MONASH University

MONASH
INFORMATION
TECHNOLOGY

Week 9 - Structured Query Language (SQL) – Part 2 (Intermediate)

FIT2094 - FIT3171 Databases

Clayton Campus S2 2019.

GROUP OF EIGHT AUSTRALIA

Overview

- **Hour 1**
  - –SQL SELECT - Intermediate
  - –Aggregate Functions: count, min, max, avg, sum
  - –GROUP BY and HAVING clauses.
  - –Subquery: Inner vs outer query; comparison operators (IN, ANY, ALL)

… then COFFEE BREAK!

- **Hour 2**
  - –SQL SELECT - **Actual practice**
  - **–PL/SQL Triggers.**

[CLAYTON ONLY] Week 9 chat (reminder)...

ATTENTION: Week 9 and Grand Final holiday might affect your tutorial THIS week.

IF YOU HAVE A FRIDAY TUTE:

- **In MSB: more open consultation sessions if you need to catch up with a tutor for help etc.**
  a. **and you may get marked for participation in the open consultation in lieu of Week 9 Tute**
- **Other options: do self study at home, and email your tutor (or share on Google Drive) your answers to the Week 9 tute questions for participation marks.**
  a. **Note that you need to get most of the answers correct and demonstrate a decent attempt at answering them.**

NB: Following week is the MSB :-)

[Clayton] Audience Q&A

Next week is the MSB

Have a good break, but remember to prepare for the Unit Test in Week 10! A sample test will be available on Moodle under the MSB section.

Hence - please make good use of the forums for discussion. Also, consultations will be running on the break (see Moodle for more details).

SELECT...
(Intermediate lessons)

Img src: @samuelzeller at Unsplash /

Aggregate Functions

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

```
SELECT max(mark)
FROM enrolment;
```

```
SELECT min(mark)
FROM enrolment;
```

```
SELECT avg(mark)
FROM enrolment;
```

```
SELECT count(stu_nbr)
FROM enrolment
WHERE mark >= 50;
```

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

Q1. What will be displayed by the following SQL statement?

```
SELECT count(*), count(mark)
FROM enrolment;
```

A. 8, 8

B. 8, 3

C. 3, 3

D. 3, 8

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

Q2. What will be displayed by the following SQL statement?

```
SELECT count(*), count(stu_nbr), count(distinct stu_nbr)
FROM enrolment;
```

A. 8, 8, 4

B. 8, 8, 8

C. 8, 4, 8

D. 8, 4, 4

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

Q3. We want to calculate the average mark of the 8 rows in the above table. What SQL statement should we use?
Note: We want to calculate (78+35+65)/8=22.25

A. `SELECT avg(mark) FROM enrolment;`

B. `SELECT sum(mark)/count(mark) FROM enrolment;`

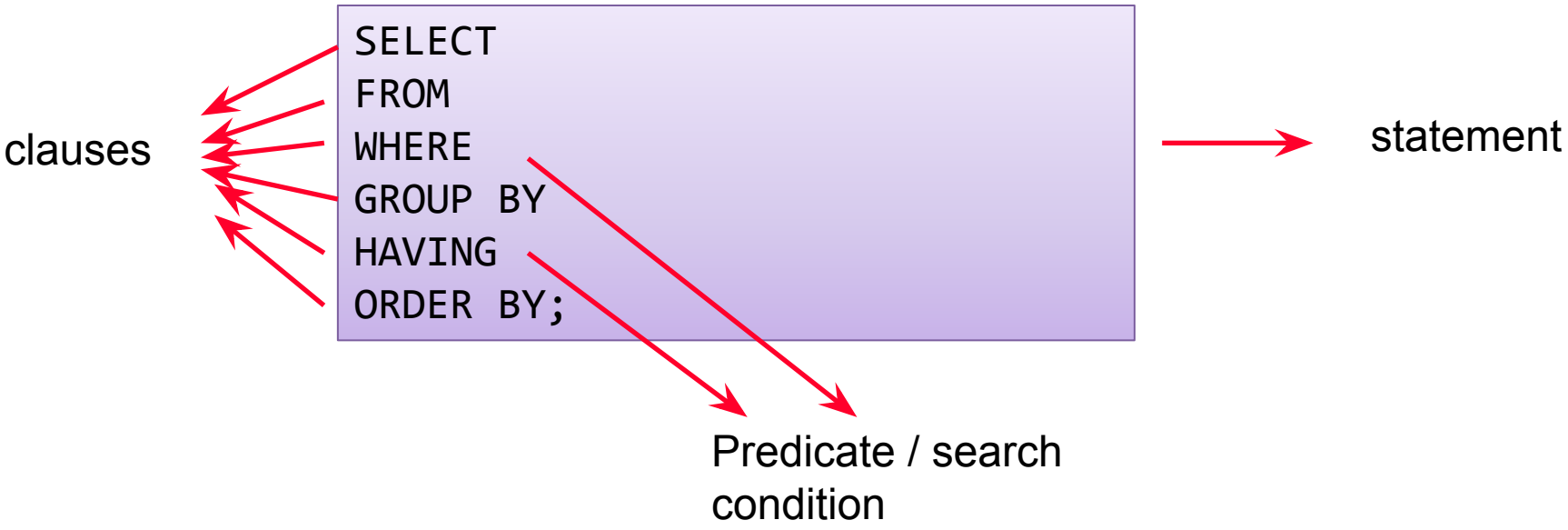
C. `SELECT sum(mark)/count(*) FROM enrolment;`

