQL Keywords

Union All, Intersect All, Except All

SQL Keywords	Purpose
Union all	Bag Union
Intersect all	Bag intesection
Except all	Bag Difference

Use Bag semantics

SQL Keywords:

UNION : Used for Set Union

(Treats Inputs as Sets & Produces a Set of Tuples)

UNION ALL : Used for Bag Union

(Treats Inputs as Bags & Produces a Bag of Tuples)

SQL Keywords:

INTERSECT : Used for Set Intersection
(Treats Inputs as Sets & Produces a Set of Tuples)

INTERSECT ALL : Used for Bag Intersection
(Treats Inputs as Bags & Produces a Bag of Tuples)

SQL Keywords:

EXCEPT : Used for Set Difference
(Treats Inputs as Sets & Produces a Set of Tuples)

EXCEPT ALL : Used for Bag Difference
(Treats Inputs as Bags & Produces a Bag of Tuples)

Next Topic:

SQL

Bag Union, Bag Intersection, Bag Difference

Bag union : frequency union

Bag intersection : frequency intersection

Bag Difference : frequency difference

Bag Union:

Bag

Bag

{1,2,1} Bag Union {1,1,2,3,1}

= {1,1,1,1,1,2,2,3}

Bag Intersection:

$\{1,2,1\}$ Bag Intersection $\{1,2,3\}$

$= \{1,2\}$

Bag Difference:

{1,2,1} Bag Minus {1,2,3}

= {1}

$\{1,2,3\} - \{1,2,1\}$

$= \{ \quad 3 \quad \}$

Result: Bag R = 'm' Copies of element a

Bag S = 'n' Copies of element a

In "R Bag union S": m+n Copies of element a

(frequency union)

Result: Bag R = 'm' Copies of element a

Bag S = 'n' Copies of element a

In "R Bag intersection S": min(m, n) Copies of element a.

(frequency intersection)

Result: Bag R = 'm' Copies of element a

Bag S = 'n' Copies of element a

In "R Bag Difference S": max(0, m-n) Copies of element a.

frequency minus

5.1.2 Union, Intersection, and Difference of Bags

These three operations have new definitions for bags. Suppose that R and S are bags, and that tuple t appears n times in R and m times in S. Note that either n or m (or both) can be 0. Then:

- In the bag union $R \cup S$, tuple t appears $n + m$ times.
- In the bag intersection $R \cap S$, tuple t appears $\min(n, m)$ times.
- In the bag difference $R - S$, tuple t appears $\max(0, n - m)$ times. That is, if tuple t appears in R more times than it appears in S, then t appears in $R - S$ the number of times it appears in R, minus the number of times it appears in S. However, if t appears at least as many times in S as it appears in R, then t does not appear at all in $R - S$. Intuitively, occurrences of t in S each "cancel" one occurrence in R.

Next Topic:

SQL Keywords

Union All, Intersect All, Except All

UNION ALL ; INTERSECT ALL; EXCEPT ALL :

to note about UNION, INTERSECT, and EXCEPT follows. In contrast to the default that duplicates are not eliminated unless DISTINCT is specified in the basic query form, the default for UNION queries is that duplicates *are* eliminated! To retain duplicates, UNION ALL must be used; if so, the number of copies of a row in the result is $m + n$, where m and n are the numbers of times that the row appears in the two parts of the union. Similarly, one version of INTERSECT retains duplicates—the number of copies of a row in the result is $min(m, n)$—and one version of EXCEPT also retains duplicates—the number of copies of a row in the result is $m - n$, where m corresponds to the first relation.

o/p

(a)

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

(b)

T

A
a1
a1
a2
a2
a2
a3
a4
a5

(c)

T

A
a2
a3

(d)

T

A
a1
a2

Query Q_1

(SELECT * FROM R

Union all

SELECT * FROM S)

(a)

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

(b)

T

A
a1
a1
a2
a2
a2
a3
a4
a5

(c)

T

A
a2
a3

(d)

T

A
a1
a2

Figure 6.5
The results of SQL multiset operations. (a) Two tables, R(A) and S(A).
(b) R(A)UNION ALL S(A).
(c) R(A) EXCEPT ALL S(A).
(d) R(A) INTERSECT ALL S(A).

SQL set and bag operations

- `UNION`, `EXCEPT`, `INTERSECT`
  - Set semantics
    - Duplicates in input tables, if any, are first eliminated
    - Duplicates in result are also eliminated (for `UNION`)
  - Exactly like set $\cup$, $-$, and $\cap$ in relational algebra
- `UNION ALL`, `EXCEPT ALL`, `INTERSECT ALL`
  - Bag semantics
  - Think of each row as having an implicit count (the number of times it appears in the table)
  - Bag union: sum up the counts from two tables
  - Bag difference: proper-subtract the two counts
  - Bag intersection: take the minimum of the two counts

6.4.2 Duplicates in Unions, Intersections, and Differences

Unlike the SELECT statement, which preserves duplicates as a default and only eliminates them when instructed to by the DISTINCT keyword, the union, intersection, and difference operations, which we introduced in Section 6.2.5, normally eliminate duplicates. That is, bags are converted to sets, and the set version of the operation is applied. In order to prevent the elimination of duplicates, we must follow the operator UNION, INTERSECT, or EXCEPT by the keyword ALL. If we do, then we get the bag semantics of these operators as was discussed in Section 5.1.2.

