MONASH University

MONASH
INFORMATION
TECHNOLOGY

Week 9 – SQL Intermediate - SQL Advanced

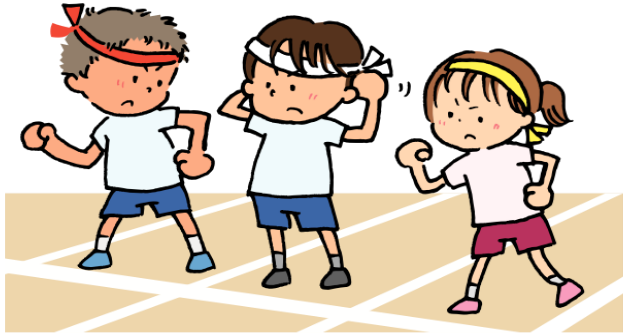
FIT3171 Databases
Semester 1 2022

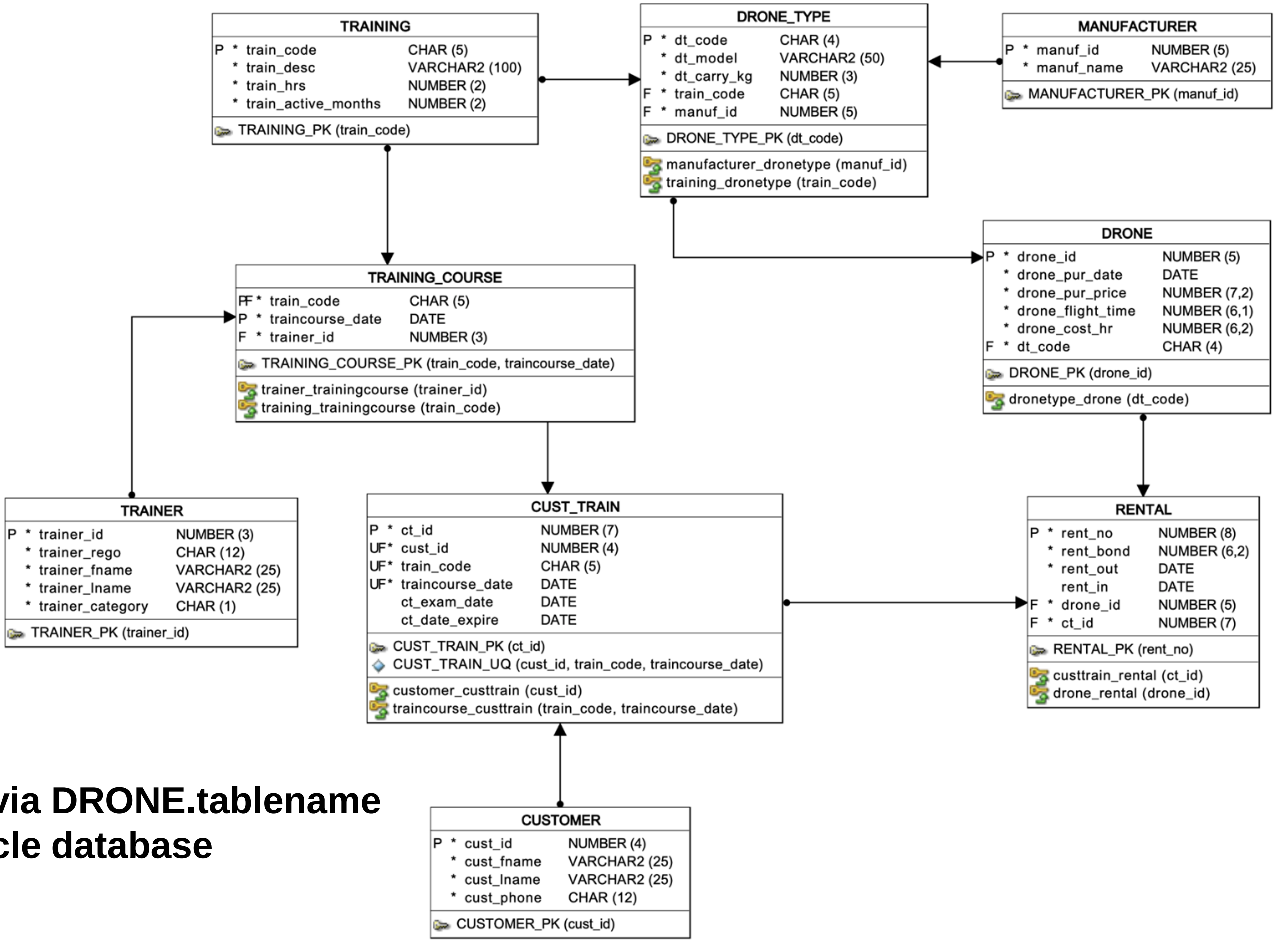
Malaysia Campus

Preparation for the workshop - ready, set ……

Please

- connect to Flux - flux.qa and be ready to answer questions
  - flux.qa/QBGYRS
- login to the Oracle database via **SQL Developer** (you will need to run the CISCO or Global VPN first if you are off campus)





Access tables via DRONE.tablename in Monash Oracle database

Aggregate Functions

- COUNT, MAX, MIN, SUM, AVG
- Example:

```
SELECT
    MAX(drone_flight_time)
FROM
    drone.drone;
```

```
SELECT
    MIN(drone_flight_time)
FROM
    drone.drone;
```

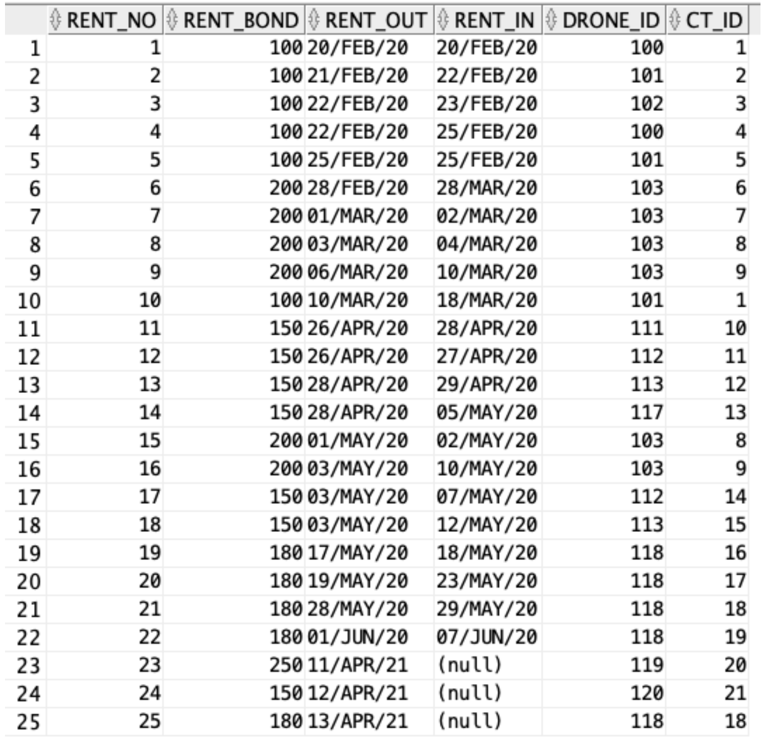
```
SELECT
    AVG(drone_flight_time)
FROM
    drone.drone;
```

```
SELECT COUNT(*)
FROM drone.drone
WHERE drone_flight_time > 100;
```

count(*) and count(column_name)

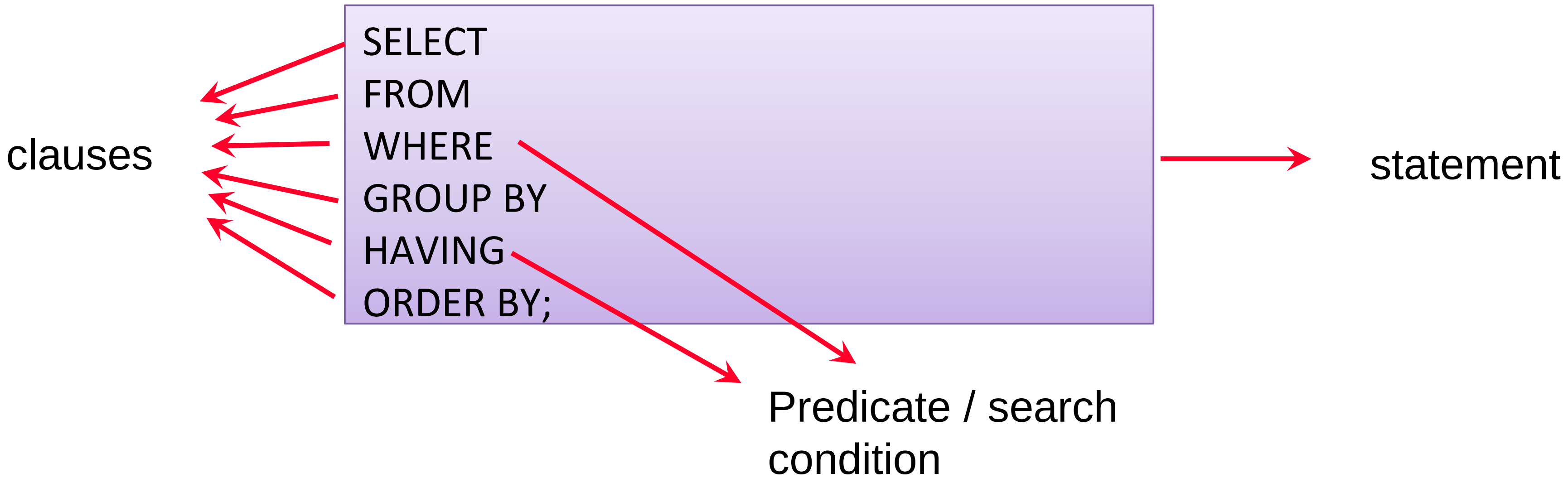
```
SQL> SELECT
  2      COUNT(*),
  3      COUNT(rent_out),
  4      COUNT(rent_in)
  5  FROM
  6      drone.rental;

  COUNT(*) COUNT(RENT_OUT) COUNT(RENT_IN)
---------- --------------- --------------
        25              25             22
```



	RENT_NO	RENT_BOND	RENT_OUT	RENT_IN	DRONE_ID	CT_ID
1	1	100	20/FEB/20	20/FEB/20	100	1
2	2	100	21/FEB/20	22/FEB/20	101	2
3	3	100	22/FEB/20	23/FEB/20	102	3
4	4	100	22/FEB/20	25/FEB/20	100	4
5	5	100	25/FEB/20	25/FEB/20	101	5
6	6	200	28/FEB/20	28/MAR/20	103	6
7	7	200	01/MAR/20	02/MAR/20	103	7
8	8	200	03/MAR/20	04/MAR/20	103	8
9	9	200	06/MAR/20	10/MAR/20	103	9
10	10	100	10/MAR/20	18/MAR/20	101	1
11	11	150	26/APR/20	28/APR/20	111	10
12	12	150	26/APR/20	27/APR/20	112	11
13	13	150	28/APR/20	29/APR/20	113	12
14	14	150	28/APR/20	05/MAY/20	117	13
15	15	200	01/MAY/20	02/MAY/20	103	8
16	16	200	03/MAY/20	10/MAY/20	103	9
17	17	150	03/MAY/20	07/MAY/20	112	14
18	18	150	03/MAY/20	12/MAY/20	113	15
19	19	180	17/MAY/20	18/MAY/20	118	16
20	20	180	19/MAY/20	23/MAY/20	118	17
21	21	180	28/MAY/20	29/MAY/20	118	18
22	22	180	01/JUN/20	07/JUN/20	118	19
23	23	250	11/APR/21	(null)	119	20
24	24	150	12/APR/21	(null)	120	21
25	25	180	13/APR/21	(null)	118	18

Anatomy of an SQL Statement - Revisited



GROUP BY

- If a GROUP BY clause is used with aggregate function, the DBMS will apply the aggregate function to the different groups defined in the clause rather than all rows.

