SQL

Introduction

- Structured Query Language (SQL) is the most widely used commercial relational database language.
- It was originally developed at IBM in the SEQUEL-XRM and System-R projects (1974-1977).
  - SEQUEL (Structured English Query Language)
- Almost immediately, other vendors introduced DBMS products based on SQL, and it is now a standard.
- Not all DBMS products support the full SQL standards.

The SQL language has several types:

- The Data Manipulation Language (DML)
  - The subset of SQL that allows users to pose queries and to insert, delete, and modify rows.
- The Data Definition Language (DDL)
  - The subset of SQL that supports the creation, deletion, and modification to definitions for tables and views.
- Triggers and Advanced Integrity Constraints
  - The new SQL:1999 standard supports triggers, which are actions executed by the DBMS whenever changes to the database meet conditions specified in the trigger

Basic SQL Query

- The basic form of SQL

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 relations in relation-list.

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

- qualification
  - Comparisons (Attr op const or Attr1 op Attr2, where op is one of $<, >, =, \leq, \geq, \neq$) combined using *AND*, *OR* and *NOT*.

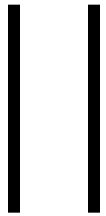
- DISTINCT
  - an optional keyword indicating that the answer should not contain duplicates.
- Default is that duplicates are ***not*** eliminated.

- *SELECT* clause
  - specifies columns to be retained in the result.
- *FROM* clause
  - specifies a cross-product of tables.
- *WHERE* clause (optional)
  - specifies selection conditions on the tables mentioned in the FROM clause.
- An SQL query intuitively corresponds to a relational algebra expression involving selections, projections, and cross-products.

SELECT DISTINCT $a_1, a_2, \ldots, a_n$

FROM $R_1, R_2, \ldots, R_m$

WHERE P



$\Pi_{a1, a2, \ldots, an} (\sigma_P (R_1 \times R_2 \times \ldots \times R_m))$

Find the names of all branches in the loan relation.

SELECT branch-name

FROM Loan

Loan

branch_name	loan_number	amount
CUHK	222	2
CMTR	333	1
CUHK	777	2

Result

branch_name
CUHK
CMTR
CUHK

- To remove duplications

SELECT DISTINCT branch-name

FROM Loan

Loan

branch_name	loan_number	amount
CUHK	222	2
CMTR	333	1
CUHK	777	2

Result

branch_name
CUHK
CMTR

$\Pi_{branch\_name}(Loan)$

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

- # Conceptual Evaluation Strategy

1. Compute the cross-product of *relation-list*.
2. Select resulting tuples if they fulfill *qualifications*.
3. Delete attributes that are not in *target-list*.
4. If *DISTINCT* is specified, eliminate duplicate rows.

(This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.)

Find the names of sailors who have reserved boat number 103

sid	bid	day
22	101	10/10/05
58	103	11/12/05

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

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

S.sid	sname	rating	age	R.sid	bid	day
22	dustin	7	45.0	22	101	10/10/05
22	dustin	7	45.0	58	103	11/12/05
31	lubber	8	55.5	22	101	10/10/05
31	lubber	8	55.5	58	103	11/12/05
58	rusty	10	35.0	22	101	10/10/05
58	rusty	10	35.0	58	103	11/12/05

Row remains after selection.

Result

$\Pi_{sname}\, \sigma_{bid=103,\ S.sid=R.sid}$ (Sailors X Reserves)

$\Pi_{sname}\, \sigma_{bid=103,}$ (Sailors $\bowtie$ Reserves)

More Examples

- Given the following schema:

  Sailors (sid: integer, sname: string, rating:integer, age: real)

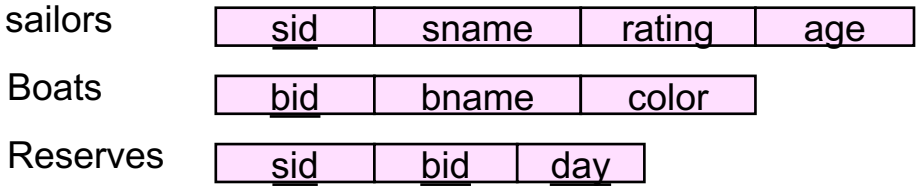
  Boats (bid: integer, bname: string, color: string)

  Reserves (sid: integer, bid: integer, day: date)

Sailors	sid	sname	rating	age

Boats	bid	bname	color

Reserves	sid	bid	day

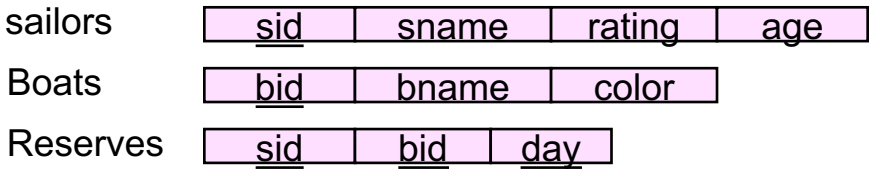


Find the sids of sailors who have reserved a red boat.

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

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

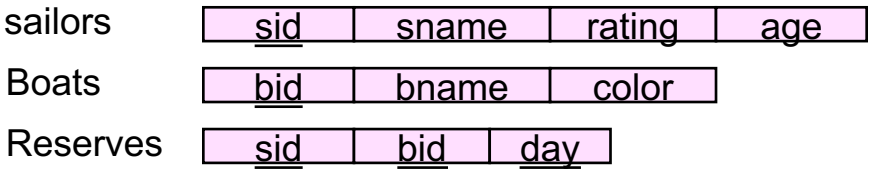
```
SELECT  S.sname
FROM  Sailors S, Reserves R, Boat B
WHERE  S.sid = R.sid AND R.bid = B.bid AND
          B.color = 'red'
```



Find the colors of boats reserved by Lubber.

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

(In general, there may be more than one sailor called Lubber. In this case, it will return the colors of boats reserved by all such Lubber's).



Find the names of sailors who have reserved at least one boat.

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

(If a sailor has not made a reservation, the second step in the conceptual evaluation strategy would eliminate all rows in the cross-product that involve this sailor).

Expressions and Strings

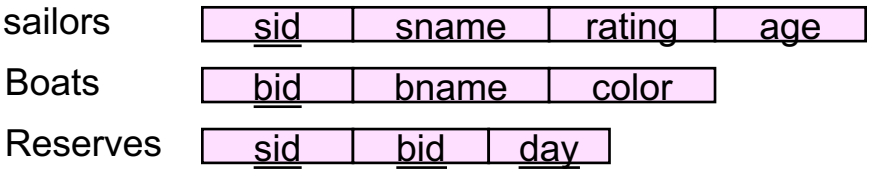
```
SELECT  S.age, age1 = S.age – 5,  2 * S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: Find triples (of ages of sailors and two fields defined by expressions) for sailors whose names begin and end with B and contain at least three characters.

- AS and = are two ways to name fields in result.
- LIKE is used for string matching.
- `_' stands for any one character (exact one character)
- `%' stands for 0 or more arbitrary characters.

Union, Intersect, and Except

- SQL provides three set-manipulation constructs that extend the basic query form presented earlier.
  - Union ($\cup$)
  - Intersection ($\cap$)
  - Except ($-$)
- many systems recognize the keyword MINUS for EXCEPT
- By default, duplicates are eliminated



Find sid's of sailors who've reserved a red or a green boat

- UNION: compute the union of any two union-compatible sets of tuples (which are themselves the result of SQL queries).
- If we replace OR by AND in the first version, what do we get?

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