- `UNION`, `EXCEPT`, `INTERSECT` eliminate duplicates (set semantics)
  - Exactly like set $\cup$, $-$, $\cap$
- `UNION ALL`, `EXCEPT ALL`, `INTERSECT ALL` retain duplicates (bag semantics)
  - Bag union: sum the times an element appears in the two bags
  - Bag difference: proper-subtract the times an element appears in the two bags
  - Bag intersection: take the minimum of the times an element appears in the two bags

$\rightsquigarrow$ Oracle calls difference `MINUS` instead of `EXCEPT`

Examples of bag operations

Bag1

fruit
apple
apple
orange

Bag2

fruit
apple
orange
orange

```
(SELECT * FROM Bag1)
UNION ALL
(SELECT * FROM Bag2);
```

fruit
apple
apple
orange
apple
orange
orange

```
(SELECT * FROM Bag1)
EXCEPT ALL
(SELECT * FROM Bag2);
```

fruit
apple

```
(SELECT * FROM Bag1)
INTERSECT ALL
(SELECT * FROM Bag2);
```

fruit
apple
orange

UGC NET CSE | August 2016 | Part 3 | Question: 11

asked Sep 30, 2016 • edited Jul 24 by **Deepak Poonia**

951 views

3

Consider the following ORACLE relations :

One $(x, y) = \{< 2, 5 >, < 1, 6 >, < 1, 6 >, < 1, 6 >, < 4, 8 >, < 4, 8 >\}$

Two $(x, y) = \{< 2, 55 >, < 1, 1 >, < 4, 4 >, < 1, 6 >, < 4, 8 >, < 4, 8 >, < 9, 9 >, < 1, 6 >\}$

Consider the following two SQL queries $SQ1$ and $SQ2$:

Consider the following two SQL queries $SQ1$ and $SQ2$:

SQ1 :

```
SELECT * FROM One)

    EXCEPT

(SELECT * FROM Two);
```

SQ2 :

```
SELECT * FROM One)

    EXCEPT ALL

(SELECT * FROM Two);
```

For each of the SQL queries, what is the cardinality (number of rows) of the result obtained when applied to the instances above ?

A. 2 and 1 respectively
B. 1 and 2 respectively
C. 2 and 2 respectively
D. 1 and 1 respectively

Consider the following two SQL queries $SQ1$ and $SQ2$:

SQ1 :

```
SELECT * FROM One)
EXCEPT
(SELECT * FROM Two);
```

≡ one(x,y) Except two(x,y)

Except → set difference

SQ2 :

```
SELECT * FROM One)
EXCEPT ALL
(SELECT * FROM Two);
```

≡ one(x,y) Except all two(x,y)

Except all → Bag difference

Except → frequency minus

For each of the SQL queries, what is the cardinality (number of rows) of the result obtained when applied to the instances above ?

A. 2 and 1 respectively
B. 1 and 2 respectively ✓
C. 2 and 2 respectively
D. 1 and 1 respectively

UGC NET CSE | August 2016 | Part 3 | Question: 11

Consider the following ORACLE relations :

3

One $(x, y) = \{< 2,5 >, < 1,6 >, < 1,6 >, < 1,6 >, < 4,8 >, < 4,8 >\}$

Two $(x, y) = \{< 2,55 >, < 1,1 >, < 4,4 >, < 1,6 >, < 4,8 >, < 4,8 >, < 9,9 >, < 1,6 >\}$

SQ1 : Except : $\{<2,5>\}$

SQ1 : o/p has 1 Tuple.

One – two :

UGC NET CSE | August 2016 | Part 3 | Question: 11

Consider the following ORACLE relations :

3

One $(x, y) = \{< 2,5 >, < 1,6 >, < 1,6 >, < 1,6 >, < 4,8 >, < 4,8 >\}$

Two $(x, y) = \{< 2,55 >, < 1,1 >, < 4,4 >, < 1,6 >, < 4,8 >, < 4,8 >, < 9,9 >, < 1,6 >\}$

$\{<2,5>, <1,6>\}$

SQ2 : Except all

SQ2 : 2 Tuples ✓

Next Topic:

SQL

SQL Queries using Union, Intersect, Except

Q: Output of the following Query ??

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

A. Names of sailors who have reserved a red or a green boat.

B. Names of sailors who have reserved a red and a green boat.

C. Empty Table

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Q: Output of the following Query ??