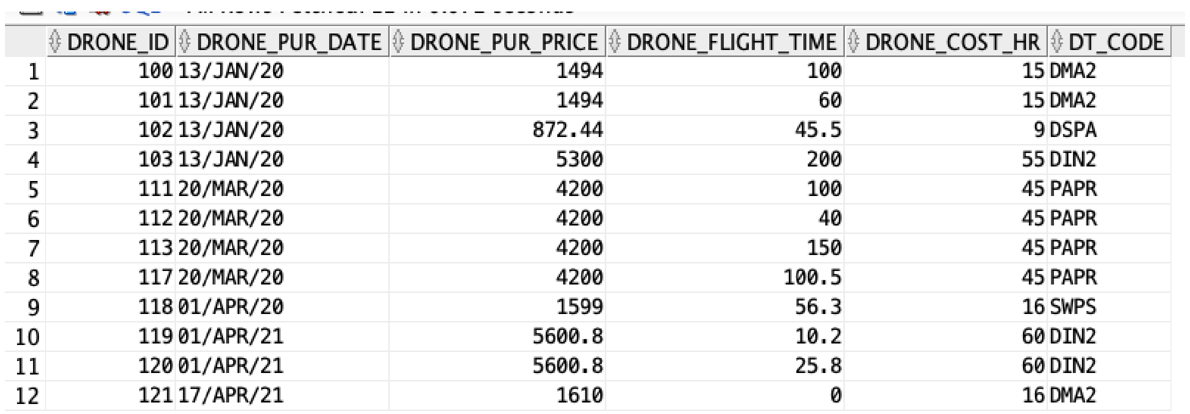
```
SELECT
 AVG(drone_flight_time)
FROM
    drone.drone;
```

```
SELECT dt_code, AVG(drone_flight_time)
FROM  drone.drone
GROUP BY  dt_code
ORDER BY dt_code;
```

```
SQL> SELECT
  2      AVG(drone_flight_time)
  3  FROM
  4      drone.drone;

AVG(DRONE_FLIGHT_TIME)
----------------------
                74.025
SQL>
SQL> SELECT
  2      dt_code,
  3      AVG(drone_flight_time)
  4  FROM
  5      drone.drone
  6  GROUP BY
  7      dt_code
  8  ORDER BY
  9      dt_code;

DT_C AVG(DRONE_FLIGHT_TIME)
---- ----------------------
DIN2             78.6666667
DMA2             53.3333333
DSPA                   45.5
PAPR                 97.625
SWPS                   56.3
```



	DRONE_ID	DRONE_PUR_DATE	DRONE_PUR_PRICE	DRONE_FLIGHT_TIME	DRONE_COST_HR	DT_CODE
1	100	13/JAN/20	1494	100	15	DMA2
2	101	13/JAN/20	1494	60	15	DMA2
3	102	13/JAN/20	872.44	45.5	9	DSPA
4	103	13/JAN/20	5300	200	55	DIN2
5	111	20/MAR/20	4200	100	45	PAPR
6	112	20/MAR/20	4200	40	45	PAPR
7	113	20/MAR/20	4200	150	45	PAPR
8	117	20/MAR/20	4200	100.5	45	PAPR
9	118	01/APR/20	1599	56.3	16	SWPS
10	119	01/APR/21	5600.8	10.2	60	DIN2
11	120	01/APR/21	5600.8	25.8	60	DIN2
12	121	17/APR/21	1610	0	16	DMA2

MONASH University

Q1. List all customer ids and the total number of courses taken by each customer:

A. select cust_id, count(*) as no_of_courses_taken
   from drone.cust_train
   order by cust_id;

A. select cust_id, sum(train_code) as no_of_courses_taken
   from drone.cust_train
   group by cust_id
   order by cust_id;

[✓ A.] select cust_id, count(*) as no_of_courses_taken
   from drone.cust_train
   group by cust_id
   order by cust_id;

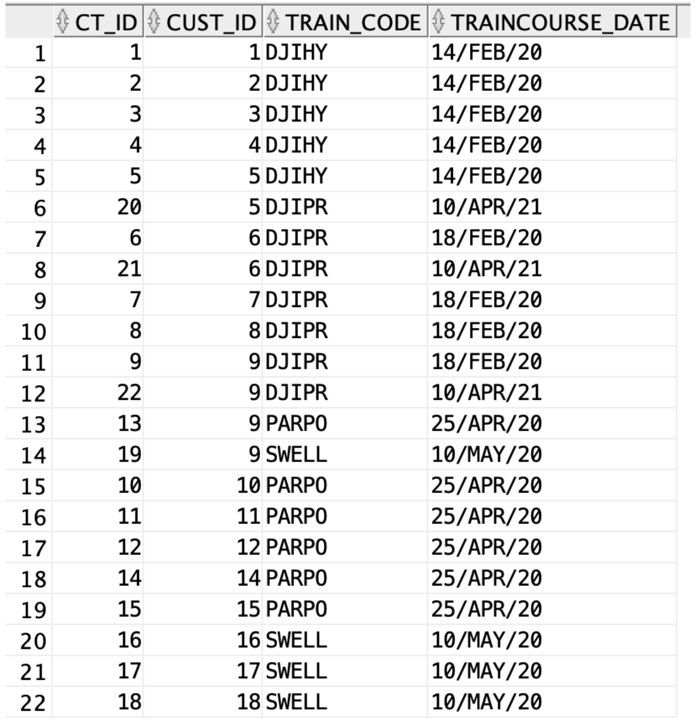
A. None of the above

What output is produced?

```
SELECT count(*)
FROM drone.cust_train;

SELECT cust_id, COUNT(*) AS no_courses_taken
FROM drone.cust_train
GROUP BY cust_id
ORDER BY cust_id;
```

```
SELECT AVG(COUNT(*))
       AS average_no_courses_taken
FROM drone.cust_train
GROUP BY cust_id;
```



	CT_ID	CUST_ID	TRAIN_CODE	TRAINCOURSE_DATE
1	1	1	DJIHY	14/FEB/20
2	2	2	DJIHY	14/FEB/20
3	3	3	DJIHY	14/FEB/20
4	4	4	DJIHY	14/FEB/20
5	5	5	DJIHY	14/FEB/20
6	20	5	DJIPR	10/APR/21
7	6	6	DJIPR	18/FEB/20
8	21	6	DJIPR	10/APR/21
9	7	7	DJIPR	18/FEB/20
10	8	8	DJIPR	18/FEB/20
11	9	9	DJIPR	18/FEB/20
12	22	9	DJIPR	10/APR/21
13	13	9	PARPO	25/APR/20
14	19	9	SWELL	10/MAY/20
15	10	10	PARPO	25/APR/20
16	11	11	PARPO	25/APR/20
17	12	12	PARPO	25/APR/20
18	14	14	PARPO	25/APR/20
19	15	15	PARPO	25/APR/20
20	16	16	SWELL	10/MAY/20
21	17	17	SWELL	10/MAY/20
22	18	18	SWELL	10/MAY/20

```
SQL> SELECT count(*)
  2  FROM drone.cust_train;

  COUNT(*)
----------
        22
```

```
SQL> SELECT cust_id, COUNT(*) AS
no_courses_taken
  2  FROM drone.cust_train
  3  GROUP BY cust_id
  4  ORDER BY cust_id;

   CUST_ID NO_COURSES_TAKEN
---------- ----------------
         1                1
         2                1
         3                1
         4                1
         5                2
         6                2
         7                1
         8                1
         9                4
        10                1
        11                1
        12                1
        14                1
        15                1
        16                1
        17                1
        18                1

17 rows selected.
```

```
SQL> SELECT AVG(COUNT(*))
  2      AS average_no_courses_taken
  3  FROM drone.cust_train
  4  GROUP BY cust_id;

AVERAGE_NO_COURSES_TAKEN
------------------------
              1.29411765
```

Q2. List all customer ids and the number of times each customer has taken a specific course:

A. select cust_id, train_code, count(*) as no_of_courses_taken
   from drone.cust_train
   group by cust_id
   order by cust_id;

[✓ A.] select cust_id, train_code, count(*) as no_of_courses_taken
   from drone.cust_train
   group by cust_id, train_code
   order by cust_id, train_code;

A. select cust_id, count(*) as no_of_courses_taken
   from drone.cust_train
   group by train_code
   order by train_code;

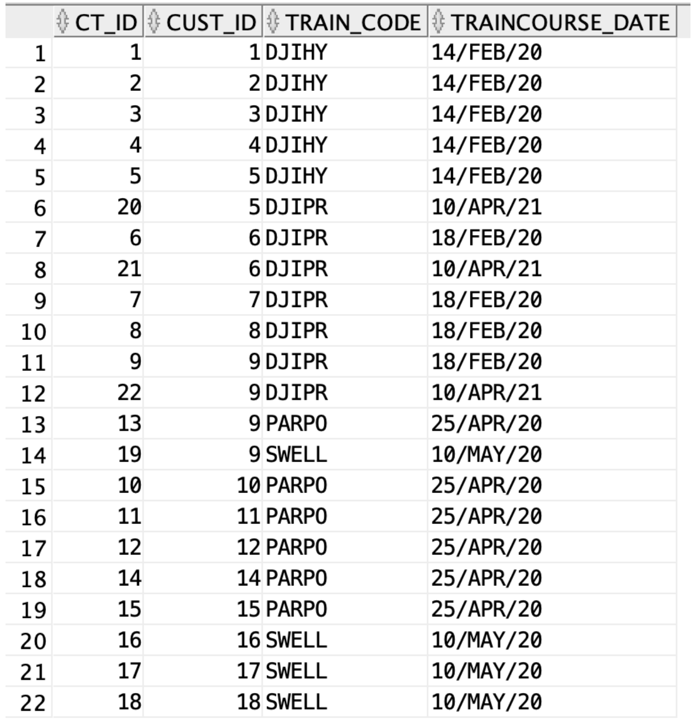
A. None of the above

IF THE ATTRIBUTES CONTAIN IN SELECT CLAUSE THEN IT MUST ALSO BE IN GROUP BY CLAUSE

MONASH University

What output is produced?

```
SELECT cust_id, train_code, count(train_code)
        as no_of_courses_taken
FROM drone.cust_train
GROUP BY cust_id, train_code
ORDER BY cust_id, train_code;
```



	CT_ID	CUST_ID	TRAIN_CODE	TRAINCOURSE_DATE
1	1	1	DJIHY	14/FEB/20
2	2	2	DJIHY	14/FEB/20
3	3	3	DJIHY	14/FEB/20
4	4	4	DJIHY	14/FEB/20
5	5	5	DJIHY	14/FEB/20
6	20	5	DJIPR	10/APR/21
7	6	6	DJIPR	18/FEB/20
8	21	6	DJIPR	10/APR/21
9	7	7	DJIPR	18/FEB/20
10	8	8	DJIPR	18/FEB/20
11	9	9	DJIPR	18/FEB/20
12	22	9	DJIPR	10/APR/21
13	13	9	PARPO	25/APR/20
14	19	9	SWELL	10/MAY/20
15	10	10	PARPO	25/APR/20
16	11	11	PARPO	25/APR/20
17	12	12	PARPO	25/APR/20
18	14	14	PARPO	25/APR/20
19	15	15	PARPO	25/APR/20
20	16	16	SWELL	10/MAY/20
21	17	17	SWELL	10/MAY/20
22	18	18	SWELL	10/MAY/20

```
SQL> SELECT cust_id, train_code, count(train_code) as no_of_courses_taken
  2  FROM drone.cust_train
  3  GROUP BY cust_id, train_code
  4  ORDER BY cust_id, train_code;

   CUST_ID TRAIN NO_OF_COURSES_TAKEN
---------- ----- -------------------
         1 DJIHY                   1
         2 DJIHY                   1
         3 DJIHY                   1
         4 DJIHY                   1
         5 DJIHY                   1
         5 DJIPR                   1
         6 DJIPR                   2
         7 DJIPR                   1
         8 DJIPR                   1
         9 DJIPR                   2
         9 PARPO                   1
         9 SWELL                   1
        10 PARPO                   1
        11 PARPO                   1
        12 PARPO                   1
        14 PARPO                   1
        15 PARPO                   1
        16 SWELL                   1
        17 SWELL                   1
        18 SWELL                   1

20 rows selected.
```

