UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 3380 Spring 2024

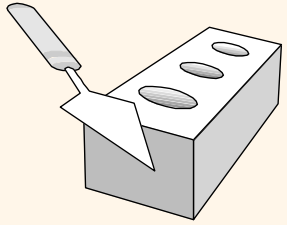
Database Systems

M & W 4:00 to 5:30 PM

Prof. Victoria Hilford

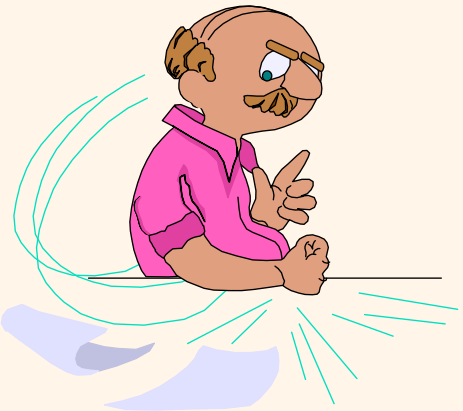
PLEASE TURN your webcam ON (must have)

NO CHATTING during LECTURE



COSC 3380

4 to 5:30

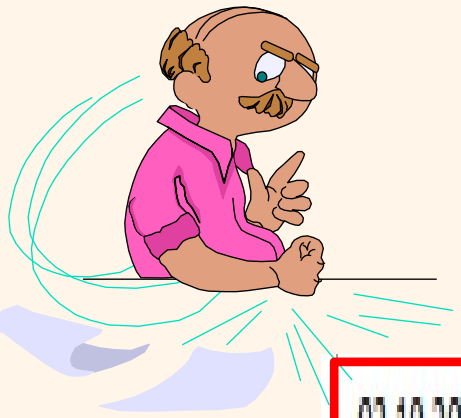


PLEASE
LOG IN
CANVAS

Please close all other windows.

02.19.2024 (10 - Mo)	**ZyBook SET 2 - 2**	**Set 2** LECTURE 7 SQL II
02.21.2024 (11 - We)	**ZyBook SET 2- 3**	**Set 2** LECTURE 10 APPLICATIONS
02.26.2024 (12 - Mo)	**ZyBook SET 2 - 4**	**Set 2** LECTURE 11 WEB APPLICATIONS
02.28.2024 (13 - We)		**EXAM 2 Practice** **(PART of 20 points)**
03.04.2024 (14 - Mo)	**TA Download** **ZyBook SET 2 Sections** **(4 PM)** **(PART of 30 points)**	**EXAM 2 Review** **(PART of 20 points)**
03.06.2024 (15 - We)		**EXAM 2** **(PART of 50 points)**

COSC 3380



Class 10

02.19.2024 (10 - Mo)	ZyBook SET 2 - 2	Set 2 LECTURE 7 SQL II

From 4:00 to 4:07 PM – 5 minutes.

02.19.2024 (10 - Mo)	ZyBook SET 2 - 2	Set 2 LECTURE 7 SQL II

CLASS PARTICIPATION 20 points — 20% of Total

SQL II

Class 10 BEGIN PARTICIPATION
Not available until Feb 19 at 4:00pm | Due Feb 19 at 4:07pm | 100 pts

VH, publish

This is a synchronous online class.

Attendance is required.

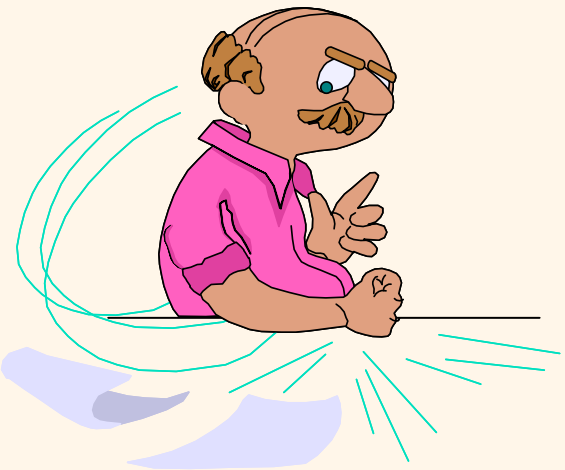
Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

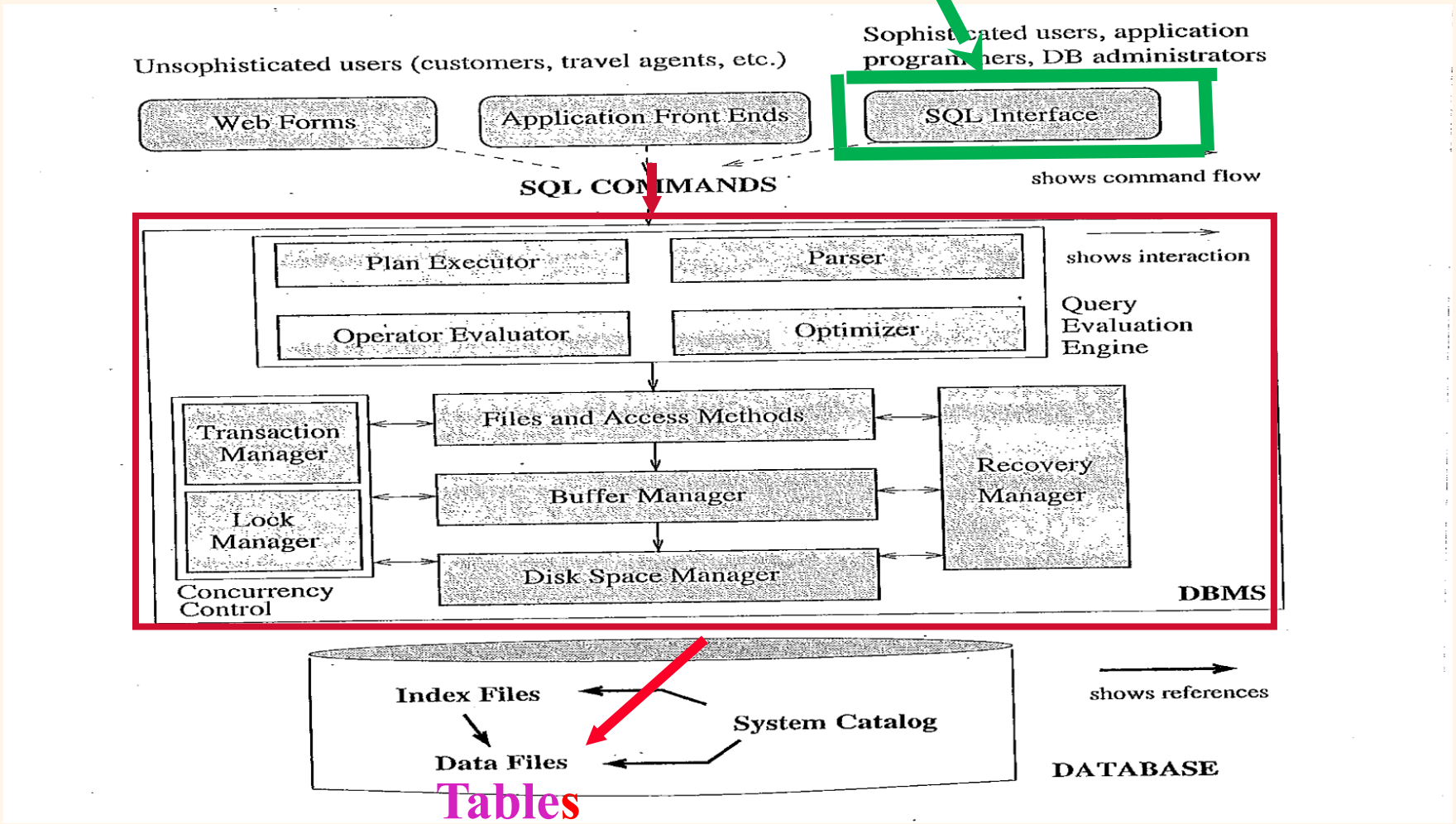
From 4:07 to 5:00 PM – 53 minutes.