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

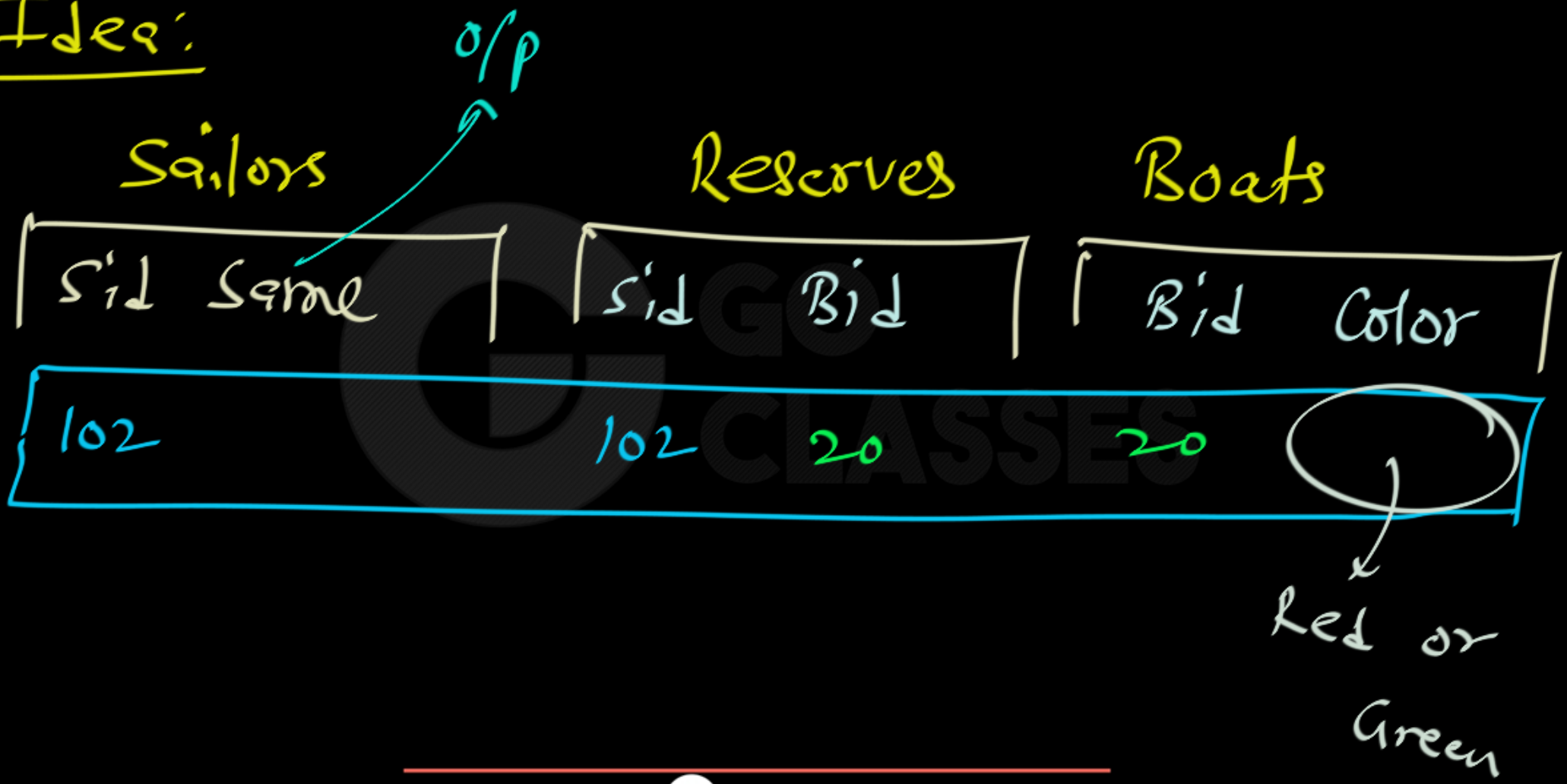
A. Names of sailors who have reserved a red or a green boat.

B. Names of sailors who have reserved a red and a green boat.

C. Empty Table

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Idea:


o/p
Sailors
Reserves
Boats
Sid
Name
Sid
Bid
Bid
Color
102
102
20
20
Red or
Green

Q: Output of the following Query ??

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

A. Names of sailors who have reserved a red and a green boat.

B. Names of sailors who have reserved a red or a green boat.

C. Empty Table

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Q: Output of the following Query ??