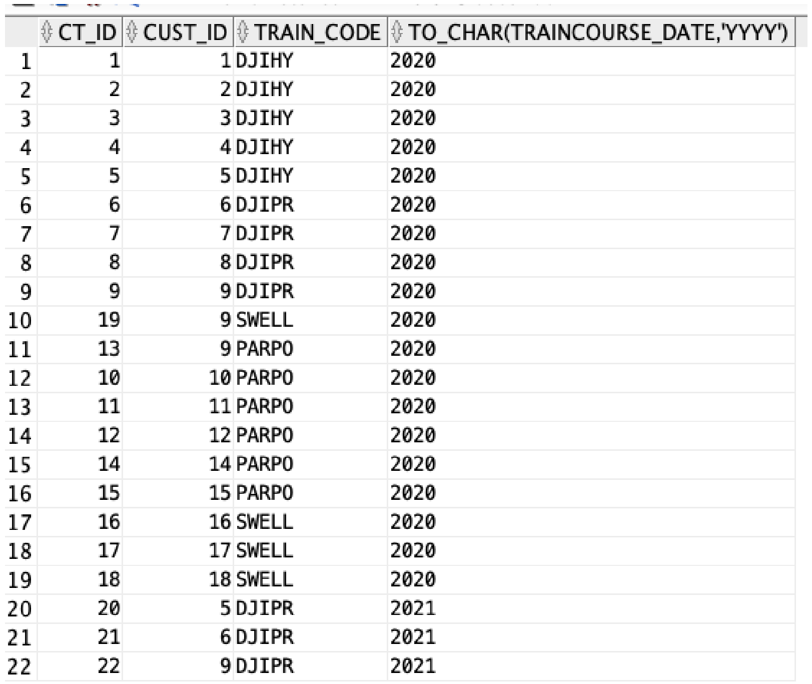
What output is produced?

```
SELECT cust_id,
       to_char(traincourse_date, 'yyyy') as year,
       count(train_code) as no_of_courses_taken
FROM drone.cust_train
GROUP BY cust_id, to_char(traincourse_date, 'yyyy')
ORDER BY cust_id, year;
```

Note: column alias cannot be used in group by clause

WHY?

```
cust_id, year, no_of_course_taken
9          2020     3
9          2021     1
```



	CT_ID	CUST_ID	TRAIN_CODE	TO_CHAR(TRAINCOURSE_DATE,'YYYY')
1	1	1	DJIHY	2020
2	2	2	DJIHY	2020
3	3	3	DJIHY	2020
4	4	4	DJIHY	2020
5	5	5	DJIHY	2020
6	6	6	DJIPR	2020
7	7	7	DJIPR	2020
8	8	8	DJIPR	2020
9	9	9	DJIPR	2020
10	19	9	SWELL	2020
11	13	9	PARPO	2020
12	10	10	PARPO	2020
13	11	11	PARPO	2020
14	12	12	PARPO	2020
15	14	14	PARPO	2020
16	15	15	PARPO	2020
17	16	16	SWELL	2020
18	17	17	SWELL	2020
19	18	18	SWELL	2020
20	20	5	DJIPR	2021
21	21	6	DJIPR	2021
22	22	9	DJIPR	2021

```
SQL> SELECT cust_id, to_char(traincourse_date, 'yyyy') as year, count(train_code) as no_of_courses_taken
  2  FROM drone.cust_train
  3  GROUP BY cust_id, to_char(traincourse_date, 'yyyy')
  4  ORDER BY cust_id, year;

   CUST_ID YEAR NO_OF_COURSES_TAKEN
---------- ---- -------------------
         1 2020                   1
         2 2020                   1
         3 2020                   1
         4 2020                   1
         5 2020                   1
         5 2021                   1
         6 2020                   1
         6 2021                   1
         7 2020                   1
         8 2020                   1
         9 2020                   3
         9 2021                   1
        10 2020                   1
        11 2020                   1
        12 2020                   1
        14 2020                   1
        15 2020                   1
        16 2020                   1
        17 2020                   1
        18 2020                   1

20 rows selected.
```

Q3. Which rows that will be returned by this select statement:

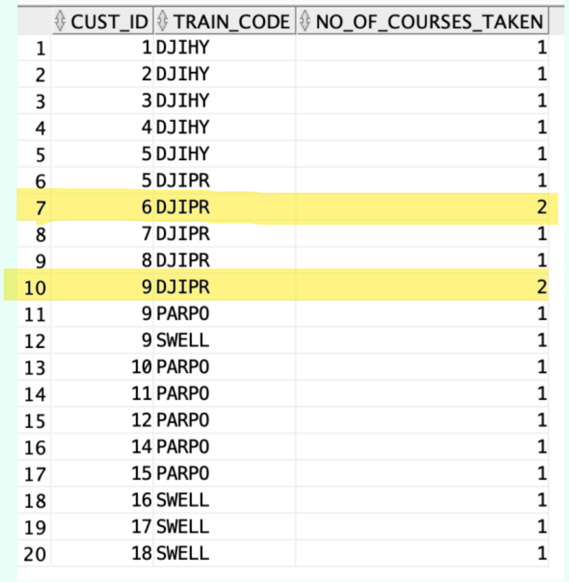
```
SELECT cust_id, train_code, count(train_code)
    as no_of_courses_taken
FROM drone.cust_train
GROUP BY cust_id, train_code
HAVING count(train_code) > 1
ORDER BY cust_id, train_code;
```

A. all rows

[✓ B.] 7, 10

C. none of them

D. all rows except row 7 and 10



	CUST_ID	TRAIN_CODE	NO_OF_COURSES_TAKEN
1	1	DJIHY	1
2	2	DJIHY	1
3	3	DJIHY	1
4	4	DJIHY	1
5	5	DJIHY	1
6	5	DJIPR	1
7	6	DJIPR	2
8	7	DJIPR	1
9	8	DJIPR	1
10	9	DJIPR	2
11	9	PARPO	1
12	9	SWELL	1
13	10	PARPO	1
14	11	PARPO	1
15	12	PARPO	1
16	14	PARPO	1
17	15	PARPO	1
18	16	SWELL	1
19	17	SWELL	1
20	18	SWELL	1

HAVING clause

- It is used to put a condition or conditions on the groups defined by GROUP BY clause.

```
SELECT cust_id, train_code, count(train_code)
    as no_of_courses_taken
FROM drone.cust_train
GROUP BY cust_id, train_code
HAVING count(train_code) > 1
ORDER BY cust_id, train_code;
```

What output is produced?

```
SELECT cust_id, train_code, count(train_code) as no_of_courses_taken
FROM drone.cust_train
GROUP BY cust_id, train_code
HAVING count(train_code) > 1
ORDER BY cust_id, train_code;
```

```
SELECT dt_code, AVG(drone_flight_time) as average_drone_flight
FROM drone.drone
GROUP BY dt_code
HAVING AVG(drone_flight_time)>50
ORDER BY dt_code;
```

```
SQL> SELECT cust_id, train_code, count(train_code) as no_of_courses_taken
  2  FROM drone.cust_train
  3  GROUP BY cust_id, train_code
  4  HAVING count(train_code) > 1
  5  ORDER BY cust_id, train_code;

   CUST_ID TRAIN NO_OF_COURSES_TAKEN
---------- ----- -------------------
         6 DJIPR                   2
         9 DJIPR                   2

SQL> SELECT dt_code, AVG(drone_flight_time) as average_drone_flight
  2  FROM drone.drone
  3  GROUP BY dt_code
  4  HAVING AVG(drone_flight_time)>50
  5  ORDER BY dt_code;

DT_C AVERAGE_DRONE_FLIGHT
---- --------------------
DIN2           78.6666667
DMA2           53.3333333
PAPR               97.625
SWPS                 56.3
```

MONASH University

HAVING and WHERE clauses

```
SELECT dt_code, AVG(drone_flight_time) as average_drone_flight
FROM drone.drone
WHERE to_char(drone_pur_date,'yyyy') = '2020'
GROUP BY dt_code
HAVING AVG(drone_flight_time)>50
ORDER BY dt_code;
```

having is only applicable to the result of groupby

- The WHERE clause is applied to ALL rows in the table.
- The HAVING clause is applied to the groups defined by the GROUP BY clause.
- The order of operations performed is FROM, WHERE, GROUP BY, HAVING and then ORDER BY.
- On the above example, the logic of the process will be:
  - All rows where drone purchase year = 2020 are retrieved. (due to the WHERE clause)
  - The retrieved rows then are grouped into different dt_code.
  - If the average flight time in a group is greater than 50, the dt_code and the average flight time is displayed. (due to the HAVING clause)

MONASH University

```
SQL> SELECT
  2      dt_code,
  3      AVG(drone_flight_time) AS average_drone_flight
  4  FROM
  5      drone.drone
  6  WHERE
  7      to_char(drone_pur_date, 'yyyy') = '2020'
  8  GROUP BY
  9      dt_code
 10  HAVING
 11      AVG(drone_flight_time) > 50
 12  ORDER BY
 13      average_drone_flight desc;

DT_C AVERAGE_DRONE_FLIGHT
---- --------------------
DIN2                  200
PAPR               97.625
DMA2                   80
SWPS                 56.3
```

```
SELECT cust_id, train_code, count(*) as no_of_courses_taken
FROM drone.cust_train
GROUP BY cust_id
ORDER BY cust_id;
```

whatever in select clause except aggregate function must be in group by part

The above SQL generates error message

```
SQL Error: ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
```

Why and how to fix this?

- Why? Because the grouping is based on the cust_id, whereas the display is based on cust_id and train_code. The two groups may not have the same members.
- How to fix this?
  - Include the train_code as part of the GROUP BY condition.
- Attributes that are used in the SELECT, HAVING and ORDER BY must be included in the GROUP BY clause (reverse is not necessary).

Subqueries

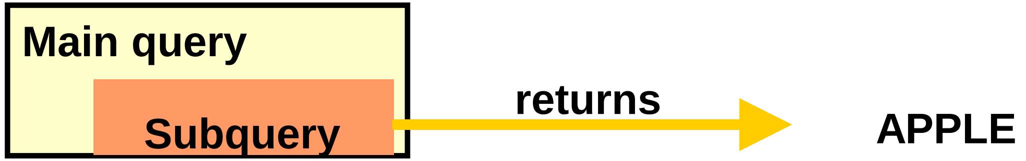
Query within a query.

"Find all drones which flight time is higher than the average flight time of all drones"

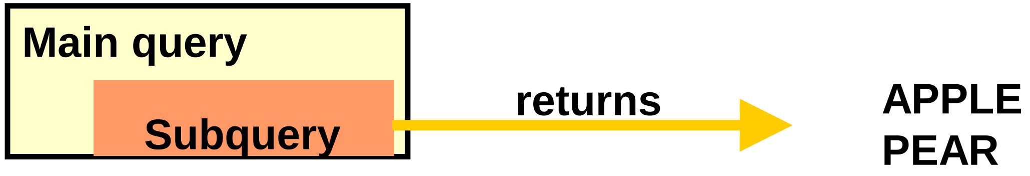
```
SELECT *
FROM drone.drone
WHERE drone_flight_time >
   (
      SELECT AVG(drone_flight_time)
      FROM drone.drone
   )
ORDER BY drone_id;
```

Types of Subqueries

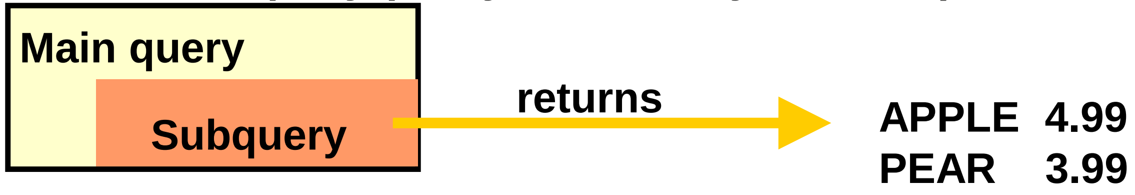
Single-value



Multiple-row subquery (a list of values – many rows, one column)



Multiple-column subquery (many rows, many columns)



Q5. What will be returned by the *inner query*?

```
SELECT *
FROM drone.drone
WHERE drone_pur_price > (SELECT AVG(drone_pur_price)
                         FROM drone.drone
                         GROUP BY drone_pur_date)
ORDER BY drone_id;
```

A. A value (a single column, single row).
[✓ B.] A list of values.
C. Multiple columns, multiple rows.
D. None of the above.