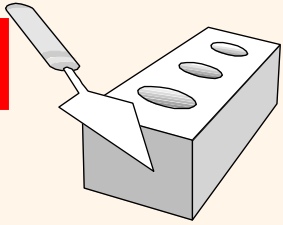
Lecture 7

SQL: Queries, Programming, Triggers



SQL – Structured Query Language





Reserves

sid	bid	day
22	101	10/10/96
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

R1

S1

Boats

B1

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

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

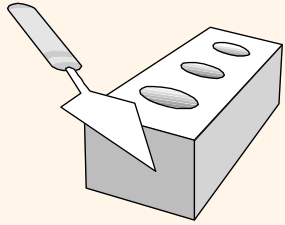
Sailors(sid:integer,sname:string,rating:string,age:real)
Boats(bid:integer,bname:string,color:string)
Reserves(sid:integer,bid:integer,day:date)

Aggregate Operators

Significant extension of **R**elational **A**lgebra (**RA**).

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

single column



```
SQL> SELECT  S.sname
  2  FROM  Sailors S
  3  WHERE  S.rating= (SELECT  MAX(S2.rating)
  4                                FROM  Sailors S2);

SNAME
----------
Rusty
Zorba
```

```
SQL> SELECT  COUNT (*)
  2  FROM  Sailors S;

  COUNT(*)
----------
        10
```

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

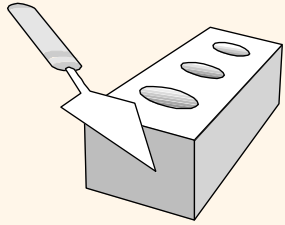
S

1 2

Aggregate Operators

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

single column



```
SQL> SELECT  AVG (S.age)
  2  FROM  Sailors S
  3  WHERE  S.rating=10;

AVG(S.AGE)
----------
      25.5
```

SQL Average age of sailors with rating 10?

```
SQL> SELECT  AVG ( DISTINCT S.age)
  2  FROM  Sailors S
  3  WHERE  S.rating=10;

AVG(DISTINCTS.AGE)
------------------
              25.5
```

?????

SQL Average age of **DISTINCT** sailors with rating 10?

```
SQL> SELECT  COUNT (DISTINCT S.rating)
  2  FROM  Sailors S
  3  WHERE S.sname = 'Bob';

COUNT(DISTINCTS.RATING)
-----------------------
                      1
```

?????

?????

3 4 5

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

S


9. SET 2 - 2:SQL II
0%
9.1 Functions
Hidden
0%
9.2 Subqueries
Hidden
0%
9.3 Complex queries
Hidden
0%

9.1 Functions

Present Note

Aggregate functions

MySQL has functions for processing strings, numbers, dates, and times. An ***aggregate function*** is a function that works on a group of values. Some common aggregate functions include:

- ***COUNT()*** - Counts the number of rows retrieved by a SELECT statement.
- ***MIN()*** - Finds the minimum value in a group.
- ***MAX()*** - Finds the maximum value in a group.
- ***SUM()*** - Sums all the values in a group.
- ***AVG()*** - Finds the arithmetic mean (average) of all the values in a group.

PARTICIPATION ACTIVITY | 9.1.1: Using aggregate functions in a SELECT statement.

Employee

ID	Name	Salary	Bonus
2538	Lisa Ellison	45000	0
5384	Sam Snead	32000	3000
6381	Maria Rodriguez	95000	1000

```
SELECT COUNT(*)
FROM Employee
WHERE Bonus > 500;
```

COUNT(*)
2

```
SELECT MIN(Salary)
FROM Employee;
```

MIN(Salary)
32000

```
SELECT AVG(Salary)
FROM Employee;
```

AVG(Salary)
57333.333333

9.1 Functions

Present Note

Aggregate functions

MySQL has functions for processing strings, numbers, dates, and times. An ***aggregate function*** is a function that works on a group of values. Some common aggregate functions include:

- ***COUNT()*** - Counts the number of rows retrieved by a SELECT statement.
- ***MIN()*** - Finds the minimum value in a group.
- ***MAX()*** - Finds the maximum value in a group.
- ***SUM()*** - Sums all the values in a group.
- ***AVG()*** - Finds the arithmetic mean (average) of all the values in a group.

PARTICIPATION ACTIVITY | 9.1.1: Using aggregate functions in a SELECT statement.

Employee

ID	Name	Salary	Bonus
2538	Lisa Ellison	45000	0
5384	Sam Snead	32000	3000
6381	Maria Rodriguez	95000	1000

```
SELECT AVG(Salary)
FROM Employee;
```

Salary
45000
32000
95000

TA time (Alvaro) – 4 minutes
(CA 9.2.1 Step 1 – Aggregate Functions)

CHALLENGE ACTIVITY | 9.2.1: Aggregate functions.

Country

ID	Name	Capital	PopulationMillions
40	Austria	Vienna	8.84
417	Kyrgyzstan	Bishkek	6.32
516	Namibia	Windhoek	2.45
862	Venezuela	Caracas	28.87

Complete the SELECT statement to count the number of countries with an PopulationMillions greater than 12.

```
SELECT COUNT(*)
FROM Country
WHERE PopulationMillions > 12
(A) /* Type your code here */
(B)
```

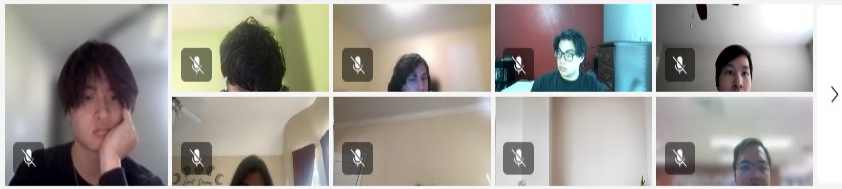
COUNT(*)
1

1

COSC 3380 23578 M & W 4 to 5:30 PM LECTURE (in TEAMS from 3:50 to 5:15 PM)

28:52

Take control | Pop out | Chat | 110 People | Raise | View | Rooms | Apps | More | Camera | Mic | Share | Leave



AU Urtaza, Alvaro A

VE Elkins, Vict...