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

Tuple selection Condition

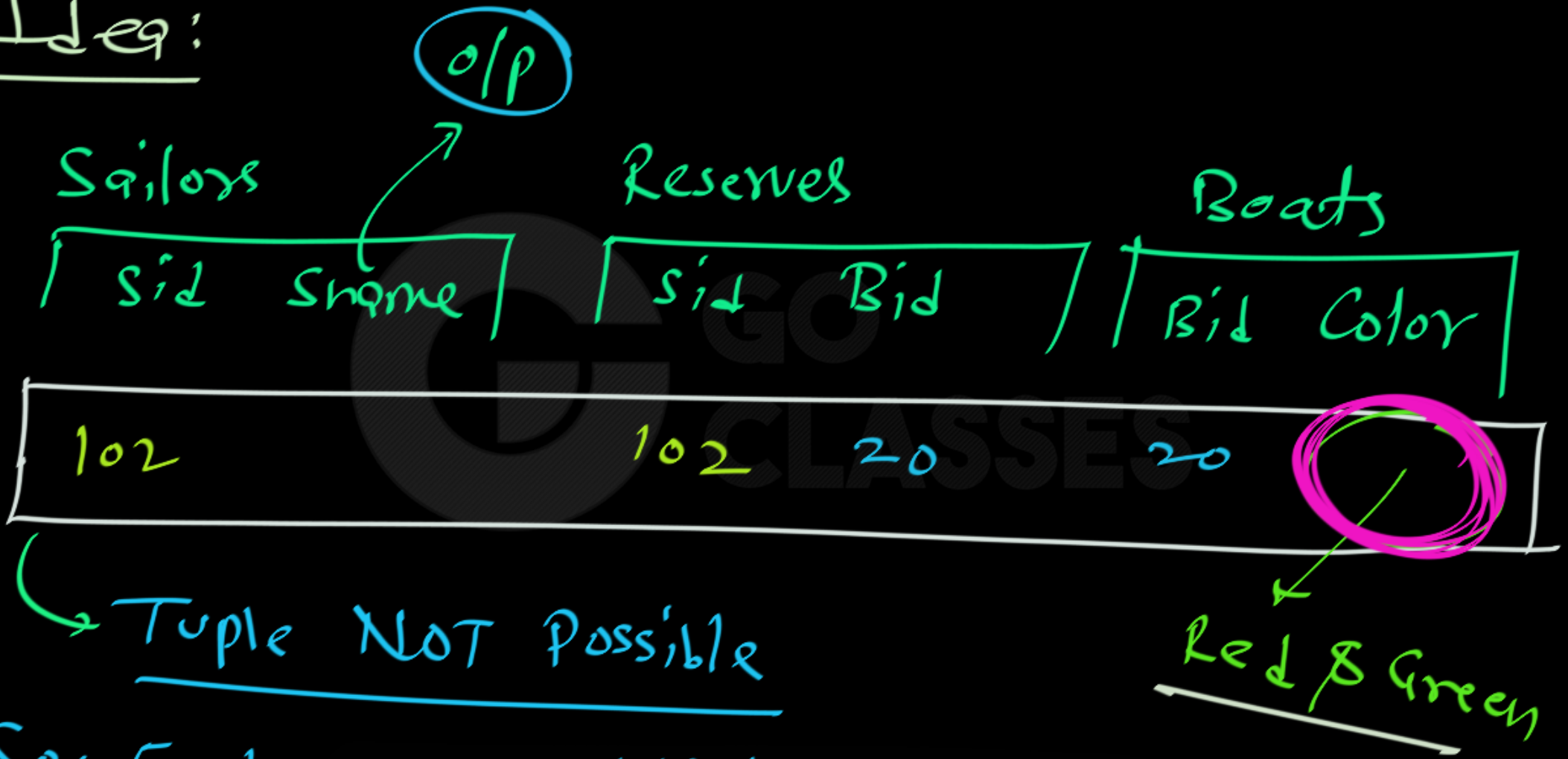
A. Names of sailors who have reserved a red and a green boat.

B. Names of sailors who have reserved a red or a green boat.

C. Empty Table ✓

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Ideq:



→ Tuple NOT Possible

So; Empty Table output.


o/p
Sailors
Sid Sname
Reserves
Sid Bid
Boats
Bid Color
Reserves
Sid Bid
Boats
Bid Color
102
102 20
20 Red
102 25
25 Green
=
=
=
=
Idea: ✓

Names of sailors who have reserved a red and a green boat.

Correct SQL query:

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

Q: Is the following Query output Correct ??

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
UNION
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Names of sailors who have reserved a red or a green boat.

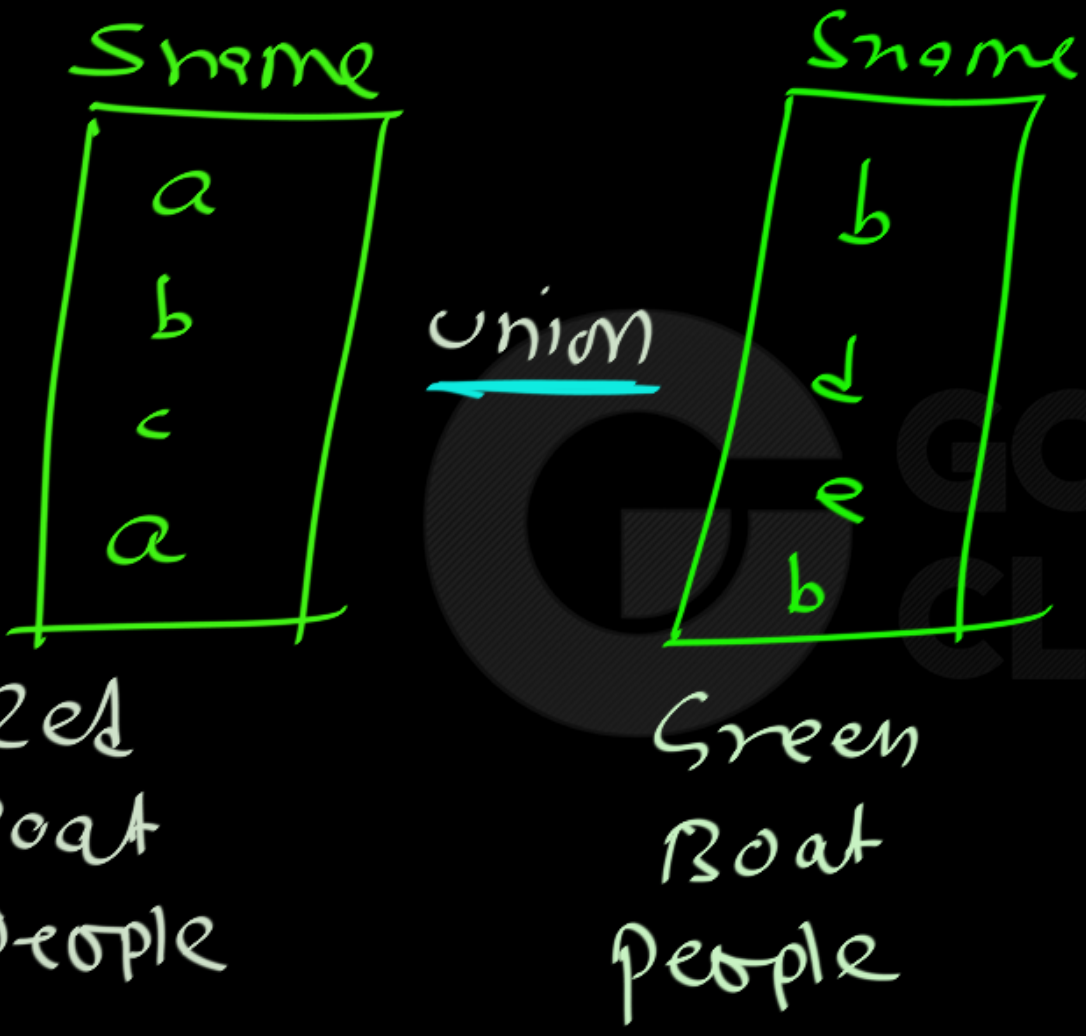
Q: Is the following Query output Correct ?? Yes