MONASH University

```
SQL> SELECT
  2      *
  3  FROM
  4      drone.drone
  5  WHERE drone_pur_price > (SELECT AVG(drone_pur_price)
  6                           FROM drone.drone
  7                           GROUP BY drone_pur_date);

Error starting at line : 1 in command -
SELECT
    *
FROM
    drone.drone
WHERE drone_pur_price > (SELECT AVG(drone_pur_price)
                         FROM drone.drone
                         GROUP BY drone_pur_date)
Error report -
ORA-01427: single-row subquery returns more than one row
```

Q6. What will be returned by the *inner query*?

```
SELECT dt_code,dt_model,drone_id, drone_pur_price
FROM drone.drone_type natural join drone.drone
WHERE (dt_code, drone_pur_price) IN
                                        (SELECT dt_code, MAX(drone_pur_price)
                    FROM drone.drone_type NATURAL JOIN drone.drone
                    GROUP BY dt_code)
ORDER BY dt_code;
```

A. A value (a single column, single row).

B. A list of values.

[✓ C.] Multiple columns, multiple rows.

D. None of the above.

MONASH University

Comparison Operators for Subquery

- Operator for single value comparison.

  =, <, >

- Operator for multiple rows or a list comparison.
  - –equality
    - IN
  - –inequality
    - ALL, ANY combined with <, >

Q7. Write the SQL Query to find the details of all drones which have a purchase price less than the average purchase price for all drones manufactured by DJI Da-Jiang Innovations.

Begin by your group listing the steps which need to be taken

After this code the SQL step by step.

Your output must show the drone id, the type code, the purchase price, the year purchased and the manufacturers name.

Order the output by drone id.

```
SELECT
    drone_id,
    dt_code,
    drone_pur_price,
    to_char(drone_pur_date,'yyyy') as yearpurchased,
    manuf_name
FROM
         drone.drone
    NATURAL JOIN drone.drone_type
    NATURAL JOIN drone.manufacturer
WHERE
    drone_pur_price < (
        SELECT
            AVG(drone_pur_price)
        FROM
                 drone.drone
            NATURAL JOIN drone.drone_type
            NATURAL JOIN drone.manufacturer
        WHERE
            upper(manuf_name) = 'DJI DA-JIANG INNOVATIONS'
    )
ORDER BY
    drone_id;
```

What you have learnt

- Aggregate Functions
  - count, min, max, avg, sum
- GROUP BY and HAVING clauses.
- Subquery
  - Inner vs outer query
  - comparison operators (IN, ANY, ALL)

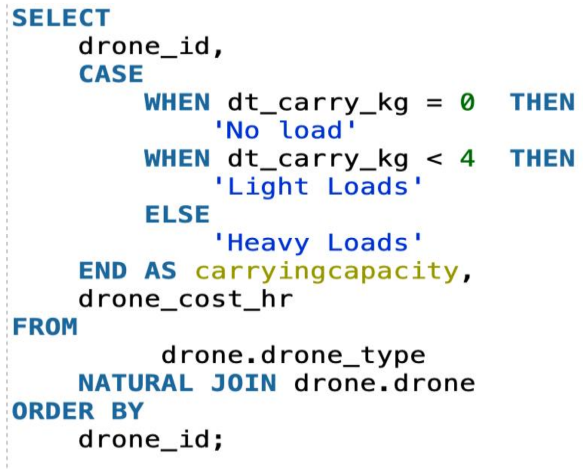
What’s more

- CASE
- Subquery – nested, inline, correlated
- Views
- Joins - self join, outer join
- Set Operators
- Oracle Functions

SQL CASE statement

The CASE statement used in the select list enables a query to evaluate an attribute and output a particular value based on that evaluation.

Drones which can carry objects have been classified by HyFlying as light carriers for carrying less than 4 Kg, heavy carriers for 4 Kg and greater. Display all drones and their carrying capacity classification as either ‘No load’, ‘Light Loads’ or ‘Heavy Loads’ :



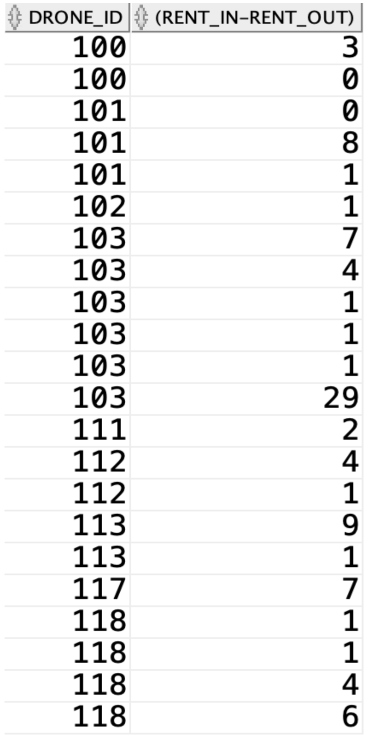
```
SELECT
    drone_id,
    CASE
        WHEN dt_carry_kg = 0  THEN
            'No load'
        WHEN dt_carry_kg < 4  THEN
            'Light Loads'
        ELSE
            'Heavy Loads'
    END AS carryingcapacity,
    drone_cost_hr
FROM
         drone.drone_type
    NATURAL JOIN drone.drone
ORDER BY
    drone_id;
```

Query

For each drone, find the customers (cust_id only) who rented the drone for the highest number of days. Include the drone id, number of rented days and customer id in the output.

For each completed rental list the number of days the drone is out

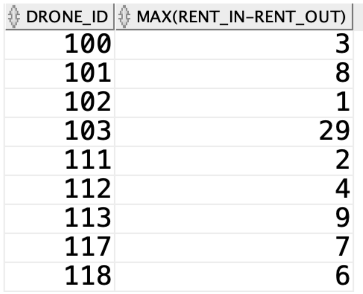
```
SELECT
         drone_id,
         ( rent_in -
rent_out )
FROM
         drone.rental
WHERE
         rent_in IS NOT
NULL
ORDER BY
         drone_id;
```



DRONE_ID	(RENT_IN-RENT_OUT)
100	3
100	0
101	0
101	8
101	1
102	1
103	7
103	4
103	1
103	1
103	1
103	29
111	2
112	4
112	1
113	9
113	1
117	7
118	1
118	1
118	4
118	6

For a given drone list the maximum number of days the drone was out

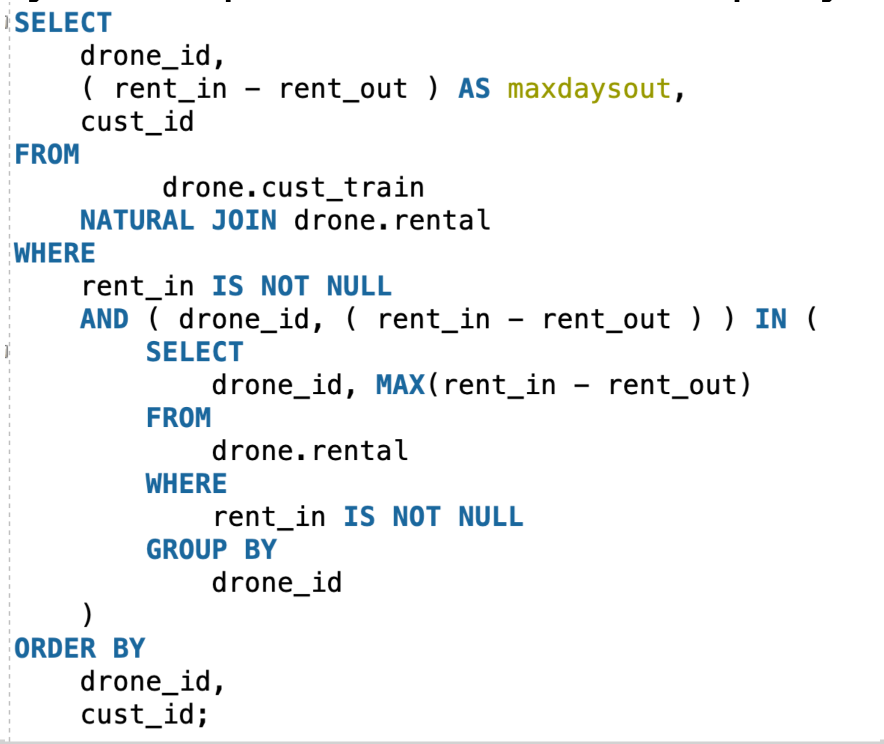
```
SELECT
         drone_id,
         MAX(rent_in - rent_out)
FROM
         drone.rental
WHERE
         rent_in IS NOT NULL
GROUP BY
         drone_id
ORDER BY
         drone_id;
```



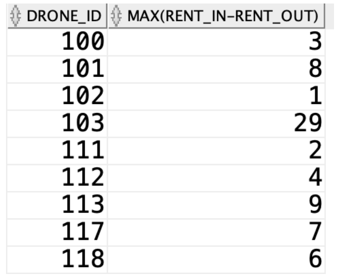
DRONE_ID	MAX(RENT_IN-RENT_OUT)
100	3
101	8
102	1
103	29
111	2
112	4
113	9
117	7
118	6

Subquery (NESTED)

- The subquery is independent of the outer query and is executed only once.



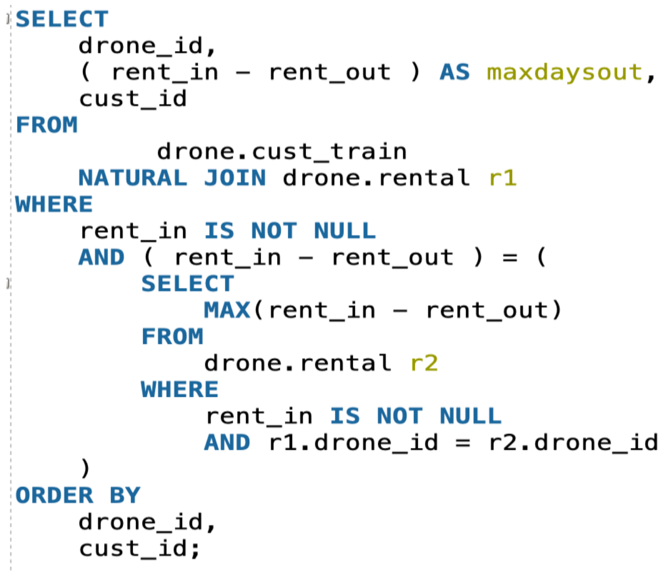
```
SELECT
    drone_id,
    ( rent_in - rent_out ) AS maxdaysout,
    cust_id
FROM
         drone.cust_train
    NATURAL JOIN drone.rental
WHERE
    rent_in IS NOT NULL
    AND ( drone_id, ( rent_in - rent_out ) ) IN (
        SELECT
            drone_id, MAX(rent_in - rent_out)
        FROM
            drone.rental
        WHERE
            rent_in IS NOT NULL
        GROUP BY
            drone_id
    )
ORDER BY
    drone_id,
    cust_id;
```