View all

zyBooks My library > COSC 3380: Database Systems home > 9.2: Aggregate functions

zyBooks catalog | Help/FAQ | Alvaro Urtaz

266	Gabon	Libreville	1960
24	Angola	Luanda	1975

5

Complete the SELECT statement that finds the average IndependenceYear.

```
SELECT ___(A)___(___(B)___)
FROM Country;
```

(A) AVG

(B) IndependenceYear

1 2 3 4 5

Check Next

Expected:
(A) AVG
(B) IndependenceYear

AVG(IndependenceYear): Finds the arithmetic mean (average) of all the values of IndependenceYear in the table.
The statement returns the average of all the values of IndependenceYear in the table, which is 1963.

View solution (Instructors only)

Feedback?

How was this section? Provide section feedback

Urtaza, Alvaro A

4:19 PM 2/19/2024

Participants

Invite someone or dial a number

Share invite

In this meeting (110) Mute all

- Hilford, Victoria — Organizer
- RA Adhikari, Rohit
- MA Ahmed, Mohamed A
- AA Akram, Ali
- SA Altaf, Sameer
- SA Alvarez, Stephanie
- OA Anayor-Achu, Ogochukwu E
- HA Avci, Hatice Kubra
- RA Aysola, Riya
- AB Bahl, Anish
- SB Banza, Sean Paolo B
- HB Bui, Hieu
- Burger, Jake
- VC Carrillo-Zepeda, Victor E

GROUP BY and HAVING

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

So far, we've applied aggregate operators to all **(qualifying) tuples**. Sometimes, we want to apply them to **each of several *groups* of tuples**.

Consider: (Q31)

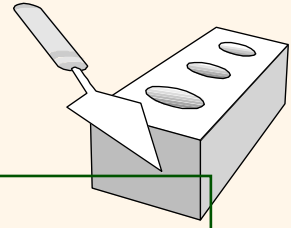
*Find the **age** of the youngest sailor **for each rating level**.*

- In general, we don't know **how many rating levels exist**, and **what the rating values** for these levels are!

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

for *i* = 1; 2, ... ; 10:

Queries With *GROUP BY* and *HAVING*



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

```
SELECT      [DISTINCT]
target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-
qualification
```

- The *target-list* contains
  - (i) **attribute names**
  - (ii) terms with **aggregate operations** (e.g., **MIN** (*S.age*)).

- The *grouping-list* must be a subset of *target-list*. Intuitively, each answer tuple corresponds to a *group*, and these **attributes** must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in

Conceptual Evaluation

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

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

- The cross-product of ***Relation****-list* is computed, tuples that fail *qualification* are discarded, `*unnecessary*' fields are deleted, and the remaining tuples are partitioned into **groups** by the value of **attributes** in *grouping-list*.

- The *group-qualification* is then applied to eliminate some **groups**. Expressions in *group-qualification* must have a *single value per group*!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*.

GROUP BY and HAVING clauses

Aggregate functions are commonly used with the GROUP BY clause. The ***GROUP BY*** clause groups rows with identical values into a set of summary rows. Some important points about the GROUP BY clause:

- One or more columns are listed after GROUP BY, separated by commas.
- GROUP BY clause returns one row for each group.
- Each group may be ordered with the ORDER BY clause.
- GROUP BY clause must appear before the ORDER BY clause and after the WHERE clause (if present).

PARTICIPATION ACTIVITY 9.1.3: GROUP BY clause.

City

ID	Name	CountryCode	District	Population
3162	Lusaka	ZMB	1	1317000
3163	Ndola	ZMB	2	329200
3164	Kitwe	ZMB	2	288600
3165	Kabwe	ZMB	3	154300
3166	Chingola	ZMB	2	142400
4068	Harare	ZWE	1	1410000
4069	Bulawayo	ZWE	2	621742
4070	Chitungwiza	ZWE	1	274912

```
SELECT CountryCode, SUM(Population)
FROM City
GROUP BY CountryCode;
```

CountryCode	SUM(Population)
ZMB	2231500
ZWE	2306654

(Q32) Find the rating and age of the youngest sailor with age ≥ 18, for each rating with at least 2 such sailors

```
SQL> SELECT  S.rating,  MIN (S.age)
  2  FROM  Sailors S
  3  WHERE  S.age >= 18
  4  GROUP BY  S.rating
  5  HAVING  COUNT (*) > 1;

    RATING MIN(S.AGE)
---------- ----------
         7         35
```

Sailors

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
~~71~~	~~zorba~~	~~10~~	~~16.0~~
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

- Only S.rating and S.age are mentioned in the SELECT, GROUP BY or HAVING clauses; other **attributes** `*unnecessary*'.
- 2nd column of result is unnamed. (Use **AS** to name it.)

rating	age
~~1~~	~~33.0~~
7	45.0
7	35.0
~~8~~	~~55.5~~
~~10~~	~~35.0~~

rating	age
7	35.0

Answer ***Relation*** *?*

TA, Jordan (A – L).

TA, Alvaro (M – Z).

Please compare CANVAS vs. TEAMS Attendance.
Print screens of students in CANVAS but not in the TEAMS meeting.
(2.19.2024 Attendance X missing LastName.docx)

General Constrain

- Useful when more general **IC**s than keys are involved.
- Can use queries to express constraint.
- Constraints can be **named**.

(**Interlake boats cannot be reserved** constraint)

Reserves

sid	bid	day
22	101	10/10/96
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

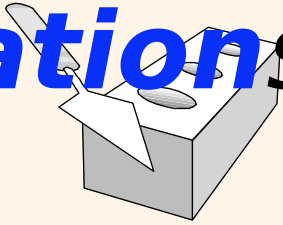
bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

```
CREATE TABLE Sailors
        ( sid INTEGER,
          sname CHAR(10),
          rating INTEGER,
          age REAL,
          PRIMARY KEY (sid),
          CHECK ( rating >= 1
                  AND rating <=
10))
```

```
CREATE TABLE Reserves
        ( sid INTEGER,
          bid INTEGER,
          day DATE,
          PRIMARY KEY (sid,bid,day),
          CONSTRAINT noInterlakeRes
          CHECK (`Interlake' <>
                 ( SELECT B.bname
                   FROM Boats B
                   WHERE B.bid=bid)))
```

WOW!

Constraints Over Multiple Relations



bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

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

```
CREATE TABLE Sailors
 ( sid INTEGER,
sname CHAR(10),
rating INTEGER,
age REAL,
PRIMARY KEY (sid),
CHECK
(  (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 10
```

Number of boats plus number of sailors is < 100

Awkward and wrong!

If Sailors is empty, the number of Boats tuples can be anything!

ASSERTION is th
right solution;
not associated
with either

```
CREATE ASSERTION smallClub
CHECK
(  (SELECT COUNT (S.sid) FROM Sailors S)
+ (SELECT COUNT (B.bid) FROM Boats B) < 100
```

Not supported in Oracle!

CHECK constraint

The ***CHECK*** constraint specifies an expression that limits the range of a column's values. Ex: `CHECK (Salary > 20000)` ensures the Salary is greater than 20,000. If the CHECK expression does not evaluate to TRUE or UNKNOWN (for NULL values), the constraint is violated.

A CHECK constraint can be a column-level or table-level constraint.