D. `SELECT avg(NVL(mark,0)) FROM enrolment;`

E. None of the above.

F. More than one option is correct.

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.**

```
-- Example A
SELECT avg(mark)
FROM enrolment;
```

```
-- Example B
SELECT unit_code, avg(mark)
FROM enrolment
GROUP BY unit_code;
```

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

```
-- Example A
SELECT avg(mark)
FROM enrolment;
```

```
-- Example B
SELECT unit_code, avg(mark)
FROM enrolment
GROUP BY unit_code;
```

```
-- Example C
SELECT unit_code, avg(mark), count(*)
FROM enrolment
GROUP BY unit_code;
```

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

```
-- Example D
SELECT unit_code, avg(mark), count(*)
FROM enrolment
GROUP BY unit_code, enrol_year;

-- This is tricky -- but will make more sense
-- when we add enrol_year to the SELECT
```

HAVING clause

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

```
SELECT unit_code, count(*)
FROM enrolment
GROUP BY unit_code
HAVING count(*) > 2;
```

HAVING and WHERE clauses

- 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.

HAVING and WHERE clauses

- On the example, the logic of the process will be:
  - **All rows where mark is NULL are retrieved. (due to the WHERE clause)**
  - The retrieved rows then are grouped into different unit_code.
  - **If the number of rows in a group is greater than 1, the unit_code and the total is displayed. (due to the HAVING clause)**

```
SELECT unit_code,
count(*)
FROM enrolment
WHERE mark IS NULL
GROUP BY unit_code
HAVING count(*) > 1;
```

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

```
-- Example E
SELECT unit_code,
avg(mark), count(*)
FROM enrolment
GROUP BY unit_code
HAVING count(*) > 2;
```

```
-- Example F
SELECT unit_code,
avg(mark), count(*)
FROM enrolment
GROUP BY unit_code
HAVING avg(mark) > 55;
```

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	(null)	(null)
3	11111111	FIT1004	2013	1	(null)	(null)
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	(null)	(null)
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	(null)	(null)
8	11111114	FIT1004	2013	1	(null)	(null)

```
-- Example G
SELECT unit_code, avg(nvl(mark,0)), count(*)
FROM enrolment
WHERE enrol_semester = '1'
GROUP BY unit_code
HAVING count(*) > 2
ORDER BY avg(nvl(mark,0)) DESC;
```

Unit_code	Mark	Studid	Year
FIT2094	80	111	2016
FIT2094	20	111	2015
FIT2004	100	111	2016
FIT2004	40	222	2015
FIT2004	40	333	2015

Q4. What is the output for:

```
SELECT unit_code, avg(mark), studid
FROM enrolmentA
GROUP BY unit_code
HAVING avg(mark) > 55;
```

A. FIT2094, 50, 111
B. FIT2004, 60, 111
C. FIT2004, 60, 111, 222, 333
D. FIT2004, 100, 111
E. Will print three rows
F. Error

ORA-00979 Errors!

```
SELECT stu_nbr, unit_code, avg(mark)
FROM enrolment
GROUP BY unit_code;
```

The above SQL generates error message
`"ORA-00979: not a GROUP BY expression"`

Why and how to fix this?