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
UNION
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Sailor names booking a Red Boat

→ Sailor names booking a Green Boat

Names of sailors who have reserved a red or a green boat. ✓



= all sailors names who reserves a Red or a Green boat.

Q: Is the following Query output Correct ??

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Names of sailors who have reserved a red and a green boat.

Q: Is the following Query output Correct ?? No

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Names of Red Boat People

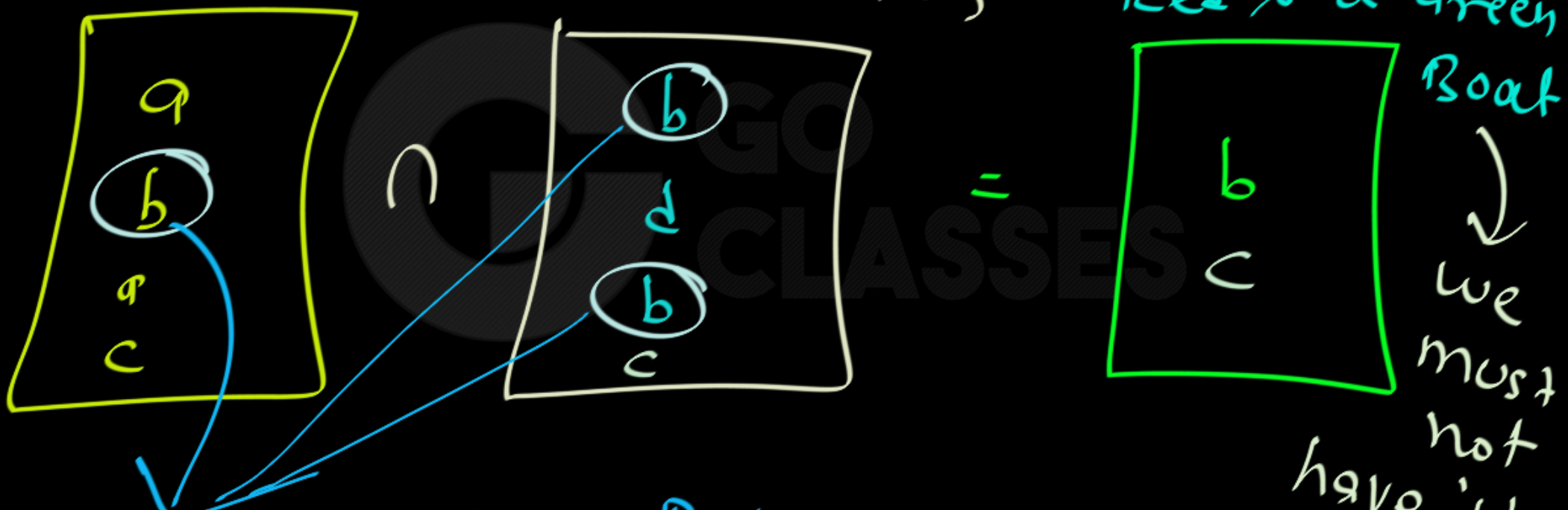
Names of Green Boat Sailors

Names of sailors who have reserved a red and a green boat. ✗

Red Boat Sailors names

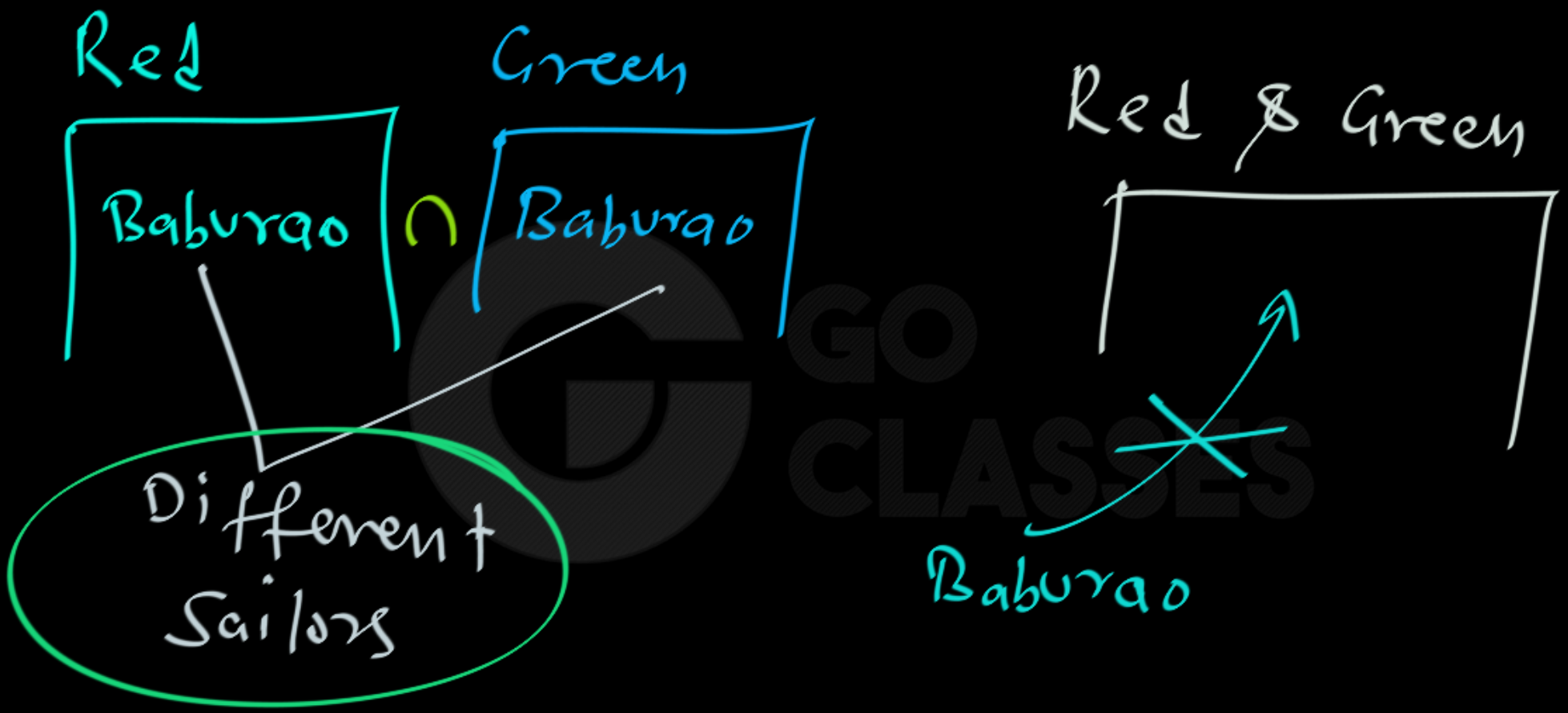
Green Boat Sailors names

Sailors names who book a Red & a Green Boat



we must not have 'b' in o/p.

3 different Sailors, But same name


Red
Green
Baburao
∩
Baburao
Different
Sailors
Red & Green
Baburao

Q: Is the following Query output Correct ??

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
INTERSECT
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Non-key attribute

Names of sailors who have reserved a red and a green boat.

Q: Is the following Query output Correct ??

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

Intersect

```
SELECT  S2.sid
FROM    Sailors S2, Reserves R2, Boats B2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

The sids of all sailors who have reserved red boats and green boats.

Q: Is the following Query output Correct ?? YES

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

Intersect

```
SELECT  S2.sid
FROM    Sailors S2, Reserves R2, Boats B2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

key

The sids of all sailors who have reserved red boats and green boats.