PARTICIPATION ACTIVITY 8.9.7: CHECK constraints on the Employee table.

```
CREATE TABLE Employee (
   ID        SMALLINT UNSIGNED,
   Name      VARCHAR(60),
   BirthDate DATE,
   HireDate  DATE CHECK (HireDate >= '2000-01-01' AND HireDate <= '2019-12-31'),
   CHECK (BirthDate < HireDate),
   PRIMARY KEY (ID)
);
```

Employee

ID	Name	BirthDate	HireDate
6381	Maria Rodriguez	1970-12-01	2000-01-31
~~2538~~	~~Lisa Ellison~~	~~2000-01-01~~	~~2020-03-15~~
~~5384~~	~~Sam Snead~~	~~2003-11-29~~	~~2003-11-01~~
8312	Jiho Chen	NULL	2013-06-03

Subqueries

A ***subquery***, sometimes called a ***nested query*** or ***inner query***, is a query within another SQL query. The subquery is typically used in a SELECT statement's WHERE clause to return data to the outer query and restrict the selected results. The subquery is placed inside parentheses ( ).

PARTICIPATION ACTIVITY 9.2.1: Subquery examples.

CountryLanguage

CountryCode	Language	IsOfficial	Percentage
ABW	Dutch	T	5.3
AFG	Balochi	F	0.9
AGO	Kongo	F	13.2
ALB	Albaniana	T	97.9
AND	Catalan	T	32.3

TA time (Alvaro) – 4 minutes
(CA 9.3.1 Step 1 – Subqueries)

CHALLENGE ACTIVITY | 9.2.1: Subqueries.

Course

CourseId	CourseCode	CourseName	Capacity	InstructorId
3891	HIST692	World History	25	1
9844	HIST89	European History	200	1
2156	BUS83	Intro to Business	50	2
6556	ENGL22	English Literature	75	3
2145	ENGL821	Modern Literature	150	3

Instructor

InstructorId	InstructorName	Rank	Department
1	Gus Ruiz	Associate Professor	History
2	Ari Chen	Associate Professor	Business
3	Taj Vega	Assistant Professor	English

Note: Both tables may not be necessary to complete this level.

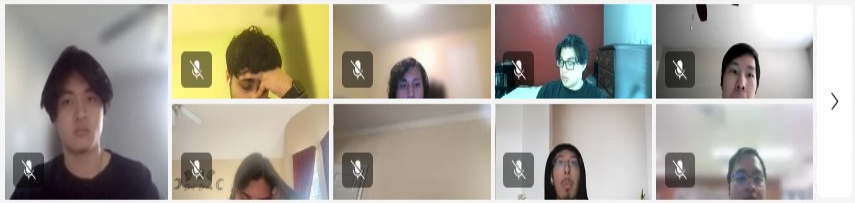
Select the values returned by the query below.

```
SELECT CourseName
FROM Course
WHERE InstructorId =
    (SELECT InstructorId
    FROM Instructor
    WHERE InstructorName = 'Gus Ruiz');
```

1

- [ ] Modern Literature
- [ ] HIST89
- [x] European History
- [x] World History
- [ ] 1
- [ ] English Literature

1



8979	HIST388	European History	25	2

Instructor

• InstructorId	InstructorName	Rank	Department
1	Abe Chen	Associate Professor	Computer Science
2	Zoe Cook	Assistant Professor	History
3	Guy Choi	Assistant Professor	Math

Note: Both tables may not be necessary to complete this level.

Select the values returned by the query below.

```
SELECT CourseCode
FROM Course
WHERE InstructorId IN
   (SELECT InstructorId
   FROM Instructor
   WHERE Department != 'History');
```

- ☐ CS879
- ☐ HIST388
- ☐ HIST860
- ☐ Math
- ☐ MATH300
- ☐ MATH588

1 2 3 4 5

Check Next

✔ Expected: CS879, MATH300, MATH588

The subquery selects the instructor ids that are not in the History department, which are 1 and 3. Then, the outer query selects all course codes with InstructorId = 1 or InstructorId = 3, so MATH300, MATH588, and CS879.

View solution *(Instructors only)*

Feedback?

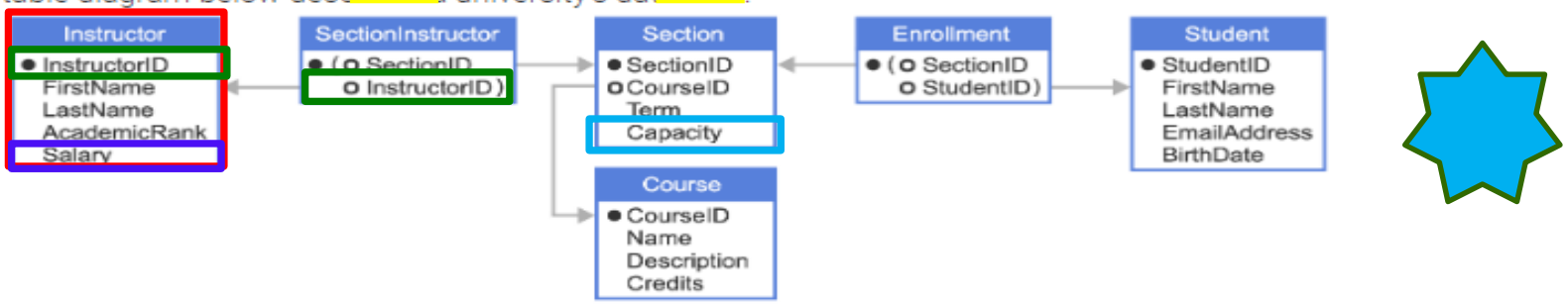
Exploring further:

TA time (Alvaro) – 4 minutes
(CA 9.4.1 Step 4 – Complex Queries)

CHALLENGE ACTIVITY | 9.3.1: Complex queries.

I SI S

The table diagram below desc a university's dat .



The university wants to know the minimum salary of instructors who have taught sections in a Summer term with capacities greater than the average capacity of all sections.

The query below answers the university's question. Fill in the blanks.

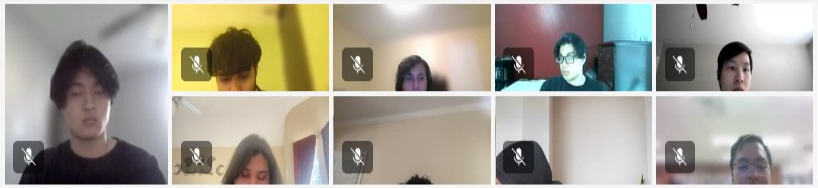
```
SELECT I.FirstName, I.LastName,
FROM Instructor I
INNER JOIN SectionInstructor SI ON I.InstructorID = SI.InstructorID
INNER JOIN Section S ON SI.SectionID = S.SectionID
WHERE S.Term LIKE 'Summer%' AND S.Capacity >
    (SELECT AVG(Capacity)
    FROM Section)
GROUP BY I.FirstName, I.LastName
```

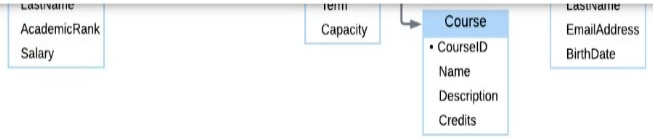
MIN(I.Salary)

(A) Ex: Keyword
(B)
(C)

4

Welcome to 4GL Languages





The query below answers the university's question. Fill in the blanks.

```
SELECT ST.FirstName, ST.LastName, ___(A)___
FROM Student ST
INNER JOIN Enrollment E ON ST.StudentID = E.StudentID
INNER JOIN Section SE ON E.SectionID = SE.SectionID
INNER JOIN Course C ON SE.CourseID = C.CourseID
WHERE C.Credits >
    (SELECT MIN(Credits)
    FROM Course)
GROUP BY ___(B)___, ST.LastName, SE.Term
HAVING COUNT(*) ___(C)___;
```

(A) COUNT(*)
(B) ST.FirstName
(C) > 5

1 | 2 | 3 | 4

Check | Next | **Done**. Click any level to practice more. Completion is preserved.