```
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'
```

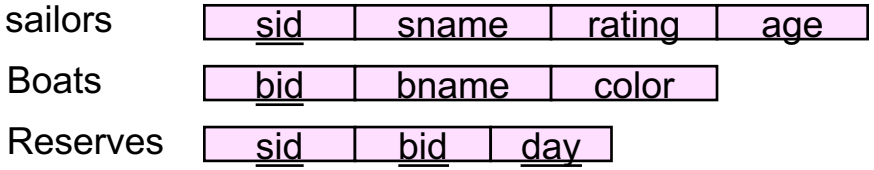
sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Find sid's of sailors who've reserved a red and a green boat

- INTERSECT: compute the intersection of any two union-compatible sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.

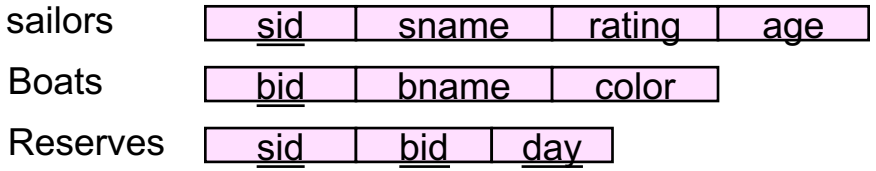
```
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE   R.bid=B.bid
            AND B.color='red'
INTERSECT
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE   R.bid=B.bid
            AND B.color='green'
```

```
SELECT  S.sid
FROM  Sailors S, Boats B1, Reserves R1,
          Boats B2, Reserves R2
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')
```



Find sid's of all sailors who've reserved red boat but not green boat.

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



Find sid's of all sailors who have a rating of 10 or reserved boat 104

```
SELECT  S.sid
FROM  Sailors S
WHERE  S.rating = 10
UNION
SELECT  R.sid
FROM  Reserves R
WHERE  R.bid=104
```

Nested Queries

- A nested query is a query that has another query embedded within it.
- The embedded query is called a subquery.
- The embedded query can be a nested query itself.
  - Queries may have very deeply nested structures.

Find names of sailors who've reserved boat #103:

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

- A very powerful feature of SQL: a WHERE clause can itself contain an SQL query! (Actually, so can FROM clauses.)
- To find sailors who've *not* reserved #103, use NOT IN.
- To understand semantics of nested queries, think of a *nested loops* evaluation:
  - *For each Sailors tuple, check the qualification by computing the subquery.*

Correlated Nested Queries

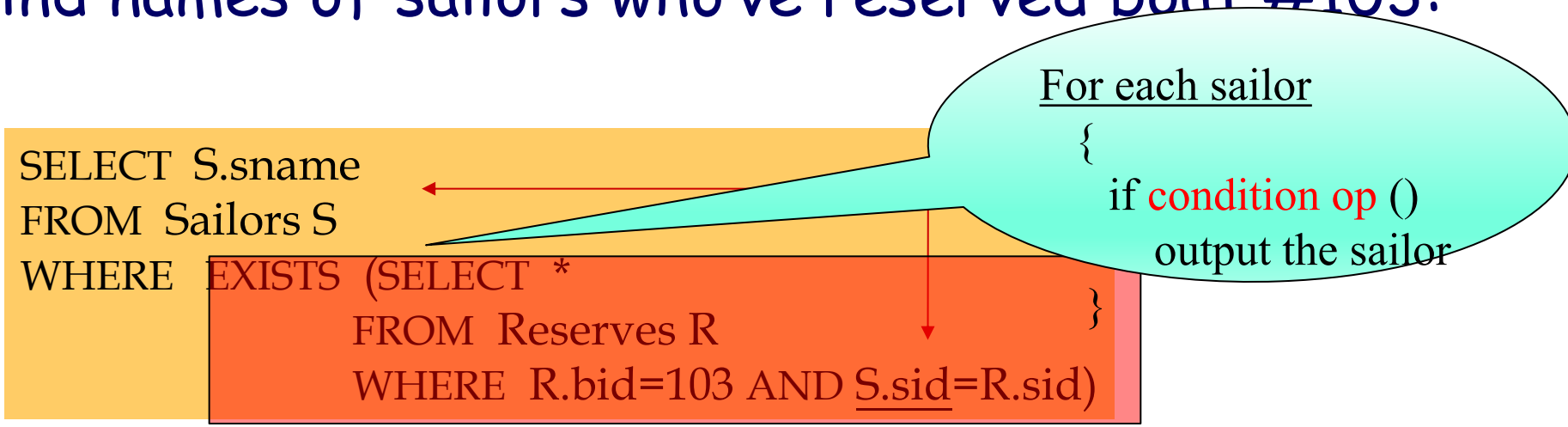
```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

- In the previous example, the inner subquery is independent of the outer query.

- In general, the inner subquery could depend on the row currently being examined in the outer query.

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Find names of sailors who've reserved boat #103:



- EXISTS is another set comparison operator, which allows us to test whether a set is nonempty.

- *For the query "finds sailors with at most one reservation for boat #103", we use* UNIQUE instead of EXISTS, and * is replaced by *R.day*

Set-comparison Operators

- We've already seen IN, EXISTS and UNIQUE.
  We can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *op* ANY, *op* ALL
  - Where op is one of the arithmetic comparison operator
    $>, <, =, \geq, \leq, \neq$
  - SOME is also available, but it is just a synonym for ANY.

- Example: Find sailors whose rating is greater than that of some sailor called Chris:

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

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Find sailors whose rating is better than every sailor called Chris.

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

Find the sailors with the highest rating.

```
SELECT  *
FROM  Sailors S
WHERE  S.rating >= ALL  (SELECT  S2.rating
                         FROM  Sailors S2)
```

- Rewriting INTERSECT queries using IN

Find the names of sailors who've reserved both a red and a green boat:

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

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

Division in SQL

Find sailors who've reserved all boats.