Q: Is the following Query output Correct ??

```
SELECT  S.sid
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
EXCEPT
SELECT  S2.sid
FROM    Sailors S2, Reserves R2, Boats B2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

The sids of all sailors who have reserved red boats but not green boats.

Q: Is the following Query output Correct ?? Yes

```
SELECT  S.sid
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
EXCEPT
SELECT  S2.sid
FROM    Sailors S2, Reserves R2, Boats B2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

key

Red S'ids – Green S'ids

The sids of all sailors who have reserved red boats but not green boats. ✓

Q: Is the following Query output Correct ??

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
EXCEPT
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Names of sailors who have reserved red boats but not green boats.

Q: Is the following Query output Correct ?? No

```
SELECT  S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE   S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
EXCEPT
SELECT  S2.sname
FROM    Sailors S2, Boats B2, Reserves R2
WHERE   S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color = 'green'
```

Monkey

Red.snames − Green.snames

Names of sailors who have reserved red boats but not green boats. No

Sailors

S1	Red	Baburao
S2	Green	Baburao

Ideally;
Baburao (s,) must be in o/p

Except Query will Remove Baburao

Conclusions :

Dangerous Combinations :

1. nonkey Except nonkey
2. nonkey intersect nonkey

---

nonkey union nonkey : not Dangerous

Conclusions :

Not Dangerous Combinations :

(1) key Except : key

(2) key intersect key

(3) key union : key ; not Dangerous

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'
```

(Q20) Find all sids of sailors who have a rating of 10 or have reserved boat 104.