✔ Expected:
(A) COUNT(*)
(B) ST.FirstName
(C) > 5

(A): COUNT(*) counts how many rows exist in each group.

(B): GROUP BY ST.FirstName, ST.LastName, SE.Term forms groups based on the ST.FirstName, ST.LastName, and SE.Term columns.

(C): HAVING COUNT(*) > 5 selects only groups that have a row count > 5.

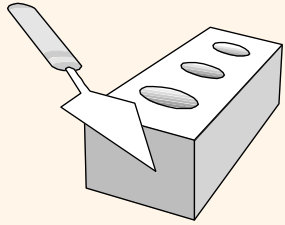
View solution (Instructors only)

TA, Jordan (A – L).

TA, Alvaro (M – Z).

Please compare CANVAS vs. TEAMS Attendance.
Print screens of students in CANVAS but not in the TEAMS meeting.
(2.19.2024 Attendance X missing LastName.docx)

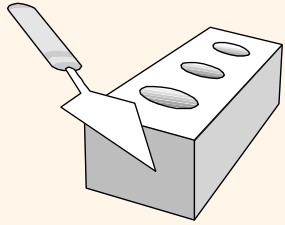
Triggers



- **Trigger**: procedure that starts automatically if specified changes occur to the DataBase

- Three parts:
  - Event (activates the **trigger**)
  - Condition (tests whether the **triggers** should run)
  - Action (what happens if the **trigger** runs)

Triggers



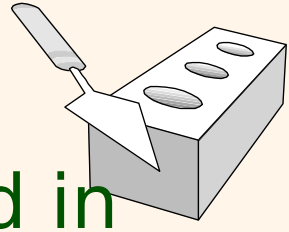
A *condition* in a **trigger** can be:

- a *true/false* statement
- a query:
  - A query is interpreted as *true* if the answer set in nonempty
  - A query is interpreted as *false* if the query has no answers

Three parts:

- Event (activates the **trigger**)
- Condition (tests whether the **triggers** should run)
- Action (what happens if the **trigger** runs)

*Uses of **Triggers***



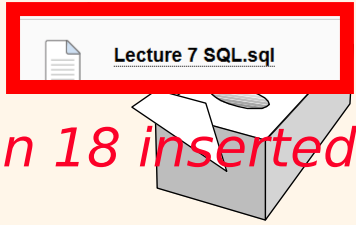
Alert users to unusual events (as reflected in updates to the **Data**base):

- Check whether a customer placing an order has made enough purchases in the past month to qualify for an additional discount;

Can generate a log of events to support auditing and security checks:

- Each time a customer places an order, we can create a record with the customer's ID and current credit limit and **insert this record in a customer history Table**;

Triggers: Example (SQL:1999)

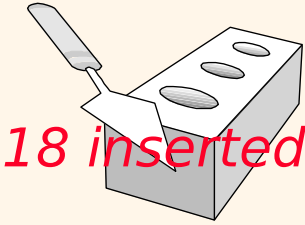


youngSailorsUpdate *keeps track of sailors younger than 18 inserted in* ***SAILORS***

Table ***YoungSailors*** ***contains only the SAILORS with age <= 18***

```
CREATE  or REPLACE TRIGGER
   youngSailorsUpdate
AFTER INSERT ON SAILORS
FOR EACH ROW
BEGIN
   INSERT INTO YoungSailors(sid, name, age,
   rating)
         SELECT S.sid, S.sname, S.age, S.rating
         FROM SAILORS S
         WHERE S.age <= 18;
END;
/
```

Triggers: Example (SQL:1999)



youngSailorsUpdate *keeps track of sailors younger than 18 inserted in* ***SAILORS***

Table ***YoungSailors*** ***contains only the SAILORS with age <= 18***

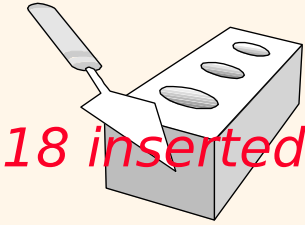
```
DROP TABLE YoungSailors;

CREATE TABLE YoungSailors ( sid     INTEGER,
                            name    VARCHAR(10),
                            age     REAL,
                            rating  INTEGER,
                            PRIMARY KEY (sid),
                            CHECK (rating >= 1 AND rating <= 10 ));
```

```
SQL> select * from sailors;

       SID SNAME          RATING        AGE
---------- ---------- ---------- ----------
        22 Dustin              7         45
        29 Brutus              1         33
        31 Lubber              8       55.5
        32 Andy                8       25.5
        58 Rusty              10         35
        64 Horatio             7         35
        71 Zorba              10         16
        74 Horatio             9         35
        85 Art                 3       25.5
        95 Bob                 3       63.5

10 rows selected.
```

```
SQL> DROP TABLE YoungSailors;

Table dropped.

SQL>
SQL> CREATE TABLE YoungSailors ( sid      INTEGER,
  2                                                  name   VARCHAR(10),
  3             age      REAL,
  4             rating   INTEGER,
  5             PRIMARY KEY (sid),
  6             CHECK (rating >= 1 AND rating <= 10 ));

Table created.
```

```
SQL> SELECT * FROM YoungSailors;

no rows selected
```

Triggers: Example (SQL:1999)



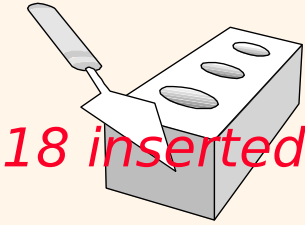
youngSailorsUpdate *keeps track of sailors younger than 18 inserted in* ***SAILORS***

Table ***YoungSailors*** ***contains only the SAILORS with age <= 18***

```
CREATE or REPLACE TRIGGER youngSailorsUpdate
  AFTER INSERT ON SAILORS
    FOR EACH ROW
    BEGIN
       INSERT INTO YoungSailors(sid, name, age, rating)
              SELECT S.sid, S.sname, S.age, S.rating
              FROM SAILORS S
              WHERE S.age <= 18;
   END;
 /
```

```
SQL> CREATE or REPLACE TRIGGER youngSailorsUpdate
  2     AFTER INSERT ON SAILORS
  3        FOR EACH ROW
  4        BEGIN
  5   INSERT INTO YoungSailors(sid, name, age, rating)
  6    SELECT S.sid, S.sname, S.age, S.rating
  7    FROM SAILORS S
  8    WHERE S.age <= 18;
  9       END;
 10  /

Trigger created.
```

Triggers: Example (SQL:1999)