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
        ((SELECT  B.bid
          FROM  Boats B)
          EXCEPT
          (SELECT  R.bid
           FROM  Reserves R
           WHERE  R.sid=S.sid))
```

All boats

All boats reserved by S

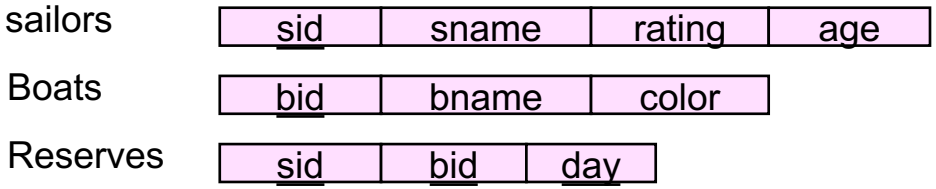
sailors: sid | sname | rating | age

Boats: bid | bname | color

Reserves: sid | bid | day

Note that this query is correlated – for each sailor S, we check to see that the set of boats reserved by S includes every boat.

Logic: there does not exist any boat that is not reserved by S



An Alternative way to write the previous query without using EXCEPT

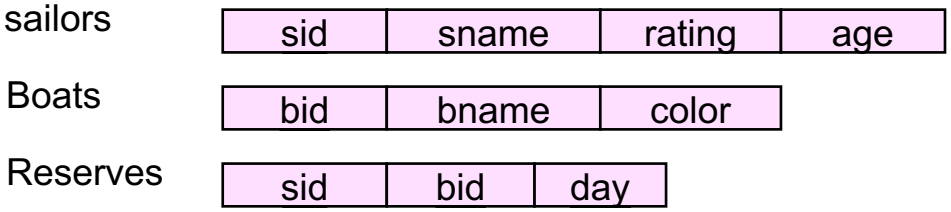
```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS  (SELECT  B.bid
                    FROM  Boats B
                    WHERE  NOT EXISTS  (SELECT  R.bid
                                        FROM  Reserves R
                                        WHERE  R.bid=B.bid
                                               AND R.sid=S.sid))
```

Intuitively, for each sailor we check that there is no boat that has not been reserved by this sailor.

Aggregate Operators

- SQL allows the use of arithmetic expressions.
- SQL supports five aggregate operations, which can be applied on any column of a relation.

COUNT([DISTINCT] A)	The number of (unique) values in the A column.
SUM ([DISTINCT] A)	The sum of all (unique) values in the A column.
AVG ([DISTINCT A)	The average of all (unique) values in the A column.
MAX (A)	The maximum value in the A column.
MIN (A)	The minimum value in the A column.



Find the average age of all sailors

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

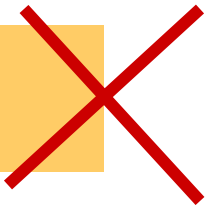
Find the average age of sailors with rating of 10

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

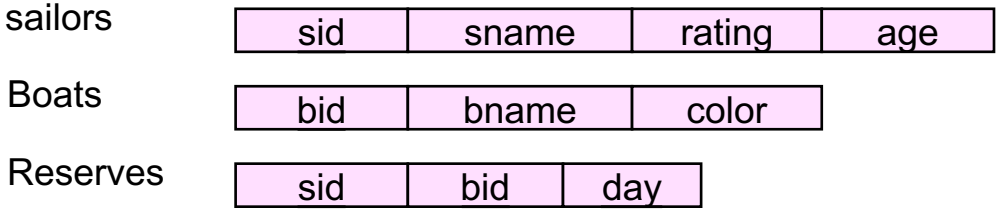
sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Find the name and age of the oldest sailor

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



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

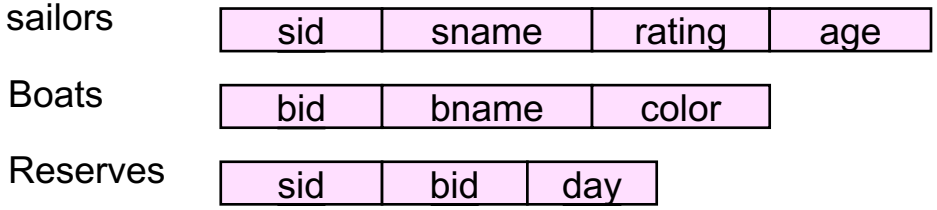


Count the number of Sailors

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

Count the number of different sailor names

```
SELECT  COUNT (DISTINCT S.name)
FROM  Sailors S
```



Aggregate operations offer an alternative to the ANY and ALL constructs.

Find the names of sailors who are older than the oldest sailor with a rating of 10.

```
SELECT  S.name
FROM  Sailors S
WHERE  S.age > ( SELECT MAX (S2.age)
                 FROM  Sailors S2
                 WHERE  S2.rating = 10)
```

Group by and Having

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

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

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

For i = 1, 2, ... , 10:

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

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

- To write such queries, we need a major extension to the basic SQL query form, namely the Group BY clause.

- The extension also includes an optional HAVING clause that can be used to specify qualifications over groups.

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

```
SELECT   S.rating, MIN (S.age)
FROM   Sailors S
GROUP BY   S.rating
```

The general format of GROUP BY and Having

```
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 attribute list (i) must be a subset of *grouping-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 *grouping-list*.)

Conceptual Evaluation

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

a. Compute the cross-product of *relation-list*
b. Select tuples that fulfill *qualification*
c. Delete fields that are not specified by *target-list, grouping-list or group-qualification*
d. Partition the remaining tuples according to the value of attributes in *grouping-list*
e. Apply the *group-qualification* to eliminate groups that are not qualified.

Conceptual Evaluation

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

The *group-qualification* is applied to eliminate some groups.

Expressions in *group-qualification* must have a *single value per group* !

Because one answer tuple is generated per qualifying group.

- In effect, an attribute in *group-qualification* must be either an argument of an aggregation operator or appears in *grouping-list*.

Find the age of the youngest sailor with age >= 18, for each rating with at least 2 such sailors

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

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 are '*unnecessary*'.
- 2nd column of result is unnamed. (Can use AS to name it)

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

rating	
7	35.0

Answer relation

Find the average age of sailors for each rating level that has at least two sailors.

```
SELECT  S.rating, AVG(S.age) AS avgage
FROM  Sailor S
GROUP BY S.rating
HAVING COUNT (*) > 1
```

OR

```
SELECT  S.rating, AVG(S.age) AS avgage
FROM  Sailor S
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
            FROM Sailors S2
            WHERE S.rating = S2.rating)
```

Answer

Rating	avgage
3	44.5
7	40.0
8	40.5
10	25.5

We can use S.rating inside the nested subquery in the HAVING because it has a single value for the current group of 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

Instance S3 of Sailor

Find the average age of sailors and the rating, who are at least 18 years old and for each rating level that has at least two sailors.

```
SELECT  S.rating, AVG(S.age) AS avgage
FROM  Sailor S
WHERE S.age >=18
GROUP BY S.rating
HAVING 1 < (SELECT COUNT (*)
                FROM Sailors S2
                WHERE S.rating = S2.rating)
```

sid	sname	rating	age
22	Dustin	7	44
25	Brutus	1	25
30	Lubber	7	22
22	Rusty	2	30
20	Zorba	7	17
24	Bouda	2	16

rating	age
7	44
7	22
1	25
2	30

S2

rating	age
7	44
7	22
7	17
1	25
2	30
2	16

Etc.

Result

rating	avgage
7	33
2	30

For the above same query

```
SELECT  S.rating, AVG(S.age) AS avgage
FROM  Sailor S
WHERE S.age >=18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

What's the final result for the above SQL ?

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

For each red boat, find the number of reservations for this boat

```
SELECT  B.bid,  COUNT (*) AS reservationcount
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
GROUP BY  B.bid
HAVING B.color = 'red'
```

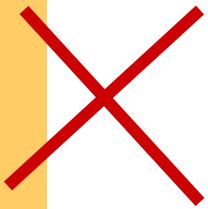
Only columns that appear in the GROUP BY clause can appear in the HAVING clause

```
SELECT  B.bid,  COUNT (*) AS reservationcount
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid AND B.color='red'
GROUP BY  B.bid
```

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

Find those ratings for which the average age is the minimum over all ratings

```
SELECT  S.rating
FROM  Sailors S
WHERE  S.age =  (SELECT  MIN (AVG (S2.age))
                 FROM Sailors S2
                 GROUP BY S2.rating)
```



Aggregate operations cannot be nested!

This query will not work even if MIN(AVG(S2.age)), which is illegal, is allowed.

In the nested query, Sailors is partitioned into groups by rating, and the average age is computed for each rating value.

For each group, applying MIN to this average age value for the group will return the same value.

Correct solution