```
SELECT  S.sid
FROM    Sailors S
WHERE   S.rating = 10

UNION
SELECT  R.sid
FROM    Reserves R
WHERE   R.bid = 104
```

Q:

Write SQL query to Show only the distinct tuples of the Input table.

Q:

Write SQL query to Show only the distinct tuples of the Input table.

Query 1:

```
SELECT DISTINCT *
FROM R;
```

Q:

Write SQL query to Show only the distinct tuples of the Input table.

Query2: (SELECT * FROM R) intersect (select * FROM R)

Removes Duplicates

Q:

Write SQL query to Show only the distinct tuples of the Input table.

Query 3: (SELECT * FROM R) Union (select * FROM R)

Removes Duplicates

Next Topic:

SQL

Aggregate Operators

Note:

For Now, Assume that there are NO NULL Values in the tables.

We will study Shortly about what happens when we have NULL Values..

Aggregate Functions : Provide Summary

Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:

- Average: **avg**
- Minimum: **min**
- Maximum: **max**
- Total: **sum**
- Count: **count**

keyword:

Avg ✓

Average ✗

Average = Total / Count

Syntax :

min (A)
max (A)
Count (A)
sum (A)
Avg (A)

→ attribute name

Semantic : we naturally understand.

Syntax :

Sum (Distinct A)
Count (Distinct A)
Avg (Distinct A)
} → ignore Duplicates & Calculate.

Min (Distinct A) ≡ Min (A)

Max (Distinct A) ≡ Max (A)

Special : Count(*)

ALWAYS Count number of Tuples in the Table.

Count (Distinct *) : Invalid

↳ Error

Count (A)

Count (*)

Always Count number of Tuples in the Table.

Don't mix them.

Study Separately

Example 5.9: Consider the relation

Avg (Distinct A) = Sum(Distinct A) / Count (Distinct A)

A	B
1	2
3	4
1	2
1	2

Count(*) = 4

Some examples of aggregations on the attributes of this relation are:

1. SUM(B) = 2 + 4 + 2 + 2 = 10. Sum(Distinct B) = 2+4=6
2. AVG(A) = (1 + 3 + 1 + 1)/4 = 1.5. Avg (Distinct A) = 1+3/2 = 2
3. MIN(A) = 1. min(Distinct A) = 1 = min(A)
4. MAX(B) = 4. max (Distinct B) = 4 = max(B)
5. COUNT(A) = 4. Count (Distinct A) = 2

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
MIN(B) = 2
AVG(B) = 3

```
SELECT SUM(A), COUNT(A), MAX(B), MIN(B), AVG(B)
FROM R;
```

o/p:

sum(A)	count(A)	max(B)	min(B)
7	3	4	2

Remark

- We can also use COUNT(*) which counts the number of tuples in the relation constructed from the FROM and WHERE clauses of the query.

count(*)
3

Avg(B)
3

5.5 AGGREGATE OPERATORS

In addition to simply retrieving data, we often want to perform some computation or summarization. As we noted earlier in this chapter, SQL allows the use of arithmetic expressions. We now consider a powerful class of constructs for computing *aggregate values* such as `MIN` and `SUM`. These features represent a significant extension of relational algebra. SQL supports five aggregate operations, which can be applied on any column, say A, of a relation:

1. `COUNT` ([`DISTINCT`] A): The number of (unique) values in the A column.
2. `SUM` ([`DISTINCT`] A): The sum of all (unique) values in the A column.
3. `AVG` ([`DISTINCT`] A): The average of all (unique) values in the A column.
4. `MAX` (A): The maximum value in the A column.
5. `MIN` (A): The minimum value in the A column.

Note that it does not make sense to specify `DISTINCT` in conjunction with `MIN` or `MAX`

SQL does not allow the use of **distinct** with **count** (*). It is legal to use **distinct** with **max** and **min**, even though the result does not change.

5.2.2 Aggregation Operators

There are several operators that apply to sets or bags of numbers or strings. These operators are used to summarize or "aggregate" the values in one column of a relation, and thus are referred to as *aggregation* operators. The standard operators of this type are:

1. `SUM` produces the sum of a column with numerical values.
2. `AVG` produces the average of a column with numerical values.
3. `MIN` and `MAX`, applied to a column with numerical values, produces the smallest or largest value, respectively. When applied to a column with character-string values, they produce the lexicographically (alphabetically) first or last value, respectively.

(Q25) Find the average age of all sailors.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

(Q25) Find the average age of all sailors.

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

```
SELECT avg(age)
FROM Sailors
```

(Q25) Find the average age of all sailors.

```
SELECT  AVG  (S.age)
FROM    Sailors S
```

(Q26) Find the average age of sailors with a rating of 10.

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

```
SELECT avg (age)
FROM Sailors
WHERE rating = 10
```

(Q26) Find the average age of sailors with a rating of 10.

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

Consider the query:

```
SELECT  sid, max(age)
FROM Sailors ;
```

Invalid Query

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

Note :

If we use at least one Aggregate function (in SELECT clause) then "NO Unaggregates attribute allowes" (Unless there is a Group by clause)

(Q27) Find the name and age of the oldest sailor. Consider the following attempt to answer this query:

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

Invalid Query

The intent is for this query to return not only the maximum age but also the name of the sailors having that age. However, this query is illegal in SQL—if the SELECT clause uses an aggregate operation, then it must use *only* aggregate operations unless the query contains a GROUP BY clause! (The intuition behind this restriction should become clear when we discuss the GROUP BY clause in Section 5.5.1.) Thus, we cannot use MAX (S.age) as well as S.sname in the SELECT clause.

(Q27) Find the name of the oldest sailor.

Consider the query. Is it Correct Query ??