DRONE_ID	MAX(RENT_IN-RENT_OUT)
100	3
101	8
102	1
103	29
111	2
112	4
113	9
117	7
118	6

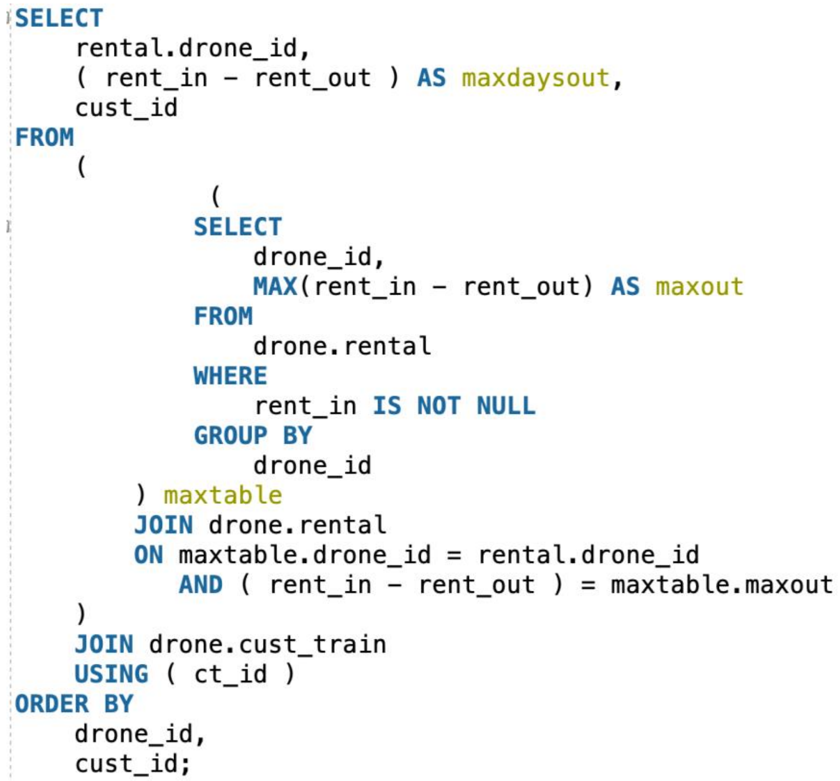
Subquery (CORRELATED)

- the subquery is related to the outer query and is ***evaluated once for each row of the outer query***
- correlated subqueries can also be used within update statements
  - outer update occurs based on value returned from subquery

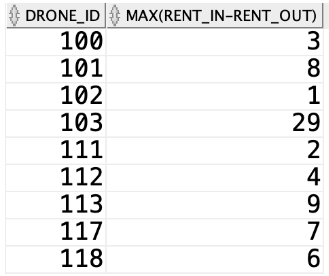


```
SELECT
    drone_id,
    ( rent_in - rent_out ) AS maxdaysout,
    cust_id
FROM
         drone.cust_train
    NATURAL JOIN drone.rental r1
WHERE
    rent_in IS NOT NULL
    AND ( rent_in - rent_out ) = (
        SELECT
            MAX(rent_in - rent_out)
        FROM
            drone.rental r2
        WHERE
            rent_in IS NOT NULL
            AND r1.drone_id = r2.drone_id
    )
ORDER BY
    drone_id,
    cust_id;
```

Subquery (INLINE) – Derived table



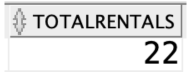
```
SELECT
    rental.drone_id,
    ( rent_in - rent_out ) AS maxdaysout,
    cust_id
FROM
    (
             (
            SELECT
                drone_id,
                MAX(rent_in - rent_out) AS maxout
            FROM
                drone.rental
            WHERE
                rent_in IS NOT NULL
            GROUP BY
                drone_id
        ) maxtable
        JOIN drone.rental
        ON maxtable.drone_id = rental.drone_id
           AND ( rent_in - rent_out ) = maxtable.maxout
    )
    JOIN drone.cust_train
    USING ( ct_id )
ORDER BY
    drone_id,
    cust_id;
```



DRONE_ID	MAX(RENT_IN-RENT_OUT)
100	3
101	8
102	1
103	29
111	2
112	4
113	9
117	7
118	6

How many completed rentals have been recorded?

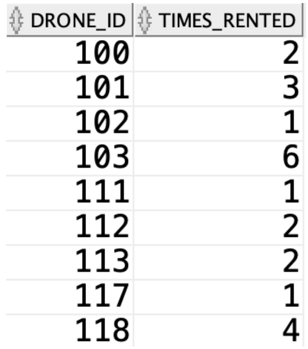
```
SELECT
        COUNT(*) AS
totalrentals
FROM
        drone.rental
WHERE
        rent_in IS NOT NULL;
```



TOTALRENTALS
22

List for each drone the number of times the drone has been rented in a completed rental

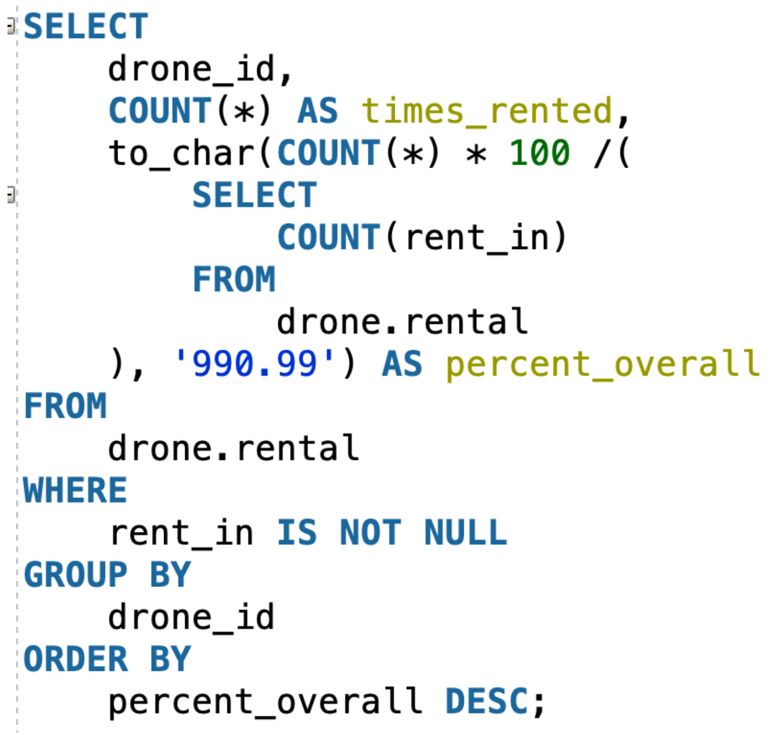
```
SELECT
        drone_id,
        COUNT(*) AS
times_rented
FROM
        drone.rental
WHERE
        rent_in IS NOT
GROUP BY
        drone_id
ORDER BY
        drone_id;
```



DRONE_ID	TIMES_RENTED
100	2
101	3
102	1
103	6
111	1
112	2
113	2
117	1
118	4

For each drone compute the percentage of the company's rentals contributed by that drone

Subquery (INLINE)



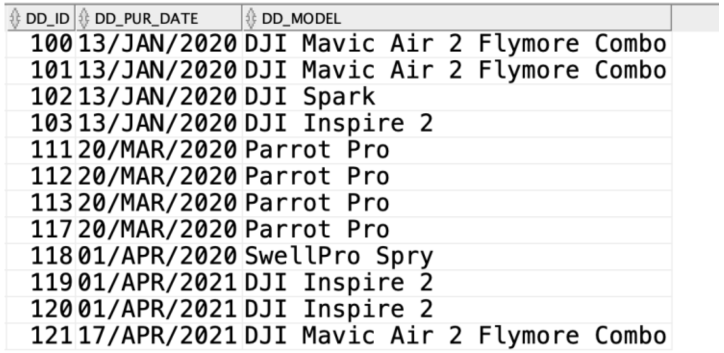
```sql
SELECT
    drone_id,
    COUNT(*) AS times_rented,
    to_char(COUNT(*) * 100 /(
        SELECT
            COUNT(rent_in)
        FROM
            drone.rental
    ), '990.99') AS percent_overall
FROM
    drone.rental
WHERE
    rent_in IS NOT NULL
GROUP BY
    drone_id
ORDER BY
    percent_overall DESC;
```

Use of subquery in INSERT

```
CREATE TABLE drone_details (
    dd_id         NUMBER(5) NOT NULL,
    dd_pur_date   DATE NOT NULL,
    dd_model      VARCHAR2(50) NOT NULL,
    CONSTRAINT drone_details_pk PRIMARY KEY ( dd_id )
);
```

Assume table created

```
INSERT INTO drone_details
    ( SELECT
        drone_id,
        drone_pur_date,
        dt_model
    FROM
             drone.drone
        NATURAL JOIN drone.drone_type
    );
```

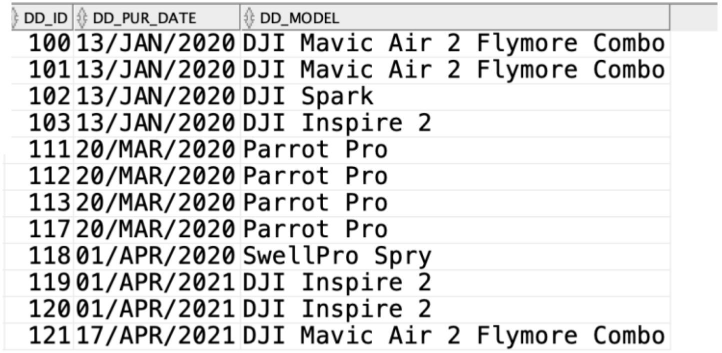


DD_ID	DD_PUR_DATE	DD_MODEL
100	13/JAN/2020	DJI Mavic Air 2 Flymore Combo
101	13/JAN/2020	DJI Mavic Air 2 Flymore Combo
102	13/JAN/2020	DJI Spark
103	13/JAN/2020	DJI Inspire 2
111	20/MAR/2020	Parrot Pro
112	20/MAR/2020	Parrot Pro
113	20/MAR/2020	Parrot Pro
117	20/MAR/2020	Parrot Pro
118	01/APR/2020	SwellPro Spry
119	01/APR/2021	DJI Inspire 2
120	01/APR/2021	DJI Inspire 2
121	17/APR/2021	DJI Mavic Air 2 Flymore Combo

If you need to both create and insert the data, is there a simpler way to achieve these two tasks?

Simpler approach (using week 7 applied class approach 7.3.4)

```
CREATE TABLE drone_details
    AS
        ( SELECT
            drone_id,
            drone_pur_date,
            dt_model
        FROM
                drone.drone
            NATURAL JOIN drone.drone_type
        );
```



DD_ID	DD_PUR_DATE	DD_MODEL
100	13/JAN/2020	DJI Mavic Air 2 Flymore Combo
101	13/JAN/2020	DJI Mavic Air 2 Flymore Combo
102	13/JAN/2020	DJI Spark
103	13/JAN/2020	DJI Inspire 2
111	20/MAR/2020	Parrot Pro
112	20/MAR/2020	Parrot Pro
113	20/MAR/2020	Parrot Pro
117	20/MAR/2020	Parrot Pro
118	01/APR/2020	SwellPro Spry
119	01/APR/2021	DJI Inspire 2
120	01/APR/2021	DJI Inspire 2
121	17/APR/2021	DJI Mavic Air 2 Flymore Combo

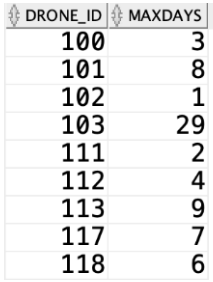
Views

- A virtual table derived from one or more base tables.
- Sometimes used as "Access Control" to the database

CREATE OR REPLACE VIEW [view_name] AS

SELECT … ;

```
CREATE OR REPLACE VIEW maxdaysout_view AS
    SELECT
        drone_id,
        MAX(rent_in - rent_out) AS maxdays
    FROM
        drone.rental
    WHERE
        rent_in IS NOT NULL
    GROUP BY
        drone_id;
```