```
SELECT  Temp.rating, Temp.avgage
FROM  (SELECT  S.rating, AVG (S.age) AS avgage
          FROM  Sailors S
          GROUP BY  S.rating) AS Temp
WHERE  Temp.avgage = (SELECT  MIN (Temp.avgage)
                                FROM  Temp)
```

It essentially computes a temporary table containing the average age for each rating value and
then finds the rating(s) for which this average age is the minimum.

Face mask – tables

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

inst_id	inst_name	inst_phone	inst_addr	TGOS_X	TGOS_Y
5901012409	家音藥局	02 -37652080	台北市松山區民生東路5段73號	121.558489	25.058709
5901012454	敦北藥局	02 -27155691	台北市松山區民生東路4段80巷5號	121.553125	25.057129
5901012490	中美藥局	02 -27627468	台北市松山區富錦街531號	121.565481	25.061285

mask(inst_id, data_time, adult_mask_num, child_mask_num)

inst_id	adult_mask_num	child_mask_num	data_time
0145080011	110	41	2020-02-06 14:10:05
0291010010	202	64	2020-02-06 14:10:05
0421040011	200	50	2020-02-06 14:10:05

record(id_card, get_time, inst_id)

<<THIS IS FAKE DATA

id_card	inst_id	get_time
A10166	5912012022	2020-02-05 17:58:50
F10705	5912013421	2020-02-06 14:06:50
W10295	5912013207	2020-02-05 14:19:43

https://g0v.hackmd.io/@kiang/mask-info?fbclid=IwAR2W2yCZ12Gxg_SzAHPe3jpU3y25sP_Y_MNCIVYvyV2Q_DigQxs-Qw4Nq64

Face mask – tasks

1. 列出2/6(實名制第一天)晚上十點時口罩庫存縣市，根據口罩庫存排序並列出前十高，包含縣市、藥局間數、口罩數
2. 列出新竹市東區在2/28中午12點時還有成人口罩的藥局名稱、電話、地址、口罩數目，並依照成人口罩數目排序
3. 列出從2/6晚上十點與2/29晚上十點庫存成長率的縣市、口罩剩餘數、成長比率，依照比率由高至低排序
4. 列出領過口罩次數各有多少人
5. Find the average number of adult face mask in each clinic
6. Find the phone number of the clinic ID that has the maximal number of adult face mask
7. Given one location(EC in NCTU(120.997436, 24.786980)), find the nearest clinic ID, address, count and distance that has face mask at 2020/02/28 12:00

- 列出2/6(實名制第一天)晚上十點時口罩庫存縣市，根據口罩庫存排序並列出前十高，包含縣市、藥局間數、口罩數

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

mask(inst_id, data_time, adult_mask_num, child_mask_num)

record(id_card, get_time, inst_id)

```
SELECT substring(i.inst_addr, 1, 3) as city, COUNT(i.inst_id) as inst_num,
SUM(m.adult_mask_num)+SUM(m.child_mask_num) as total_mask_num
FROM mask as m, institute as i
WHERE m.inst_id = i.inst_id
AND m.data_time LIKE '2020-02-06 22:00:%'
GROUP BY substring(i.inst_addr, 1, 3)
ORDER BY total_mask_num DESC
LIMIT 10;
```

```
+--------+----------+----------------+
| city   | inst_num | total_mask_num |
+--------+----------+----------------+
| 新北市 |      693 |          38627 |
| 臺中市 |      427 |          27817 |
| 臺南市 |      386 |          24774 |
| 高雄市 |      453 |          24244 |
| 臺北市 |      392 |          14531 |
| 屏東縣 |      169 |          12818 |
| 雲林縣 |      189 |          12189 |
| 彰化縣 |      184 |          11988 |
| 桃園市 |      350 |           7443 |
| 南投縣 |       61 |           5513 |
+--------+----------+----------------+
10 rows in set (0.88 sec)
```

- 列出新竹市東區在2/28中午12點時還有成人口罩的藥局名稱、電話、地址、口罩數目，並依照成人口罩數目排序

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

mask(inst_id, data_time, adult_mask_num, child_mask_num)

record(id_card, get_time, inst_id)

```
SELECT i.inst_name, i.inst_phone, i.inst_addr, m.adult_mask_num, m.child_mask_num
FROM mask as m, institute as i
WHERE m.inst_id = i.inst_id
AND m.data_time LIKE '2020-02-28 12:00:%'
AND i.inst_addr LIKE '新竹市東區%'
AND m.adult_mask_num > 0
ORDER BY m.adult_mask_num DESC
```

inst_name	inst_phone	inst_addr	adult_mask_num	child_mask_num
長明藥局	(03)5320541	新竹市東區復中里中央路１７２號２樓	605	1318
上允藥局	(03)6687664	新竹市東區光復路１段３６０之１７號１樓	401	201
康田大藥局	(03)5638217	新竹市東區金山街１０２號１樓	398	200
國豐大藥局	(03)5779960	新竹市東區光復路一段３６２－１８號	397	208
仁美藥局	(03)5625398	新竹市東區振興里振興路５２號	394	546
澄清藥局	(03)5246942	新竹市東區西大路３７４號１樓	392	1024
新康藥局	(03)5256938	新竹市東區林森路１８８號	386	554
新竹實和藥局	(03)5227636	新竹市東區中正里中山路２７號	276	1252
博正藥局	(03)5436280	新竹市東區民族路８４號	274	698
誠真藥局	(03)5278233	新竹市東區中正里中正路５７號２樓	263	1489
半津藥局	(03)5320780	新竹市東區民族路６２號１樓	170	332
健康小熊藥局	(03)5622958	新竹市東區西大路１４２號２樓	116	692
幼安藥局	(03)5316337	新竹市東區三民里民族路１２８號	112	426
中央藥局	(03)5425657	新竹市東區中央路１９７號	96	1206
杏康健保特約藥局	(03)5772966	新竹市東區光復路一段５２５巷２３號１樓	64	85
李藥局	(03)5779177	新竹市東區新莊里長春街３７號	36	19
南門藥局	(03)5222665	新竹市東區關帝里南門街８２號	32	856
芮成藥局	(03)5320622	新竹市東區北大路９２巷５９號１樓	32	1104
孟德爾領藥局	(03)5357177	新竹市東區東大路一段１１８號１樓	26	440
東海藥局	(03)5772831	新竹市東區光復路一段２１３號１樓	21	16
博弘藥局	(03)5619470	新竹市東區南大路３８７號	16	323
法瑪士藥局	(03)5152255	新竹市東區鐵道路2段8號	15	62
中美藥局	(03)5283828	新竹市東區南大里西大路一ｏ二巷三號一樓	14	308
合康連鎖藥局光復店	(03)5670111	新竹市東區光復路１段３７８號	10	0
新芳鄰藥局	(03)5622183	新竹市東區南大路４０２號	6	376
哈佛藥局	(03)5326777	新竹市東區北大路７２－１號	6	492
長青藥師藥局	(03)5439673	新竹市東區民生路１４５號	5	167
濟祐藥局	(03)5350083	新竹市東區北大路６６號之１	2	558
明湖健保藥局	(03)5191788	新竹市東區明湖路３９８號	2	262
金民好藥局	(03)5794698	新竹市東區光復路一段５４４巷３６號１樓	2	140
小樹藥局	(03)5670099	新竹市東區慈濟路３號（３樓）	2	44
安康藥局	(03)5437048	新竹市東區民族路１１２號	1	420

32 rows in set (0.64 sec)

- 列出從2/6晚上十點與2/29晚上十點庫存成長率的縣市、口罩剩餘數、成長比率，依照比率由高至低排序

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

mask(inst_id, data_time, adult_mask_num, child_mask_num)

record(id_card, get_time, inst_id)