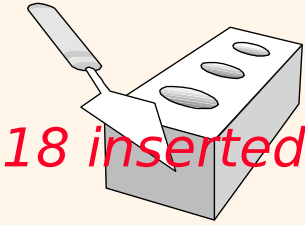
youngSailorsUpdate *keeps track of sailors younger than 18 inserted in* ***SAILORS***

Table ***YoungSailors*** ***contains only the SAILORS with age <= 18***

```
INSERT INTO Sailors VALUES (99, 'Vic',    3,  18.0);
```

```
SQL>
SQL>
SQL> INSERT INTO Sailors VALUES (99, 'Vic',     3,  18.0);
INSERT INTO Sailors VALUES (99, 'Vic',     3,  18.0)
            *
ERROR at line 1:
ORA-04091: table VHILFORD.SAILORS is mutating, trigger/function may not see it
ORA-06512: at "VHILFORD.YOUNGSAILORSUPDATE", line 2
ORA-04088: error during execution of trigger 'VHILFORD.YOUNGSAILORSUPDATE'
```

Triggers: Example (SQL:1999)



youngSailorsUpdate *keeps track of sailors younger than 18 inserted in* ***SAILORS***

Table ***YoungSailors*** ***contains only the SAILORS with age <= 18***

```
SQL> CREATE or REPLACE TRIGGER youngSailorsUpdate
  2        AFTER INSERT ON SAILORS
  3         REFERENCING NEW AS new
  4         FOR EACH ROW
  5         BEGIN
  6    INSERT INTO YoungSailors(sid, name, age, rating)
  7      VALUES  (:new.sid,  :new.sname,  :new.age,  :new.rating);
  8        END;
  9  /

Trigger created.
```

Triggers

A ***trigger*** is like a stored procedure or a stored function, with two differences:

- Triggers have neither parameters nor a return value. Triggers read and write tables but do not communicate directly with a calling program.
- Triggers are not explicitly invoked by a CALL statement or within an expression. Instead triggers are associated with a specific table and execute whenever the table is changed.

Triggers are used to enforce business rules automatically as data is updated, inserted, and deleted.

The ***CREATE TRIGGER*** statement specifies a TriggerName followed by four required keywords:

- `ON TableName` identifies the table associated with the trigger.
- `INSERT`, `UPDATE`, or `DELETE` indicates that the trigger executes when the corresponding SQL operation is applied to the table.
- `BEFORE` or `AFTER` determines whether the trigger executes before or after the insert, update, or delete operation.
- `FOR EACH ROW` indicates the trigger executes repeatedly, once for each row affected by the insert, update, or delete operations.

MySQL triggers do not support all features of the SQL standard. Ex: The standard includes keywords `FOR EACH STATEMENT` as an alternative to `FOR EACH ROW`. `FOR EACH STATEMENT` indicates the trigger executes once only, rather than repeatedly for each affected row.

Like stored procedures, the trigger body may be a simple or compound statement. Within the body, keywords `OLD` and `NEW` represent the name of the table associated with the trigger. `OLD` represents the table values prior to an update or delete operation. `NEW` represents the table values after an insert or update operation.

Figure 10.2.8: CREATE TRIGGER statement.

```
CREATE TRIGGER TriggerName
[ BEFORE | AFTER ] [ INSERT | UPDATE | DELETE ]
ON TableName
FOR EACH ROW
Body;
```

Feedback?

PARTICIPATION ACTIVITY | 10.2.10: Trigger example.

```
CREATE TRIGGER ExamGrade
BEFORE INSERT
ON Exam
FOR EACH ROW
BEGIN

   IF NEW.Score >= 90 THEN
      SET NEW.Grade = 'A';
   ELSEIF NEW.Score >= 80 THEN
      SET NEW.Grade = 'B';
   ELSEIF NEW.Score >= 70 THEN
      SET NEW.Grade = 'C';
   ELSE
      SET NEW.Grade = 'F';
   END IF;

END;
```

```
mysql> INSERT INTO Exam (ID, Score)
    -> VALUES (1, 79), (2, 92), (3, 85);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Exam;
+----+-------+-------+
| ID | Score | Grade |
+----+-------+-------+
|  1 |    79 | C     |
|  2 |    92 | A     |
|  3 |    85 | B     |
+----+-------+-------+
```

SQL II Practice Questions

9. SET 2 - 2:SQL II

) Refer to the Product table. Complete the SQL statement so the result table shows 63, which is the total quantity of all products.

ProductID	ProductName	UnitPrice	Quantity
1	Onesies set	10.50	20
2	Sunsuit	18.00	10
3	Romper	23.99	5
4	Pajama set	10.99	20
5	Shorts set	12.89	8

```
SELECT _____
FROM Product;
```

a. SUM(Quantity)

b. AVG(Quantity)

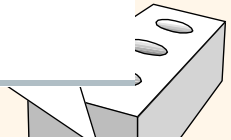
c. MIN(Quantity)

d. Quantity * 5

?????

hrke

9. SET 2 - 2:SQL II



Refer to the Product table. Complete the SQL statement so the result table shows 23.99.

ProductID	ProductName	UnitPrice	Quantity
1	Onesies set	10.50	20
2	Sunsuit	18.00	10
3	Romper	23.99	5
4	Pajama set	10.99	20
5	Shorts set	12.89	8

```
SELECT _____
FROM Product;
```

a. SUM(UnitPrice)

b. MAX(Quantity)

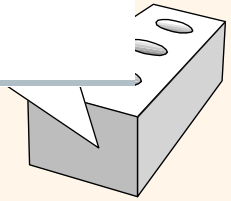
c. AVG(UnitPrice)

d. MAX(UnitPrice)

?????

hrke

9. SET 2 - 2:SQL II



Refer to the Customer table. Which query returns the result table below?

Customer

CustomerId	Name	RewardsMember
1	Klara Lane	No
2	Deja Roderick	No
3	Janine Harstad	Yes
4	Alonso Zastrow	Yes
5	Karisa Guidroz	Yes
6	Marcos Larimer	No
7	Danyell Cartagena	Yes
8	Celine Song	No
9	Grover Steiger	Yes
10	Al Burrell	Yes
11	Latasha McNutt	No
12	Lauri Hanes	Yes

Result

RewardsMember	COUNT(*)
No	5
Yes	7

a.
```
SELECT RewardsMember, COUNT(*)
ORDER BY RewardsMember
FROM Customer;
```

b.
```
SELECT RewardsMember, COUNT(*)
FROM Customer
ORDER BY RewardsMember;
```

c.
```
SELECT RewardsMember, COUNT(*)
GROUP BY RewardsMember
FROM Customer;
```

d.
```
SELECT RewardsMember, COUNT(*)
FROM Customer
GROUP BY RewardsMember;
```

?????

9. SET 2 - 2:SQL II

Refer to the Customer table. Which query returns the result table below?