DRONE_ID	MAXDAYS
100	3
101	8
102	1
103	29
111	2
112	4
113	9
117	7
118	6

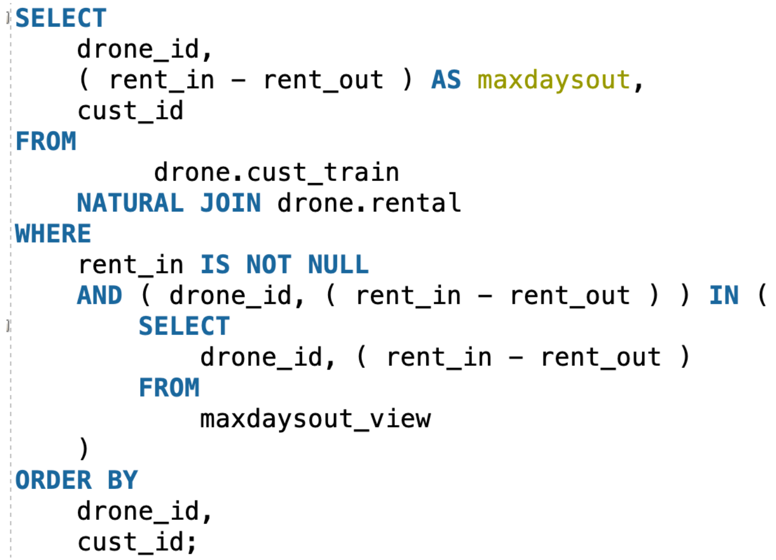
select * from maxdaysout_view
order by drone_id;

- What objects do I own?

```
select * from user_objects;
```

Using Views

- For each drone find the customers (cust_id only) who rented the drone for the highest number of days



```
SELECT
    drone_id,
    ( rent_in - rent_out ) AS maxdaysout,
    cust_id
FROM
         drone.cust_train
    NATURAL JOIN drone.rental
WHERE
    rent_in IS NOT NULL
    AND ( drone_id, ( rent_in - rent_out ) ) IN (
        SELECT
            drone_id, ( rent_in - rent_out )
        FROM
            maxdaysout_view
    )
ORDER BY
    drone_id,
    cust_id;
```

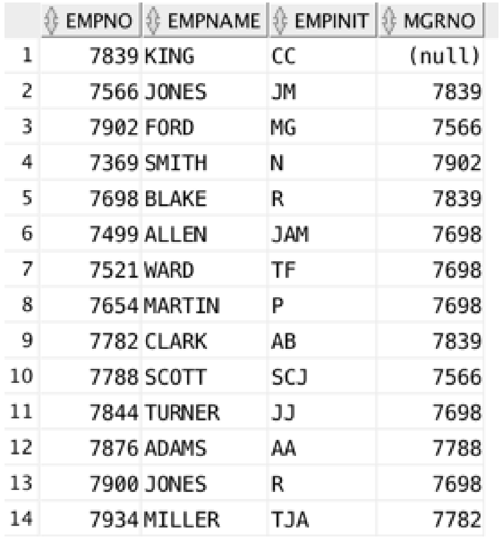
Please note VIEWS MUST NOT be used for Assignment 2 or Exam

45

Self Join

- Show the name of the manager for each employee.

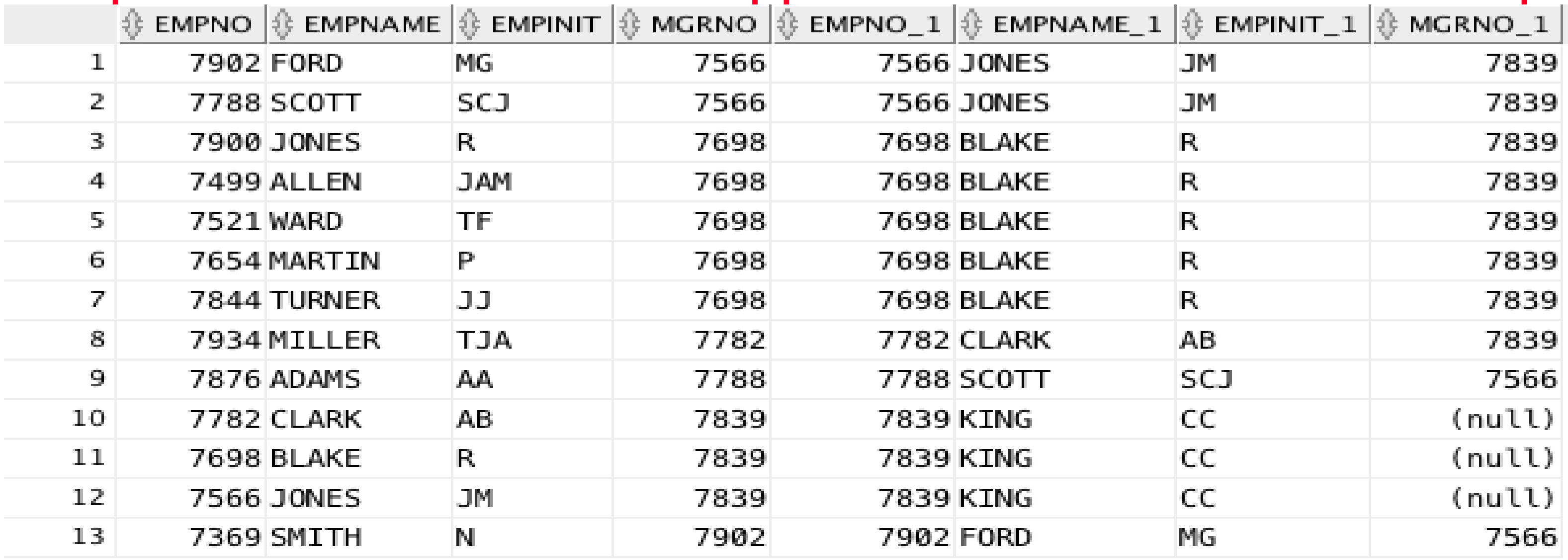
```
SELECT
    empno,
    empname,
    empinit,
    mgrno
FROM
    emp.employee;
```



	EMPNO	EMPNAME	EMPINIT	MGRNO
1	7839	KING	CC	(null)
2	7566	JONES	JM	7839
3	7902	FORD	MG	7566
4	7369	SMITH	N	7902
5	7698	BLAKE	R	7839
6	7499	ALLEN	JAM	7698
7	7521	WARD	TF	7698
8	7654	MARTIN	P	7698
9	7782	CLARK	AB	7839
10	7788	SCOTT	SCJ	7566
11	7844	TURNER	JJ	7698
12	7876	ADAMS	AA	7788
13	7900	JONES	R	7698
14	7934	MILLER	TJA	7782

```
SELECT *
FROM emp.employee e1 JOIN emp.employee e2
ON e1.mgrno = e2.empno;
```

e1 e2



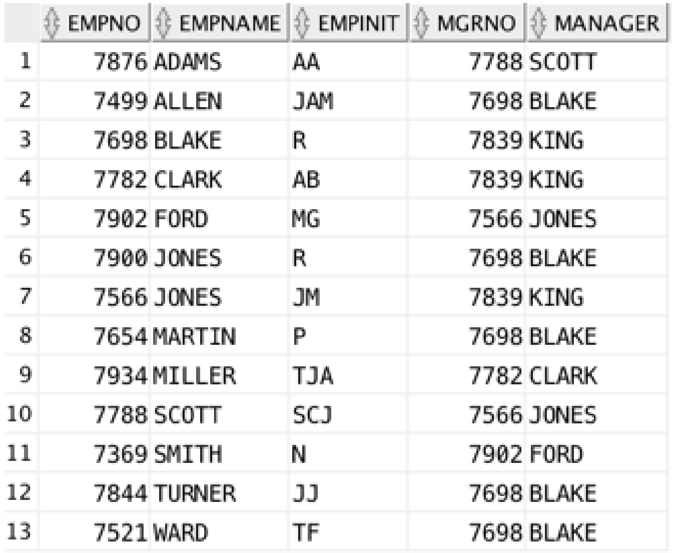
	EMPNO	EMPNAME	EMPINIT	MGRNO	EMPNO_1	EMPNAME_1	EMPINIT_1	MGRNO_1
1	7902	FORD	MG	7566	7566	JONES	JM	7839
2	7788	SCOTT	SCJ	7566	7566	JONES	JM	7839
3	7900	JONES	R	7698	7698	BLAKE	R	7839
4	7499	ALLEN	JAM	7698	7698	BLAKE	R	7839
5	7521	WARD	TF	7698	7698	BLAKE	R	7839
6	7654	MARTIN	P	7698	7698	BLAKE	R	7839
7	7844	TURNER	JJ	7698	7698	BLAKE	R	7839
8	7934	MILLER	TJA	7782	7782	CLARK	AB	7839
9	7876	ADAMS	AA	7788	7788	SCOTT	SCJ	7566
10	7782	CLARK	AB	7839	7839	KING	CC	(null)
11	7698	BLAKE	R	7839	7839	KING	CC	(null)
12	7566	JONES	JM	7839	7839	KING	CC	(null)
13	7369	SMITH	N	7902	7902	FORD	MG	7566

Joined rows
1,12
2,12
3,11

Note some columns have been hidden

Why now only 13 rows?

```
SELECT e1.empno, e1.empname, e1.empinit, e1.mgrno,
           e2.empname AS MANAGER
FROM emp.employee e1  JOIN emp.employee e2
           ON e1.mgrno = e2.empno
ORDER BY e1.empname;
```

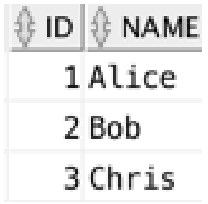


	EMPNO	EMPNAME	EMPINIT	MGRNO	MANAGER
1	7876	ADAMS	AA	7788	SCOTT
2	7499	ALLEN	JAM	7698	BLAKE
3	7698	BLAKE	R	7839	KING
4	7782	CLARK	AB	7839	KING
5	7902	FORD	MG	7566	JONES
6	7900	JONES	R	7698	BLAKE
7	7566	JONES	JM	7839	KING
8	7654	MARTIN	P	7698	BLAKE
9	7934	MILLER	TJA	7782	CLARK
10	7788	SCOTT	SCJ	7566	JONES
11	7369	SMITH	N	7902	FORD
12	7844	TURNER	JJ	7698	BLAKE
13	7521	WARD	TF	7698	BLAKE

MONASH University

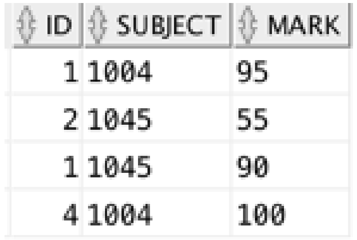
INNER JOIN

Student



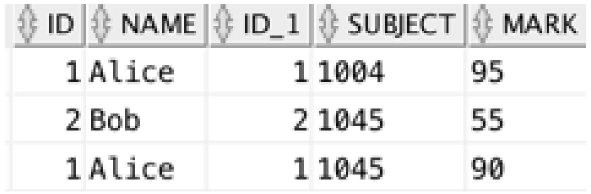
ID	NAME
1	Alice
2	Bob
3	Chris

Mark



ID	SUBJECT	MARK
1	1004	95
2	1045	55
1	1045	90
4	1004	100

Inner Join gives no information for Chris and the student with ID 4

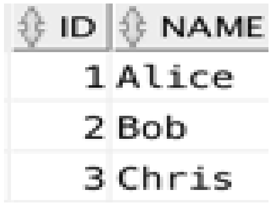


ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1004	95
2	Bob	2	1045	55
1	Alice	1	1045	90

Select * from student s join mark m on s.id = m.id;
***Note that this is an EQUI JOIN* (an inner join)**