```
SELECT lastm.city, last_total, first_total, last_total / first_total as growth_rate
FROM
  (SELECT substring(institute.inst_addr, 1, 3) as city, (SUM(mask.adult_mask_num) +
SUM(mask.child_mask_num)) as last_total
   FROM mask, institute
   WHERE data_time LIKE '2020-02-29 22:00:%'
   AND mask.inst_id = institute.inst_id
   GROUP BY city) as lastm,
  (SELECT substring(institute.inst_addr, 1, 3) as city, (SUM(mask.adult_mask_num) +
SUM(mask.child_mask_num)) as first_total
   FROM mask, institute
   WHERE data_time LIKE '2020-02-06 22:00:%'
   AND mask.inst_id = institute.inst_id
   GROUP BY city) as firstm
WHERE lastm.city = firstm.city
GROUP BY lastm.city
ORDER BY growth_rate DESC
```

city	last_total	first_total	growth_rate
金門縣	13859	10	1385.9000
基隆市	155386	1114	139.4847
澎湖縣	64856	612	105.9739
臺東縣	125441	1420	88.3387
宜蘭縣	190280	2247	84.6818
桃園市	536743	7443	72.1138
苗栗縣	101756	1775	57.3273
嘉義縣	245695	4481	54.8304
花蓮縣	225505	4172	54.0520
雲林縣	580348	12189	47.6124
臺南市	1135510	24774	45.8347
嘉義市	138274	3099	44.6189
臺北市	596986	14531	41.0836
屏東縣	484653	12818	37.8103
高雄市	915542	24244	37.7637
新竹縣	68550	1967	34.8500
彰化縣	413676	11988	34.5075
臺中市	831594	27817	29.8952
南投縣	162932	5513	29.5541
新北市	937295	38627	24.2653
連江縣	24412	1015	24.0512
新竹市	47648	2404	19.8203

22 rows in set (0.72 sec)

- 列出領過口罩次數各有多少人

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

mask(inst_id, data_time, adult_mask_num, child_mask_num)

record(id_card, get_time, inst_id)

```
SELECT record_sum.cnt, SUM(record_sum.cnt) as resident_count
FROM
    (SELECT COUNT(*) as cnt
    FROM record as r
    GROUP BY r.id_card) as record_sum
GROUP BY record_sum.cnt
```

```
+-----+----------------+
| cnt | resident_count |
+-----+----------------+
|   1 |           9287 |
|   2 |          17246 |
|   3 |          11163 |
|   4 |           2304 |
+-----+----------------+
4 rows in set (0.03 sec)
```

- Find the average number of adult face mask in each clinic

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

mask(inst_id, data_time, adult_mask_num, child_mask_num)

record(id_card, get_time, inst_id)

```
SELECT inst_id, AVG(adult_mask_num) as avg_adult
FROM mask
WHERE data_time LIKE '2020-02-__ 22:00:%'
GROUP BY inst_id
```

5946020056	38.4762
5946030025	224.5714
5946030034	99.3636
5946041028	149.6522
5946061093	693.8261
5946061128	151.2727
5946080552	761.5652
5946091108	665.5217
5946091135	151.6364
5946101065	0.1429
5990010515	19.0500
5990010524	8.8500
5990010533	9.8667
5990010542	201.9091
5990010631	9.7143
5990010668	178.9000
5990010677	9.7273
5990020020	28.5263
5990030044	0.0000
5990030062	0.5000

- Find the phone number of the clinic ID that has the maximal number of adult face mask

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)

mask(inst_id, data_time, adult_mask_num, child_mask_num)

record(id_card, get_time, inst_id)

```
SELECT i.inst_phone
FROM (
    SELECT inst_id, MAX(adult_mask_num) as MAX_ID_adult
    FROM mask
    GROUP BY inst_id
    ORDER BY MAX_ID_adult DESC
    LIMIT 1
    ) as max_institute, institute as i
WHERE max_institute.inst_id = i.inst_id;
```

```
+-------------+
| inst_phone  |
+-------------+
| 05 -2263294 |
+-------------+
1 row in set (1.89 sec)
```

- Given one location(EC in NCTU(120.997436, 24.786980)), find the nearest clinic ID, address, count and distance that has face mask at 2020/02/28 12:00

institute(inst_id, inst_name, inst_addr, inst_phone, TGOS_X, TGOS_Y)
mask(inst_id, data_time, adult_mask_num, child_mask_num)
record(id_card, get_time, inst_id)

```
SELECT i.inst_id, i.inst_addr, m.adult_mask_num, st_distance_sphere(POINT(i.TGOS_X, i.TGOS_Y),
POINT(120.997436, 24.786980)) as distance
FROM institute as i, mask as m
WHERE m.data_time LIKE '2020-02-28 12:00:%'
AND m.adult_mask_num > 0
AND m.inst_id = i.inst_id
ORDER BY distance ASC
LIMIT 1;
```

inst_id	inst_addr	adult_mask_num	distance
5912013252	新竹市東區光復路一段544巷36號1樓	2	1391.533316075414

56

COVID19 – tables

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

```
+------------------+---------------+----------------------+-----------------------+
|  1 |   30 | male | Chaohu City, Hefei City | Anhui    | China   |   31.647 |   117.717
| 2020-01-22        | yes             | 2020-01-17          | Wuhan                    |
+----+------+------+-------------------------+----------+---------+----------+-----------
```

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

province	country	data_date	confirmed_num	deaths_num	recovered_num
Taiwan	Taiwan	2020-01-22	1	0	0
Taiwan	Taiwan	2020-01-23	1	0	0
Taiwan	Taiwan	2020-01-24	3	0	0
Taiwan	Taiwan	2020-01-25	3	0	0
Taiwan	Taiwan	2020-01-26	4	0	0

https://www.kaggle.com/sudalairajkumar/novel-corona-virus-2019-dataset#time_series_covid_19_recovered.csv

COVID19 – tasks

1. 確診年齡區間與人數對應(30~40歲累計有幾人、40~50歲累計有幾人)
2. 性別與人數對應
3. 第一起中國境外旅遊史的病患資訊
4. 2/26~3/3之間，各地區確診人數增加比例，由大到小排序
5. 資料中，各地區開始有確診案例的日期
6. 在3/6時，確診人數比台灣多的地區有幾個，同時顯示有確診案例的地區總共有幾個
7. 列出各地區在2/28增加的確診案例個數，數量由大到小排序

- 確診年齡區間與人數對應(30~40歲累計有幾人、40~50歲累計有幾人)

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select if((age mod 10) < 0, 'null', concat(convert((age mod 10)*10, CHAR), '-', convert(((age mod 10)+1)*10-1, CHAR))) as age_range, count(*) as cnt
from patient
group by age mod 10;
```

age_range	cnt
null	12175
0-9	122
10-19	90
20-29	108
30-39	95
40-49	94
50-59	117
60-69	91
70-79	98
80-89	92
90-99	91