```
SELECT  sname
FROM Sailors
WHERE age = max(age)
```

Sailors(*sid:* `integer`, *sname:* `string`, *rating:* `integer`, *age:* `real`)
Boats(*bid:* `integer`, *bname:* `string`, *color:* `string`)
Reserves(*sid:* `integer`, *bid:* `integer`, *day:* `date`)

(Q27) Find the name of the oldest sailor.

Consider the query. Is it Correct Query ?? No

```
SELECT  sname
FROM Sailors
WHERE age = max(age)
```

Tuple selection condition applies on one Tuple at a time, independently

Sailors(*sid:* integer, *sname:* string, *rating:* integer, *age:* real)
Boats(*bid:* integer, *bname:* string, *color:* string)
Reserves(*sid:* integer, *bid:* integer, *day:* date)

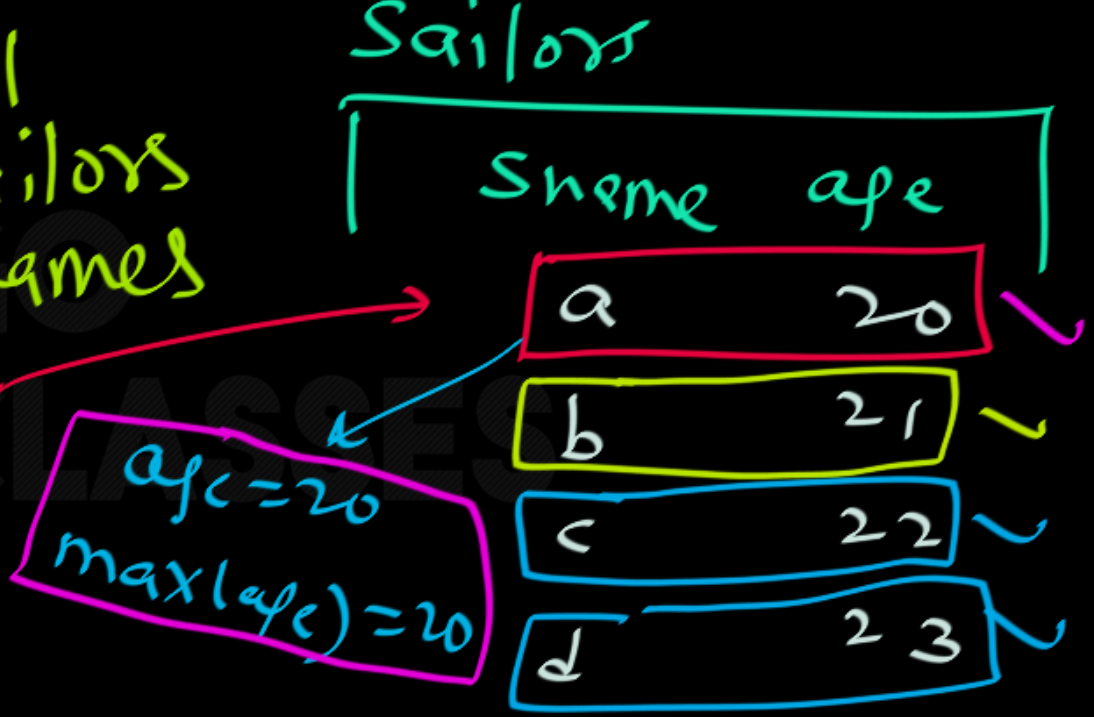
(Q27) Find the name of the oldest sailor.

Consider the query:

```
SELECT  sname
FROM Sailors
WHERE age = max(age)
```

All Sailors Names

Tuple by Tuple, individually, independently



I want name of oldest Sailors.

```
SELECT  Sname
FROM    Sailors
WHERE   age = max(age)
```

Tuple wise max age

Not giving us max age of Entire table.

I want name of oldest Sailors.

Solution

```
SELECT  Sname
FROM    Sailors
WHERE   age = ( Subquery to
                find max
                age of
                Entire Table )
```

I want name of oldest Sailors.

```
SELECT  Sname
FROM    Sailors
WHERE   age = (SELECT max(age)
               FROM Sailors)
```

simple Tuple, simple Column

NOT a value

a Table

max(age)
50

SQL internally Converts this into value

I went name of oldest sailors.

```
SELECT  Sname
FROM    Sailors
WHERE   age = ( SELECT max(age)
                FROM Sailors )
```

(Q27) Find the name and age of the oldest sailor.

```
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
```

Observe that we have used the result of an aggregate operation in the subquery as an argument to a comparison operation. Strictly speaking, we are comparing an age value with the result of the subquery, which is a relation. However, because of the use of the aggregate operation, the subquery is guaranteed to return a single tuple with a single field, and SQL converts such a relation to a field value for the sake of the comparison.

(Q28) Count the number of sailors.

(Q28) Count the number of sailors.

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

Aggregates

- Standard SQL aggregate functions: `COUNT`, `SUM`, `AVG`, `MIN`, `MAX`
- Example: number of users under 18, and their average popularity
  - ```
    SELECT COUNT(*), AVG(pop)
    FROM User
    WHERE age < 18;
    ```
  - `COUNT(*)` counts the number of rows
    ```

SQL

Complete Summary

To be Continued...