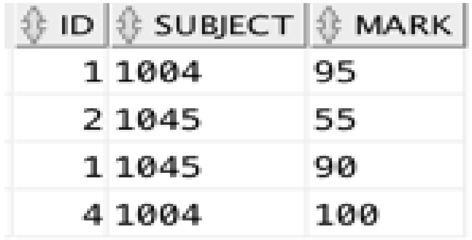
FULL OUTER JOIN

Student



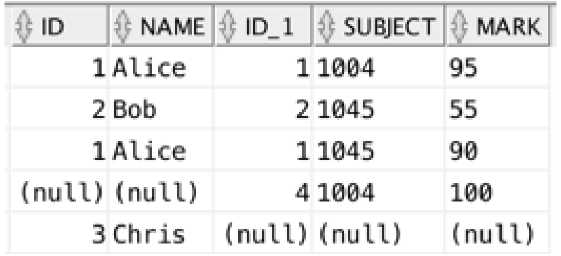
ID	NAME
1	Alice
2	Bob
3	Chris

Mark



ID	SUBJECT	MARK
1	1004	95
2	1045	55
1	1045	90
4	1004	100

Get (incomplete) information of both Chris and student with ID 4

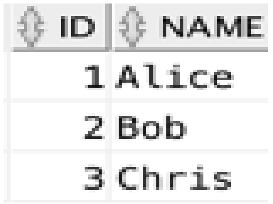


ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1004	95
2	Bob	2	1045	55
1	Alice	1	1045	90
(null)	(null)	4	1004	100
3	Chris	(null)	(null)	(null)

```
select * from
student s full outer join mark m on s.id = m.id;
```

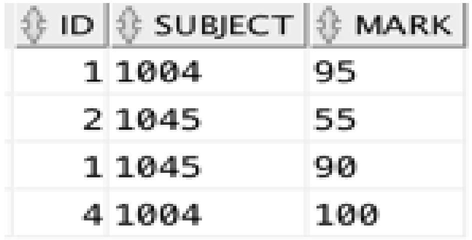
LEFT OUTER JOIN

Student



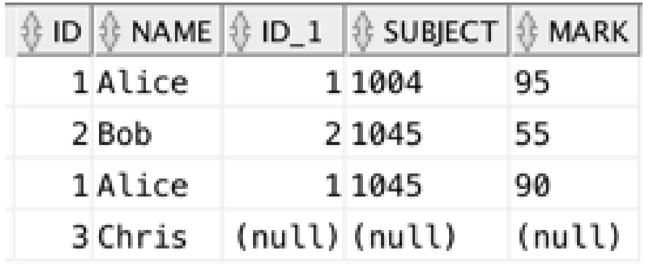
ID	NAME
1	Alice
2	Bob
3	Chris

Mark



ID	SUBJECT	MARK
1	1004	95
2	1045	55
1	1045	90
4	1004	100

Get (incomplete) information of only Chris

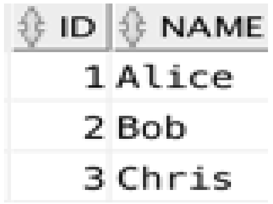


ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1004	95
2	Bob	2	1045	55
1	Alice	1	1045	90
3	Chris	(null)	(null)	(null)

```
select * from
student s left outer join mark m
on s.id = m.id;
```

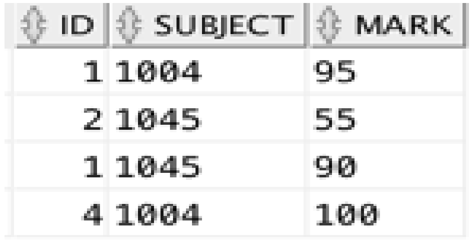
RIGHT OUTER JOIN

Student



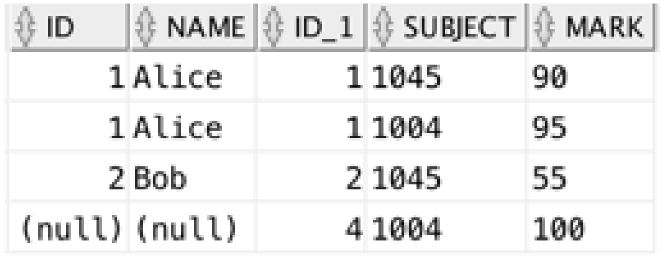
ID	NAME
1	Alice
2	Bob
3	Chris

Mark



ID	SUBJECT	MARK
1	1004	95
2	1045	55
1	1045	90
4	1004	100

Get (incomplete) information of the student with ID 4



ID	NAME	ID_1	SUBJECT	MARK
1	Alice	1	1045	90
1	Alice	1	1004	95
2	Bob	2	1045	55
(null)	(null)	4	1004	100

select * from
student s right outer join mark m
on s.id = m.id;

Outer Join

- List the number of times ALL drones have been rented

```sql
SELECT
    drone_id,
    COUNT(rent_out) as timerented
FROM
        drone.drone
    JOIN drone.rental
    USING ( drone_id )
GROUP BY
    drone_id
ORDER BY
    drone_id;
```

	DRONE_ID	TIMERENTED
1	100	2
2	101	3
3	102	1
4	103	6
5	111	1
6	112	2
7	113	2
8	117	1
9	118	5
10	119	1
11	120	1

```sql
SELECT
    drone_id,
    COUNT(rent_out) as timesrented
FROM
    drone.drone
    LEFT OUTER JOIN drone.rental
    USING ( drone_id )
GROUP BY
    drone_id
ORDER BY
    drone_id;
```

	DRONE_ID	TIMESRENTED
1	100	2
2	101	3
3	102	1
4	103	6
5	111	1
6	112	2
7	113	2
8	117	1
9	118	5
10	119	1
11	120	1
12	121	0

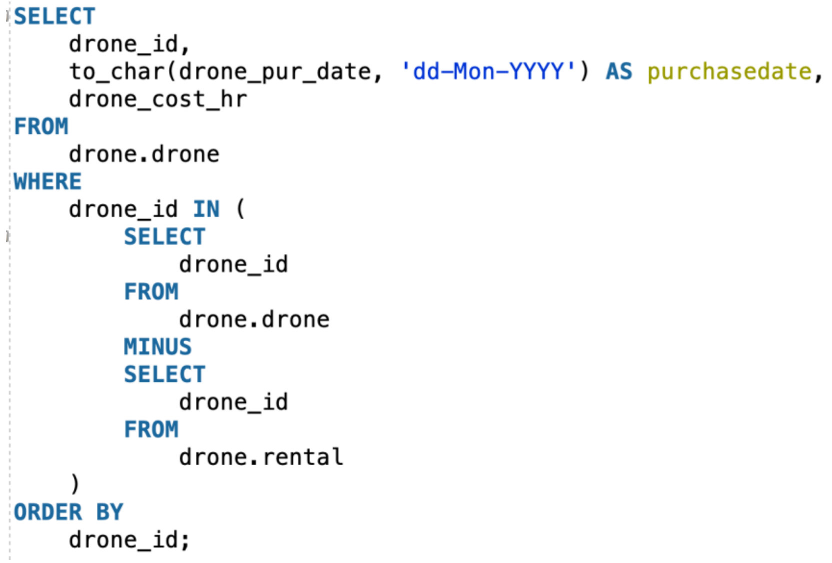
Relational Set Operators

- Using the set operators you can combine two or more sets to create new sets (relations)
- Union All
  - All rows selected by either query, including all duplicates
- Union
  - All rows selected by either query, removing duplicates (eg. DISTINCT on Union All)
- Intersect
  - All distinct rows selected by both queries
- Minus
  - All distinct rows selected by the first query but not by the second
- All set operators have equal precedence. If a SQL statement contains multiple set operators, Oracle evaluates them from the left to right if no parentheses explicitly specify another order.
- The two sets must be UNION COMPATIBLE (ie. same number of attributes and similar data types)

MINUS

List the details of drones which have not been rented. Include drone id, drone purchase date and drone cost per hour in the list.

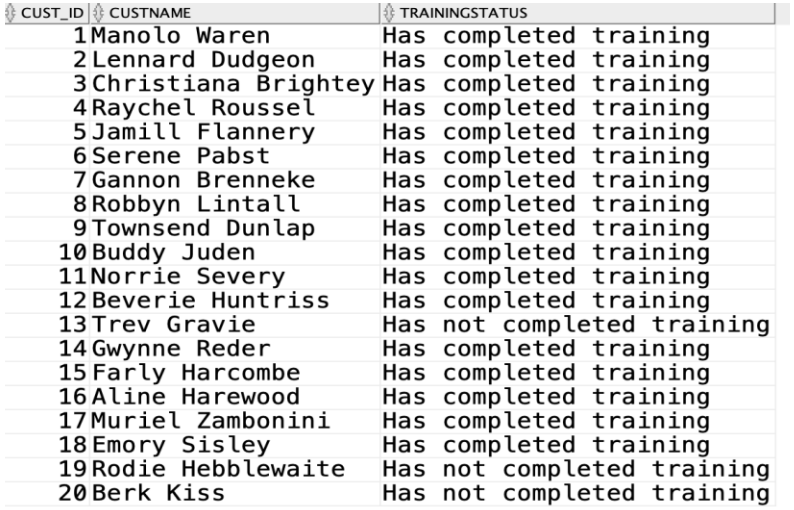
- List the drone id of all drones
- List the drone id of those drones which have been rented



```sql
SELECT
    drone_id,
    to_char(drone_pur_date, 'dd-Mon-YYYY') AS purchasedate,
    drone_cost_hr
FROM
    drone.drone
WHERE
    drone_id IN (
        SELECT
            drone_id
        FROM
            drone.drone
        MINUS
        SELECT
            drone_id
        FROM
            drone.rental
    )
ORDER BY
    drone_id;
```

UNION

- Create a list of all customers:
  - for those who have completed training show "Completed training"
  - for those who have not completed training show "Not completed training"

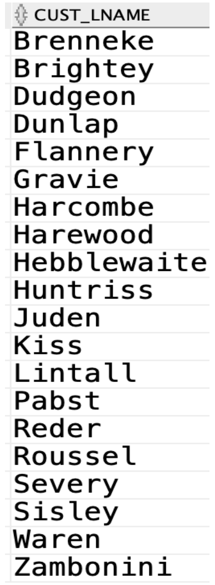


CUST_ID	CUSTNAME	TRAININGSTATUS
1	Manolo Waren	Has completed training
2	Lennard Dudgeon	Has completed training
3	Christiana Brightey	Has completed training
4	Raychel Roussel	Has completed training
5	Jamill Flannery	Has completed training
6	Serene Pabst	Has completed training
7	Gannon Brenneke	Has completed training
8	Robbyn Lintall	Has completed training
9	Townsend Dunlap	Has completed training
10	Buddy Juden	Has completed training
11	Norrie Severy	Has completed training
12	Beverie Huntriss	Has completed training
13	Trev Gravie	Has not completed training
14	Gwynne Reder	Has completed training
15	Farly Harcombe	Has completed training
16	Aline Harewood	Has completed training
17	Muriel Zambonini	Has completed training
18	Emory Sisley	Has completed training
19	Rodie Hebblewaite	Has not completed training
20	Berk Kiss	Has not completed training

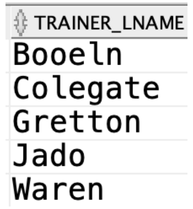
```
SELECT DISTINCT
    cust_id,
    cust_fname
    || ' '
    || cust_lname AS custname,
    'Has completed training' AS trainingstatus
FROM
         drone.customer
    NATURAL JOIN drone.cust_train
UNION
SELECT
    cust_id,
    cust_fname
    || ' '
    || cust_lname,
    'Has not completed training'
FROM
    drone.customer
WHERE
    cust_id NOT IN (
        SELECT
            cust_id
        FROM
            drone.cust_train
    )
ORDER BY
    cust_id;
```

INTERSECTION

Find the trainers who have the same last name as any customer

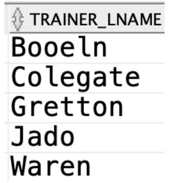


CUST_LNAME
Brenneke
Brightey
Dudgeon
Dunlap
Flannery
Gravie
Harcombe
Harewood
Hebblewaite
Huntriss
Juden
Kiss
Lintall
Pabst
Reder
Roussel
Severy
Sisley
Waren
Zambonini