59

- 性別與人數對應

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select sex, count(*) as cnt
from patient
group by sex;
```

sex	cnt
female	556
male	707
N/A	11910

60

- 第一起中國境外旅遊史的病患資訊

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select * from patient
where country != 'China'
and date_confirmation != '0000-00-00'
order by date_confirmation asc
limit 1;
```

```
+------------------+----------------+----------------------+--------------------------+
| id  | age  | sex    | city    | province           | country  | latitude | longitude
| date_confirmation | lives_in_Wuhan | travel_history_dates | travel_history_location |
+-----+------+--------+---------+--------------------+----------+----------+-----------
+------------------+----------------+----------------------+--------------------------+
| 665 |   61 | female | Bangkok | Bangkok Metropolis | Thailand |  13.7666 |   100.536
| 2020-01-12        | yes            | 2020-01-08           | Wuhan                    |
+-----+------+--------+---------+--------------------+----------+----------+-----------
```

- 2/26~3/3之間，各地區確診人數增加比例，由大到小排序

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select reg_a.province, reg_a.country, reg_b.confirmed_num as earlier_data,
reg_a.confirmed_num as later_data,
(reg_a.confirmed_num-reg_b.confirmed_num)/reg_b.confirmed_num as increase_ratio
from region_acc as reg_a, region_acc as reg_b
where reg_a.province = reg_b.province
and reg_a.country = reg_b.country
and reg_a.data_date = '2020-03-03'
and reg_b.data_date = '2020-02-26'
order by increase_ratio desc;
```

province	country	earlier_data	later_data	increase_ratio
	Norway	1	32	31.0000
	Iran	139	2336	15.8058
	Spain	13	165	11.6923
	France	18	204	10.3333
	Sweden	2	21	9.5000
	Austria	2	21	9.5000
	Germany	27	196	6.2593
	Iraq	5	32	5.4000
	Israel	2	12	5.0000
	Italy	453	2502	4.5232
	South Korea	1261	5186	3.1126
	Croatia	3	9	2.0000
	Oman	4	12	2.0000
	Romania	1	3	2.0000
	Pakistan	2	5	1.5000
	Japan	189	293	0.5503
Taiwan	Taiwan	32	42	0.3125
	Singapore	93	110	0.1828
Hong Kong	Hong Kong	91	100	0.0989
Beijing	Mainland China	400	414	0.0350
Liaoning	Mainland China	121	125	0.0331
Hubei	Mainland China	65187	67217	0.0311
Hebei	Mainland China	312	318	0.0192
Sichuan	Mainland China	531	538	0.0132
Tianjin	Mainland China	135	136	0.0074
Fujian	Mainland China	294	296	0.0068

- 資料中，各地區開始有確診案例的日期

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select province, country, min(data_date) as start_date
from region_acc
group by province, country
order by start_date asc;
```

province	country	start_date
Sichuan	Mainland China	2020-01-22
	Thailand	2020-01-22
Ningxia	Mainland China	2020-01-22
Hainan	Mainland China	2020-01-22
Gansu	Mainland China	2020-01-22
Anhui	Mainland China	2020-01-22
Shandong	Mainland China	2020-01-22
Yunnan	Mainland China	2020-01-22
Henan	Mainland China	2020-01-22
Guangdong	Mainland China	2020-01-22
Beijing	Mainland China	2020-01-22
Jiangsu	Mainland China	2020-01-22
Hubei	Mainland China	2020-01-22
	Japan	2020-01-22
Shanghai	Mainland China	2020-01-22
Zhejiang	Mainland China	2020-01-22
Inner Mongolia	Mainland China	2020-01-22
Guangxi	Mainland China	2020-01-22
Tianjin	Mainland China	2020-01-22
Fujian	Mainland China	2020-01-22
Jiangxi	Mainland China	2020-01-22
Xinjiang	Mainland China	2020-01-22
Washington	US	2020-01-22
	South Korea	2020-01-22

- 在3/6時，確診人數比台灣多的地區有幾個，同時顯示有確診案例的地區總共有幾個

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select inner_res.cnt as cnt, count(ra.province) as total_cnt
from region_acc as ra,
(
    select count(ra.province) as cnt
    from region_acc as ra,
    (
        select confirmed_num
        from region_acc
        where data_date = '2020-03-06'
        and province = 'Taiwan'
        and country = 'Taiwan'
    ) as new_taiwan_data
    where ra.data_date = '2020-03-06'
    and ra.confirmed_num > new_taiwan_data.confirmed_num
) as inner_res
where ra.data_date = '2020-03-06';
```

```
+------+-----------+
| cnt  | total_cnt |
+------+-----------+
|   44 |       119 |
+------+-----------+
```

- 列出各地區在2/28增加的確診案例個數，數量由大到小排序

patient(id, age, sex, city, province, country, latitude, longitude, date_confirmation, lives_in_Wuhan, travel_history_dates, travel_history_location)

region_acc(province, country, data_date, confirmed_num, deaths_num, recovered_num)

```
select later_data.province, later_data.country,
 earlier_data.confirmed_num as earlier_patient_num,
 later_data.confirmed_num as later_patient_num,
 (later_data.confirmed_num-earlier_data.confirmed_num) as new_patient_num
from region_acc as later_data, region_acc as earlier_data
where later_data.data_date = '2020-2-28'
and earlier_data.data_date = '2020-2-27'
and later_data.province = earlier_data.province
and later_data.country = earlier_data.country
order by new_patient_num desc;
```

province	country	earlier_patient_num	later_patient_num	new_patient_num
	South Korea	1766	2337	571
Hubei	Mainland China	65596	65914	318
	Italy	655	888	233
	Iran	245	388	143
	France	38	57	19
	Spain	15	32	17
	Japan	214	228	14
	UK	15	20	5
	Norway	1	6	5
Sichuan	Mainland China	534	538	4

Travel records – tables

user(ID, PID)

<<THIS IS FAKE DATA

ID	PID
A172059484	JB66937774
B252396231	GG66761298
H147729511	ET77584376

inbound(PID, IATA_code, flight_no, date)

PID	IATA_code	flight_no	date
JB66937774	HAN	GT176	2020-02-12
GG66761298	RGN	RH301	2020-02-06
ET77584376	BOM	SP73	2020-03-10

airport(IATA_code, city, country)

IATA_code	city	country
CAN	Guangzhou	China
KORD	Chicago	US
HKG	Hong Kong	China

http://airport.anseo.cn/
https://zh-tw.flightaware.com/live/airport/RCTP/arrivals

66

Travel records – tasks

1. Find PID of users who traveled UK and were back Taiwan on 2020/03/16
2. Find the ID of users who traveled to UK and were back Taiwan on 2020/03/16
3. Find the ID of users who were back from UK or Japan
4. Find the PID of users who were in the same plant of one confirmed case with his PID as NQ13219857 on 2020/03/06

- Find PID of users who traveled UK and were back Taiwan on 2020/03/16

user(ID, PID)

inbound(PID, IATA_code, flight_no, date)

airport(IATA_code, city, country)

Sol1:

```
select inbound.PID
from
(
    select * from airport where country='UK'
) as airport,
(
    select * from inbound where date = '2020-03-16'
) as inbound
where airport.IATA_code = inbound.IATA_code;
```

Sol2:

```
select inbound.PID
from airport, inbound
where airport.IATA_code = inbound.IATA_code
and airport.country='UK'
and inbound.date = '2020-03-16';
```

```
| XW73171764 |
| XX77962579 |
| YB28170323 |
| YL30226927 |
| YL61563022 |
| YL93852671 |
| YN64215789 |
| YP82637586 |
| YQ50972800 |
| YT76838773 |
| YU23373561 |
| YX62803618 |
| ZB71332988 |
| ZE65615859 |
| ZI55372309 |
| ZM59509852 |
| ZO34388745 |
| ZO86162571 |
| ZP76870006 |
| ZV94523464 |
| ZY30055310 |
| ZZ49855079 |
+------------+
301 rows in set (0.391 sec)
```