Customer

CustomerId	Name	StreetAddress	State
2	Deja Roderick	857 North Bayberry Ave.	IL
6	Marcos Larimer	30 Sutor St.	NY
7	Danycll Cartagena	861 Fairview St.	NY
8	Celine Song	47 Hall Ave.	NY
9	Grover Steiger	9400 S Normandie Ave #14	CA
10	Al Burrell	8424 Bowman St.	NY
11	Latasha McNutt	5723 Morgan Ave.	CA
12	Lauri Hanes	2045 W Jackson Blvd	IL

Result

State	COUNT(*)
CA	2
IL	2
NY	4

a.
```
SELECT State, COUNT(*)
FROM Customer
GROUP BY State
WHERE SUM(CustomerId) > 1;
```

b.
```
SELECT State, COUNT(*)
FROM Customer
GROUP BY State
WHERE COUNT(*) > 1;
```

c.
```
SELECT State, COUNT(*)
FROM Customer
GROUP BY State
HAVING COUNT(*) > 1;
```

d.
```
SELECT State, COUNT(*)
FROM Customer
GROUP BY State
HAVING MAX(CustomerId) > 1;
```

Conceptual Evaluation

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

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

- The cross-product of ***Relation**-list* is computed, ⋈ tuples that fail *qualification* are discarded, σ `*unnecessary*' fields are deleted, π and the remaining tuples are partitioned into **groups** by the value of **attributes** in *grouping-list*.

?????

9. SET 2 - 2:SQL II

Refer to the tables. The Product's Quantity column stores the stockroom's product quantity before any products are sold. Which products are selected by the query below?

Product

ProductID	ProductName	Size	Quantity
1	Onesies set	3-6M	20
2	Sunsuit	3-6M	10
3	Romper	9-12M	5
4	Pajama set	24M	20
5	Shorts set	18M	8

Sales

OrderID	CustomerID	ProductID	OrderDate	Quantity
1	5	2	2020-03-15	3
2	3	1	2020-03-22	1
3	4	5	2020-05-30	2
4	5	3	2020-03-16	8
5	5	4	2020-03-16	5
6	12	4	2020-06-16	1
7	12	1	2020-06-16	1
8	7	1	2020-06-17	4
9	7	5	2020-06-17	4
10	2	5	2020-06-20	2

```
SELECT ProductName
FROM Product P
WHERE Quantity >
    (SELECT SUM(Quantity)
    FROM Sales
    WHERE ProductId = P.ProductId);
```

a. All products that are sold-out.

b. All products that are in stock.

c. All of the products in the database.

d. No products are selected.

Conceptual Evaluation

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

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

- The cross-product of *Relation-list* is computed, ⋈ tuples that fail *qualification* are discarded, σ `unnecessary' fields are deleted, π and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.

?????

How?

hrke

9. SET 2 - 2:SQL II

Refer to the tables. The Product's Quantity column stores the stockroom's product quantity before any products are sold. Which products are selected by the query below?

Product

ProductID	ProductName	Size	Quantity
1	Onesies set	3-6M	20
2	Sunsuit	3-6M	10
3	Romper	9-12M	5
4	Pajama set	24M	20
5	Shorts set	18M	8

Sales

OrderID	CustomerID	ProductID	OrderDate	Quantity
1	5	2	2020-03-15	3
2	3	1	2020-03-22	1
3	4	5	2020-05-30	2
4	5	3	2020-03-16	8
5	5	4	2020-03-16	5
6	12	4	2020-06-16	1
7	12	1	2020-06-16	1
8	7	1	2020-06-17	4
9	7	5	2020-06-17	4
10	2	5	2020-06-20	2

```
SELECT ProductName
FROM Product P
WHERE Quantity >
    (SELECT SUM(Quantity)
    FROM Sales
    WHERE ProductId = P.ProductId);
```

SUM()

6

ProductName

Onesies set

9. SET 2 - 2:SQL II

Refer to the tables. The Product's Quantity column stores the stockroom's product quantity before any products are sold. Which products are selected by the query below?

Product

ProductID	ProductName	Size	Quantity
1	Onesies set	3-6M	20
2	Sunsuit	3-6M	10
3	Romper	9-12M	5
4	Pajama set	24M	20
5	Shorts set	18M	8

Sales

OrderID	CustomerID	ProductID	OrderDate	Quantity
1	5	2	2020-03-15	3
2	3	1	2020-03-22	1
3	4	5	2020-05-30	2
4	5	3	2020-03-16	8
5	5	4	2020-03-16	5
6	12	4	2020-06-16	1
7	12	1	2020-06-16	1
8	7	1	2020-06-17	4
9	7	5	2020-06-17	4
10	2	5	2020-06-20	2

```
SELECT ProductName
FROM Product P
WHERE Quantity >
    (SELECT SUM(Quantity)
    FROM Sales
    WHERE ProductId = P.ProductId);
```

ProductName

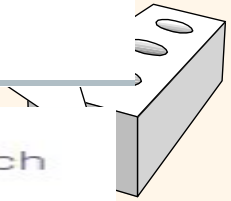
Onesies set

Sunsuit

SUM()

3

9. SET 2 - 2:SQL II



Refer to the tables. The Product's Quantity column stores the stockroom's product quantity before any products are sold. Which products are selected by the query below?

Product

ProductID	ProductName	Size	Quantity
1	Onesies set	3-6M	20
2	Sunsuit	3-6M	10
3	Romper	9-12M	5
4	Pajama set	24M	20
5	Shorts set	18M	8

Sales

OrderID	CustomerID	ProductID	OrderDate	Quantity
1	5	2	2020-03-15	3
2	3	1	2020-03-22	1
3	4	5	2020-05-30	2
4	5	3	2020-03-16	8
5	5	4	2020-03-16	5
6	12	4	2020-06-16	1
7	12	1	2020-06-16	1
8	7	1	2020-06-17	4
9	7	5	2020-06-17	4
10	2	5	2020-06-20	2

```
SELECT ProductName
FROM Product P
WHERE Quantity >
    (SELECT SUM(Quantity)
    FROM Sales
    WHERE ProductId = P.ProductId);
```

ProductName

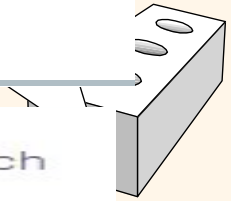
Onesies set

Sunsuit

SUM()

8

9. SET 2 - 2:SQL II



Refer to the tables. The Product's Quantity column stores the stockroom's product quantity before any products are sold. Which products are selected by the query below?

Product

ProductID	ProductName	Size	Quantity
1	Onesies set	3-6M	20
2	Sunsuit	3-6M	10
3	Romper	9-12M	5
4	Pajama set	24M	20
5	Shorts set	18M	8

Sales

OrderID	CustomerID	ProductID	OrderDate	Quantity
1	5	2	2020-03-15	3
2	3	1	2020-03-22	1
3	4	5	2020-05-30	2
4	5	3	2020-03-16	8
5	5	4	2020-03-16	5
6	12	4	2020-06-16	1
7	12	1	2020-06-16	1
8	7	1	2020-06-17	4
9	7	5	2020-06-17	4
10	2	5	2020-06-20	2

```
SELECT ProductName
FROM Product P
WHERE Quantity >
    (SELECT SUM(Quantity)
    FROM Sales
    WHERE ProductId = P.ProductId);
```

SUM()

6

ProductName

Onesies set

Sunsuit

Pajama set

9. SET 2 - 2:SQL II

Refer to the tables. The Product's Quantity column stores the stockroom's product quantity before any products are sold. Which products are selected by the query below?