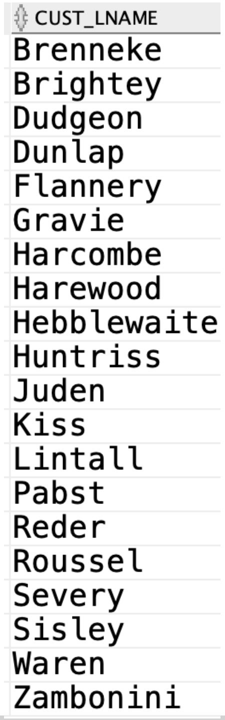


TRAINER_LNAME
Booeln
Colegate
Gretton
Jado
Waren

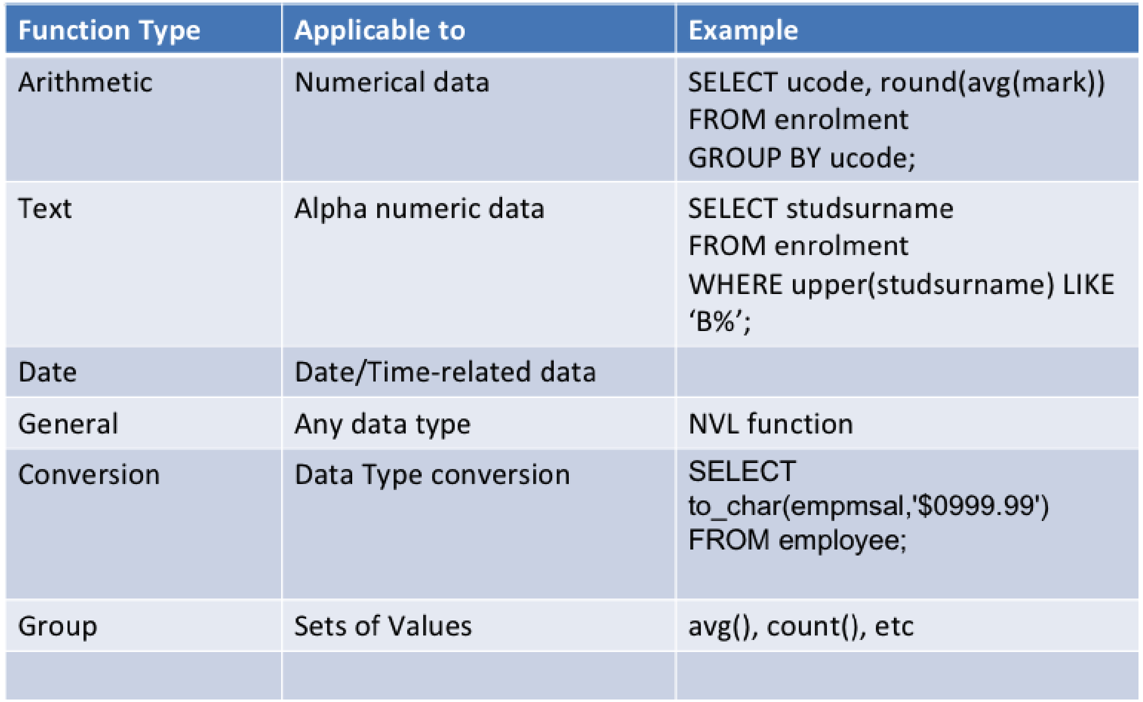
```
SELECT
    trainer_id,
    trainer_rego,
    trainer_fname,
    trainer_lname
FROM
    drone.trainer
WHERE
    trainer_lname IN (
        SELECT
            trainer_lname
        FROM
            drone.trainer
        INTERSECT
        SELECT
            cust_lname
        FROM
            drone.customer
    );
```



TRAINER_LNAME
Booeln
Colegate
Gretton
Jado
Waren



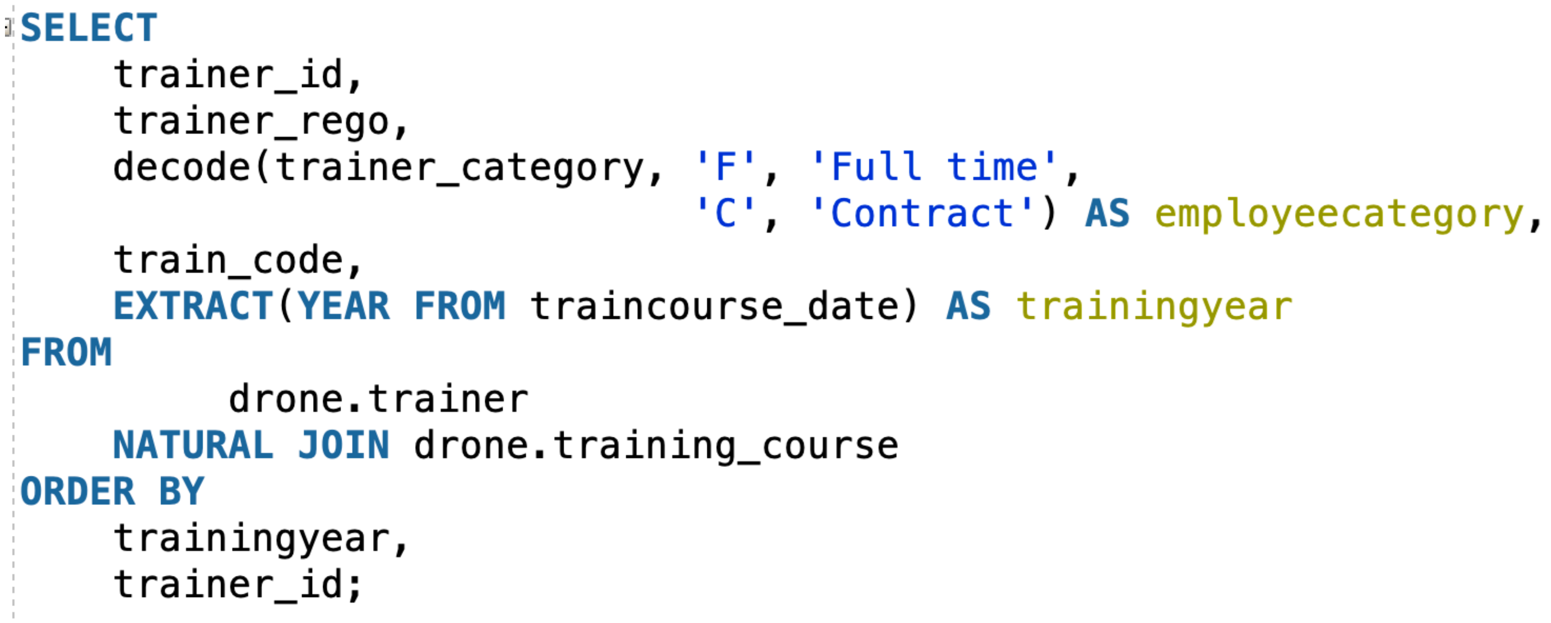
CUST_LNAME
Brenneke
Brightey
Dudgeon
Dunlap
Flannery
Gravie
Harcombe
Harewood
Hebblewaite
Huntriss
Juden
Kiss
Lintall
Pabst
Reder
Roussel
Severy
Sisley
Waren
Zambonini



Function Type	Applicable to	Example
Arithmetic	Numerical data	SELECT ucode, round(avg(mark)) FROM enrolment GROUP BY ucode;
Text	Alpha numeric data	SELECT studsurname FROM enrolment WHERE upper(studsurname) LIKE ‘B%’;
Date	Date/Time-related data	
General	Any data type	NVL function
Conversion	Data Type conversion	SELECT to_char(empmsal,'$0999.99') FROM employee;
Group	Sets of Values	avg(), count(), etc

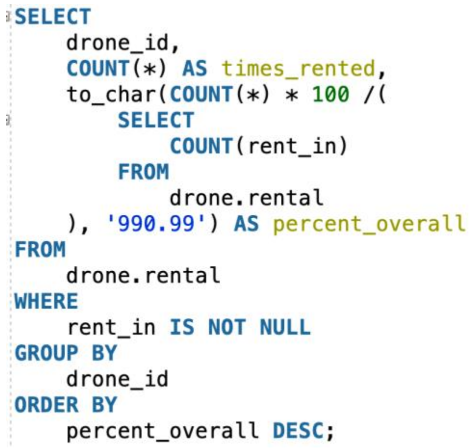
See document on Moodle

EXTRACT and DECODE

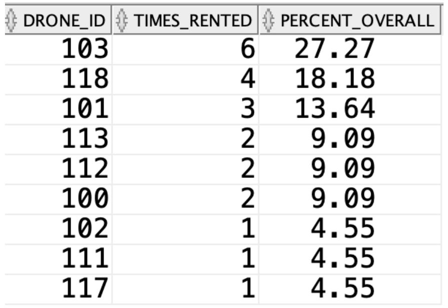


```
SELECT
    trainer_id,
    trainer_rego,
    decode(trainer_category, 'F', 'Full time',
                             'C', 'Contract') AS employeecategory,
    train_code,
    EXTRACT(YEAR FROM traincourse_date) AS trainingyear
FROM
         drone.trainer
    NATURAL JOIN drone.training_course
ORDER BY
    trainingyear,
    trainer_id;
```

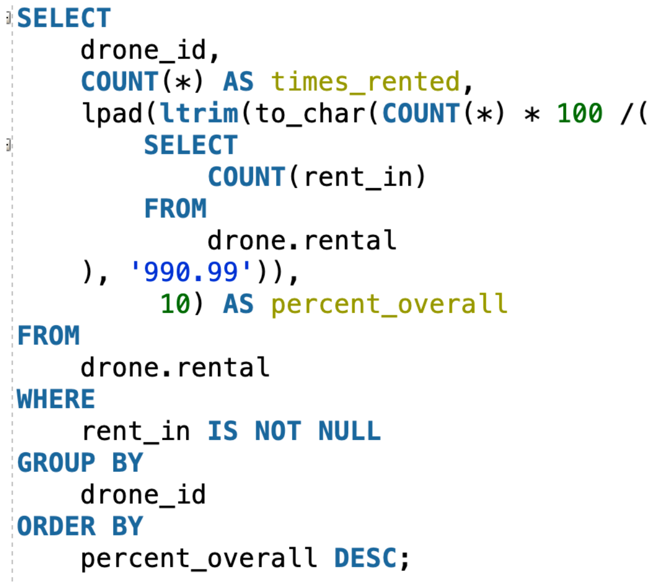
LPAD and LTRIM



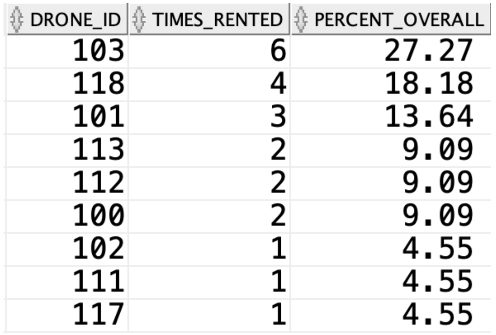
```
SELECT
    drone_id,
    COUNT(*) AS times_rented,
    to_char(COUNT(*) * 100 /(
        SELECT
            COUNT(rent_in)
        FROM
            drone.rental
    ), '990.99') AS percent_overall
FROM
    drone.rental
WHERE
    rent_in IS NOT NULL
GROUP BY
    drone_id
ORDER BY
    percent_overall DESC;
```



DRONE_ID	TIMES_RENTED	PERCENT_OVERALL
103	6	27.27
118	4	18.18
101	3	13.64
113	2	9.09
112	2	9.09
100	2	9.09
102	1	4.55
111	1	4.55
117	1	4.55



```
SELECT
    drone_id,
    COUNT(*) AS times_rented,
    lpad(ltrim(to_char(COUNT(*) * 100 /(
        SELECT
            COUNT(rent_in)
        FROM
            drone.rental
    ), '990.99')),
         10) AS percent_overall
FROM
    drone.rental
WHERE
    rent_in IS NOT NULL
GROUP BY
    drone_id
ORDER BY
    percent_overall DESC;
```



DRONE_ID	TIMES_RENTED	PERCENT_OVERALL
103	6	27.27
118	4	18.18
101	3	13.64
113	2	9.09
112	2	9.09
100	2	9.09
102	1	4.55
111	1	4.55
117	1	4.55

MONASH University