```
| XW73171764 |
| XX77962579 |
| YB28170323 |
| YL30226927 |
| YL61563022 |
| YL93852671 |
| YN64215789 |
| YP82637586 |
| YQ50972800 |
| YT76838773 |
| YU23373561 |
| YX62803618 |
| ZB71332988 |
| ZE65615859 |
| ZI55372309 |
| ZM59509852 |
| ZO34388745 |
| ZO86162571 |
| ZP76870006 |
| ZV94523464 |
| ZY30055310 |
| ZZ49855079 |
+------------+
301 rows in set (1.092 sec)
```

- Find the ID of users who traveled to UK and were back Taiwan on 2020/03/16

user(ID, PID)

inbound(PID, IATA_code, flight_no, date)

airport(IATA_code, city, country)

Sol1:

```
select user.ID
from
(
    select airport.*, inbound.PID, inbound.flight_no, inbound.date from
    (
        select * from airport where country = 'UK'
    ) as airport,
    (
        select * from inbound where date = '2020-03-16'
    ) as inbound
    where airport.IATA_code = inbound.IATA_code
) as airport, user
where user.PID = airport.PID;
```

Sol2:

```
select ID
from
(
    select PID
    from
    (
        select IATA_code from airport where country = 'UK'
    ) as airport,
    (
        select * from inbound where date = '2020-03-16'
    ) as inbound
    where airport.IATA_code = inbound.IATA_code
) as airport, user
where user.PID = airport.PID;
```

```
| I273014241 |
| Q262650750 |
| L240703070 |
| J131048507 |
| Q270883464 |
| F275103169 |
| M169297033 |
| R293194851 |
| S139182404 |
| H236094511 |
| Q249574234 |
| I122166015 |
| N146378212 |
| V106930552 |
| D252479885 |
| Z255518855 |
| G278874231 |
| E148315062 |
| N295648011 |
| F173912129 |
| Y207381662 |
| B100599890 |
+------------+
301 rows in set (16.993 sec)
```

```
| I273014241 |
| Q262650750 |
| L240703070 |
| J131048507 |
| Q270883464 |
| F275103169 |
| M169297033 |
| R293194851 |
| S139182404 |
| H236094511 |
| Q249574234 |
| I122166015 |
| N146378212 |
| V106930552 |
| D252479885 |
| Z255518855 |
| G278874231 |
| E148315062 |
| N295648011 |
| F173912129 |
| Y207381662 |
| B100599890 |
+------------+
301 rows in set (10.613 sec)
```

- Find the ID of users who were back from UK or Japan

user(ID, PID)

inbound(PID, IATA_code, flight_no, date)

airport(IATA_code, city, country)

Sol1:

```
select user.ID
from
(
    select TempAirport.*, inbound.PID, inbound.flight_no, inbound.date  from
    (
        select * from airport where country = 'UK' or country = 'Japan'
    ) as TempAirport, inbound
    where TempAirport.IATA_code = inbound.IATA_code
) as TempAirport, user
where TempAirport.PID = user.PID;
```

```
| L170258496 |
| V210525136 |
| S243766707 |
| G222513839 |
| O215184847 |
| A212382799 |
| F289337339 |
| F215846724 |
| R167404525 |
| Z209156173 |
| M238667693 |
| Q185634726 |
| U255587267 |
| E234027662 |
| Z285135669 |
| O224795008 |
| K169118753 |
| C271994817 |
| E249523907 |
| Q126626183 |
| H220014937 |
+------------+
89354 rows in set (2.806 sec)
```

- Find the ID of users who were back from UK or Japan

user(ID, PID)

inbound(PID, IATA_code, flight_no, date)

airport(IATA_code, city, country)

Sol2:

```
select user.ID
from
(
    select airport.*, inbound.PID, inbound.flight_no, inbound.date  from
    (
        select * from airport where country = 'UK'
    ) as airport, inbound
    where airport.IATA_code = inbound.IATA_code
) as airport, user
where airport.PID = user.PID
union
select user.ID
from
(
    select airport.*, inbound.PID, inbound.flight_no, inbound.date  from
    (
        select * from airport where country = 'Japan'
    ) as airport, inbound
    where airport.IATA_code = inbound.IATA_code
) as airport, user
where airport.PID = user.PID;
```

```
| L251650071 |
| S269666764 |
| L170258496 |
| V210525136 |
| S243766707 |
| G222513839 |
| O215184847 |
| F289337339 |
| F215846724 |
| R167404525 |
| Z209156173 |
| M238667693 |
| Q185634726 |
| U255587267 |
| E234027662 |
| Z285135669 |
| O224795008 |
| C271994817 |
| E249523907 |
| Q126626183 |
| H220014937 |
+------------+
89354 rows in set (5.787 sec)
```

- Find the PID of users who were in the same plant of one confirmed case with his PID as NQ13219857 on 2020/03/06

user(ID, PID)

inbound(PID, IATA_code, flight_no, date)

airport(IATA_code, city, country)

```
select inbound.PID, inbound.flight_no
from inbound
where inbound.flight_no in (
    select flight_no
    from inbound
    where PID = 'NQ13219857'
    and date = '2020-3-6'
);
```

ZX32888374	SY326
ZX38882022	SY326
ZX56446488	SY326
ZX62712843	SY326
ZX70360958	SY326
ZX74093042	SY326
ZX75085244	SY326
ZX87209501	SY326
ZX97828421	SY326
ZY48416956	SY326
ZY54462439	SY326
ZY57888586	SY326
ZY67665877	SY326
ZY83132876	SY326
ZY89141280	SY326
ZZ29987522	SY326
ZZ30262817	SY326
ZZ32534445	SY326
ZZ44809224	SY326
ZZ67638772	SY326
ZZ74639362	SY326
ZZ88045473	SY326
ZZ96977399	SY326

4984 rows in set (2.332 sec)

Example data

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Range Search Condition

List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- BETWEEN test includes the endpoints of range.

INSERT … VALUES

Insert a new row into Staff table supplying data for all columns.

```
INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M',
  Date'1957-05-25', 8300, 'B003');
```

INSERT using Defaults

Insert a new row into Staff table supplying data for all mandatory columns.

```
INSERT INTO Staff (staffNo, fName, lName,
                   position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones',
        'Assistant', 8100, 'B003');
```

- Or

```
INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL,
        NULL, 8100, 'B003');
```

INSERT … SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO TableName [ (columnList) ]
    SELECT …
```

UPDATE

```
UPDATE TableName
SET columnName1 = dataValue1
    [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

- *TableName* can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

UPDATE

- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy *searchCondition* are updated.
- New *dataValue(s)* must be compatible with data type for corresponding column.

UPDATE All Rows

Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary*1.03;
```

Give all Managers a 5% pay increase.

```
UPDATE Staff
SET salary = salary*1.05
WHERE position = 'Manager';
```

UPDATE Multiple Columns

Promote David Ford (staffNo='SG14') to Manager and change his salary to £18,000.

```
UPDATE Staff
SET position = 'Manager', salary = 18000
WHERE staffNo = 'SG14';
```

DELETE

DELETE FROM TableName
[WHERE searchCondition]

- *TableName* can be name of a base table or an updatable view.
- *searchCondition* is optional; if omitted, all rows are deleted from table. This does not delete table. If *search_condition* is specified, only those rows that satisfy condition are deleted.