Product

ProductID	ProductName	Size	Quantity
1	Onesies set	3-6M	20
2	Sunsuit	3-6M	10
3	Romper	9-12M	5
4	Pajama set	24M	20
5	Shorts set	18M	8

Sales

OrderID	CustomerID	ProductID	OrderDate	Quantity
1	5	2	2020-03-15	3
2	3	1	2020-03-22	1
3	4	5	2020-05-30	2
4	5	3	2020-03-16	8
5	5	4	2020-03-16	5
6	12	4	2020-06-16	1
7	12	1	2020-06-16	1
8	7	1	2020-06-17	4
9	7	5	2020-06-17	4
10	2	5	2020-06-20	2

```
SELECT ProductName
FROM Product P
WHERE Quantity >
    (SELECT SUM(Quantity)
    FROM Sales
    WHERE ProductId = P.ProductId);
```

ProductName

Onesies set

Sunsuit

Pajama set

SUM()

8

Welcome to 4GL Languages

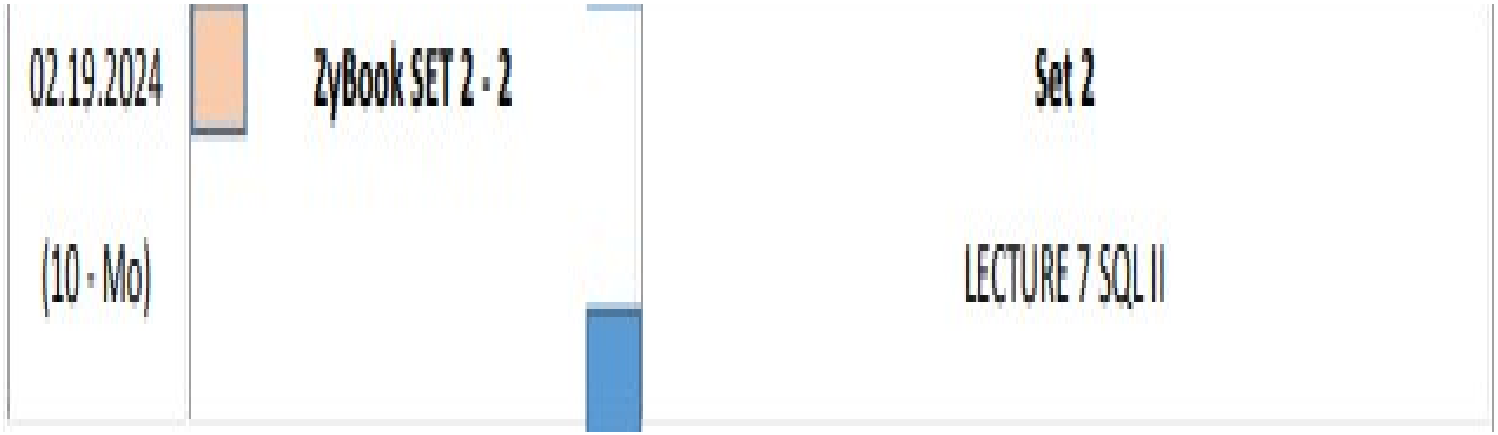
a. All products that are sold-out.

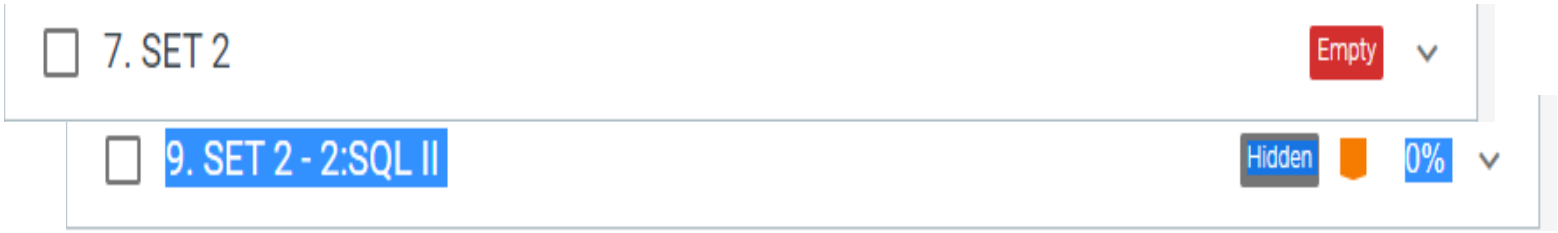
b. All products that are in stock.

c. All of the products in the database.

d. No products are selected.

At 5:00 PM .

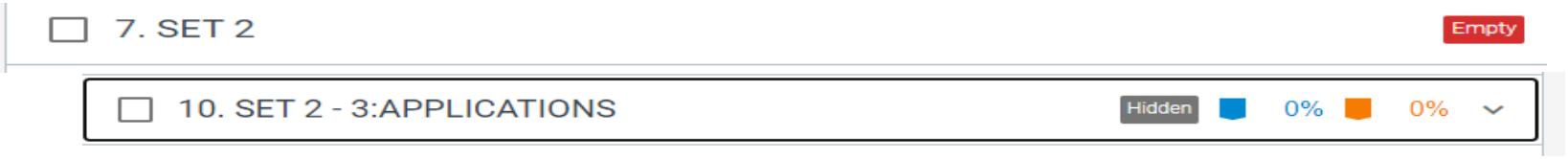




VH work on
SET 2 – 1: SQL II

Next

02.21.2024 (11 - We)	ZyBook SET 2-3	Set 2 LECTURE 10 APPLICATIONS



VH, unHIDE

From 5:05 to 5:15 PM – 5 minutes.

02.19.2024 (10 - Mo)	ZyBook SET 2 - 2	Set 2 LECTURE 7 SQL II

CLASS PARTICIPATION 20 points — 20% of Total

SQL II

Class 10 END PARTICIPATION
Not available until Feb 19 at 5:05pm | Due Feb 19 at 5:15pm | 100 pts

VH, publish

This is a synchronous online class.

Attendance is required.

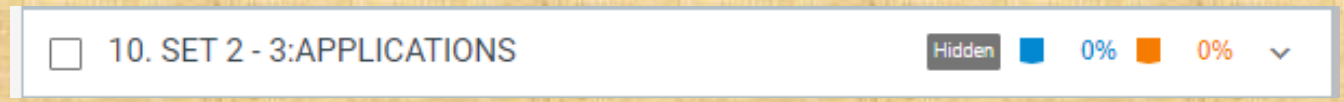
Recording or distribution of class materials is prohibited.

1. At the beginning of selected classes there is an **assessment in the first 10 minutes**. (**beige** BOX in the Detailed Syllabus)
2. At the end of selected classes there is an **assessment in the last 10 minutes**. (**blue** BOX in the Detailed Syllabus)
3. ZyBook sections will be downloaded and used for 30% of Total Score on the dates specified in the Detailed Syllabus.
4. EXAMS are in CANVAS. **No late EXAMS**.
5. I have to be **present in TEAMS in order to take any graded assignment assigned during that class.**

At 5:15 PM.

End Class 10

VH, unhide ZyBook Section 10.



VH, Download Attendance Report
Rename it:
2.19.2024 Attendance Report FINAL

VH, upload Class 10 to CANVAS