DELETE Specific Rows

Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
WHERE propertyNo = 'PG4';
```

Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```

NULL Value

- field value unknown
  - A new employee has not been assigned a supervisor yet.

Employee

eno	ename	supervisor_eno	spouse_name

- field attribute inapplicable
  - An unmarried employee does not have a spouse

- Two rows are duplicates if matching columns are either equal or both NULL
  - implicitly NULL=NULL.
  - But for comparison in where clause, (NULL=NULL) = unknown.

- NULL is counted in COUNT(*)
- All other aggregate discard NULL values

Outer Joins

List (sid, bid) for sailors and boats they have reserved.

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

sid	bid	day
22	101	10/10/05
58	103	11/12/05

SELECT R.sid, R.bid

FROM Sailors S NATURAL LEFT OUTER JOIN Reserves R

All left relation rows (Sailors) appear in the result

sid	bid
22	101
31	null
58	103

← Use of null

Outer Joins

List (sid, bid) for sailors and boats they have reserved.

sid	bid	day
22	101	10/10/05
58	103	11/12/05

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

SELECT R.sid, R.bid

FROM Reserves R NATURAL RIGHT OUTER JOIN Sailors S

sid	bid
22	101
31	null
58	103

All right relation rows (Sailors) appear in the result

Views

List (sid, bid) for sailors and boats they have reserved.

```
CREATE VIEW  SB (sid, bid)
    AS SELECT R.sid, R.bid
        FROM Reserves R NATURAL RIGHT OUTER JOIN Sailors S
```

sid	bid
22	101
31	null
58	103

View SB is a table which is not explicitly stored in the database.

A view can be used just like a base table.

Outer Joins

sailors	sid	sname	rating	age
Boats	bid	bname	color	
Reserves	sid	bid	day	

SB

sid	bid
22	101
31	null
58	103

bid	bname	color
101	apollo	white
103	maria	blue
107	princess	pink

SELECT SB.sid, B.bid

FROM SB NATURAL FULL OUTER JOIN Boats B

sid	bid
22	101
31	null
58	103
null	107

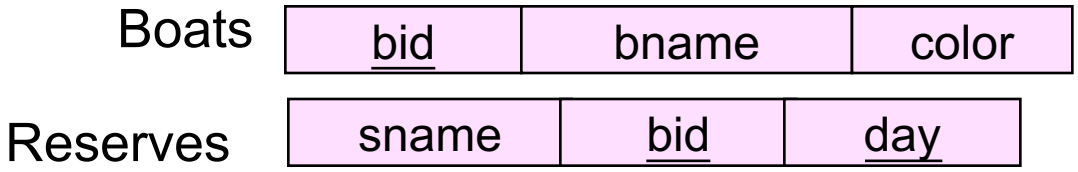
General Constraints

- Useful when more general ICs than keys are involved.
- Can use queries to express constraint.

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

Sailors

sid	sname	rating	age



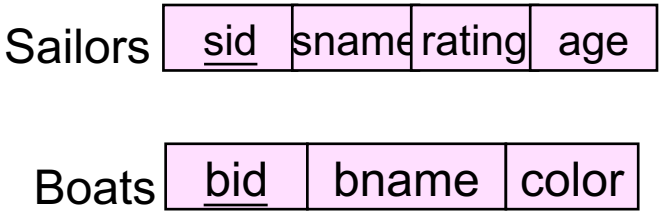
Constraints can be named.

Constraint:
Interlake boats cannot be reserved.

```
CREATE TABLE Reserves
        ( sname CHAR(10),
        bid INTEGER,
        day DATE,
        PRIMARY KEY (bid,day),
        CONSTRAINT noInterlakeRes
        CHECK (`Interlake' <>
                ( SELECT B.bname
                  FROM Boats B
                  WHERE B.bid=Reserves.bid)))
```

When a boat is inserted into Reserves or an existing row is modified, the conditional expression in the CHECK constraint is evaluated. If it evaluates to false, the command is rejected.

Constraints Over Multiple Relations



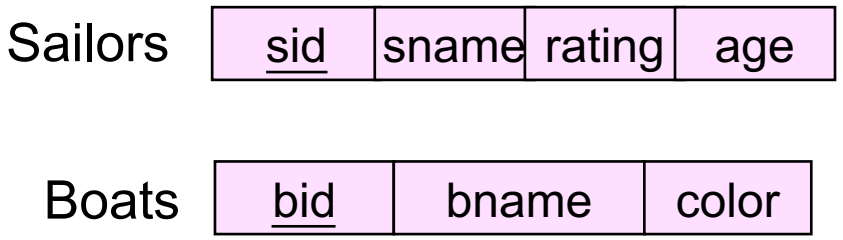
Constraint:
Number of boats plus number of sailors is < 100

```
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)
        < 100 )
```

- Awkward and wrong!
- If Sailors is empty, the number of Boats tuples can be anything!
- *A constraint is not required to hold for an empty relation*

- ASSERTION is the right solution; not associated with either table.

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



Triggers

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

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

Triggers

- Trigger
  - Defines an action that the database should take when some event occurs in the application
  - Based on Event-Condition-Action (ECA) model
- Types
  - Row-level
  - Statement-level
- Event: INSERT, UPDATE or DELETE
- Timing: BEFORE, AFTER or INSTEAD OF
- Advantages and disadvantages of triggers

95

Trigger Format

```
CREATE TRIGGER TriggerName
      BEFORE | AFTER | INSTEAD OF
      INSERT | DELETE | UPDATE [OF TriggerColumnList]
      ON TableName
      [REFERENCING {OLD | NEW} AS {OldName | NewName}
      [FOR EACH {ROW | STATEMENT}]
      [WHEN Condition]
      <trigger action>
```


96

Triggers: Example (SQL:1999)

```
CREATE TRIGGER youngSailorUpdate
    AFTER INSERT ON SAILORS
REFERENCING NEW TABLE NewSailors
FOR EACH STATEMENT
    INSERT
        INTO YoungSailors(sid, name, age, rating)
        SELECT sid, name, age, rating
        FROM NewSailors N
        WHERE N.age <= 18
```

Event

newly inserted tuples

Action

Sailors

sid	sname	rating	age

Using a BEFORE Trigger

```
CREATE TRIGGER StaffNotHandlingTooMuch
BEFORE INSERT ON PropertyForRent
REFERENCING NEW AS newrow
FOR EACH ROW
DECLARE
  vpCount          NUMBER;
BEGIN
        SELECT COUNT(*) INTO vpCount
        FROM PropertyForRent
        WHERE staffNo = :newrow.staffNo;
        IF vpCount = 100
            raise_application_error(-20000, ('Member' || :newrow.staffNo || 'already managing 100 properties');
        END IF;
END;
```


98

Triggers – Advantages

- Elimination of redundant code
- Simplifying modifications
- Increased security
- Improved integrity
- Improved processing power
- Good fit with client-server architecture

99

Summary

- SQL was an important factor in the early acceptance of the relational model; more natural than earlier, procedural query languages.
- Relationally complete; in fact, significantly more expressive power than relational algebra.
- Many alternative ways to write a query; optimizer should look for most efficient evaluation plan.
  - In practice, users need to be aware of how queries are optimized and evaluated for best results.