- Why? Because the grouping is based on the **unit_code**, whereas the display is based on stu_nbr The two groups may not have the same members.
- How to fix this?
  - Include the stu_nbr 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.**

ORA-00979 Errors!

```
SELECT unit_code, avg(mark), studid
FROM enrolmentA
GROUP BY unit_code
HAVING avg(mark) > 55;
```

The above SQL generates error message
"ORA-00979: not a GROUP BY expression"

Why and how to fix this?

- Why? avg(mark) is producing ONE average to be grouped by unit_code.
  e.g. FIT1234 has avg of 22.2
  … but what is studid doing there?
- How to fix this?
  - either remove it or make it a summary statistic, e.g. count.
- **More:**
  https://www.tekstream.com/oracle-error-messages/ora-00979-not-a-group-by-expression/

Subqueries

- Query within a query.

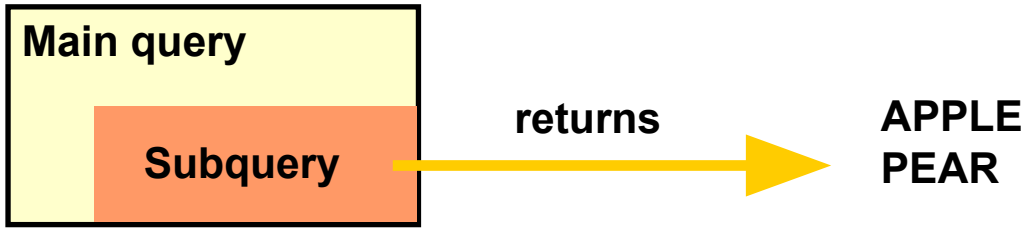
"Find all students whose mark is higher than the average mark of all enrolled students"

```
SELECT *
FROM enrolment
WHERE mark > (SELECT avg (mark)
              FROM enrolment );
```

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?
Trick question!

```
SELECT *
FROM enrolment
WHERE mark > (SELECT avg(mark)
          FROM enrolment
          GROUP BY unit_code);
```

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

✖ ORA-01427: single-row subquery returns more than one row

Q6. What will be returned by the inner query?

```
SELECT unit_code, stu_lname, stu_fname, mark
FROM  enrolment e join student s
       on e.stu_nbr = s.stu_nbr
WHERE (unit_code, mark) IN
                  (SELECT unit_code, max(mark)
                   FROM enrolment
                   GROUP BY unit_code);
```

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

B. A list of values.

C. Multiple columns, multiple rows.

D. None of the above.

Comparison Operators for Subquery

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

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	80	HD
3	11111111	FIT1004	2013	1	85	HD
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	50	P
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	89	HD
8	11111114	FIT1004	2013	1	50	P

Q7. Which row(s) in ENROL2 table will be retrieved by the following SQL statement?

```
SELECT * FROM enrol2
WHERE mark IN (SELECT max(mark)
               FROM enrol2
               GROUP BY unit_code);
```

A. 1, 2, 7

B. 7

C. 2,3,7

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	80	HD
3	11111111	FIT1004	2013	1	85	HD
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	50	P
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	89	HD
8	11111114	FIT1004	2013	1	50	P

Q8. Which row/s in ENROL2 will be retrieved by the following SQL statement?

```
SELECT * FROM enrol2
WHERE mark > ANY (SELECT avg(mark)
        FROM enrol2
        GROUP BY unit_code);
```

UCODE	ROUND(AVG(MARK))
FIT1001	57
FIT1002	80
FIT1004	75

A. 1,2,3,6,7
B. 2,3,7
C. 3,7
D. No rows will be returned
E. All answers above WRONG

	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	80	HD
3	11111111	FIT1004	2013	1	85	HD
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	50	P
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	89	HD
8	11111114	FIT1004	2013	1	50	P

Q9. Which row/s in ENROL2 will be retrieved by the following SQL statement?

```
SELECT * FROM enrol2
WHERE mark > ALL (SELECT avg(mark)
        FROM enrol2
        GROUP BY unit_code);
```

UCODE	ROUND(AVG(MARK))
FIT1001	57
FIT1002	80
FIT1004	75

A. 1,2,3,6,7

B. 2,3,7

C. 3,7

D. No rows will be returned

E. All answers above WRONG

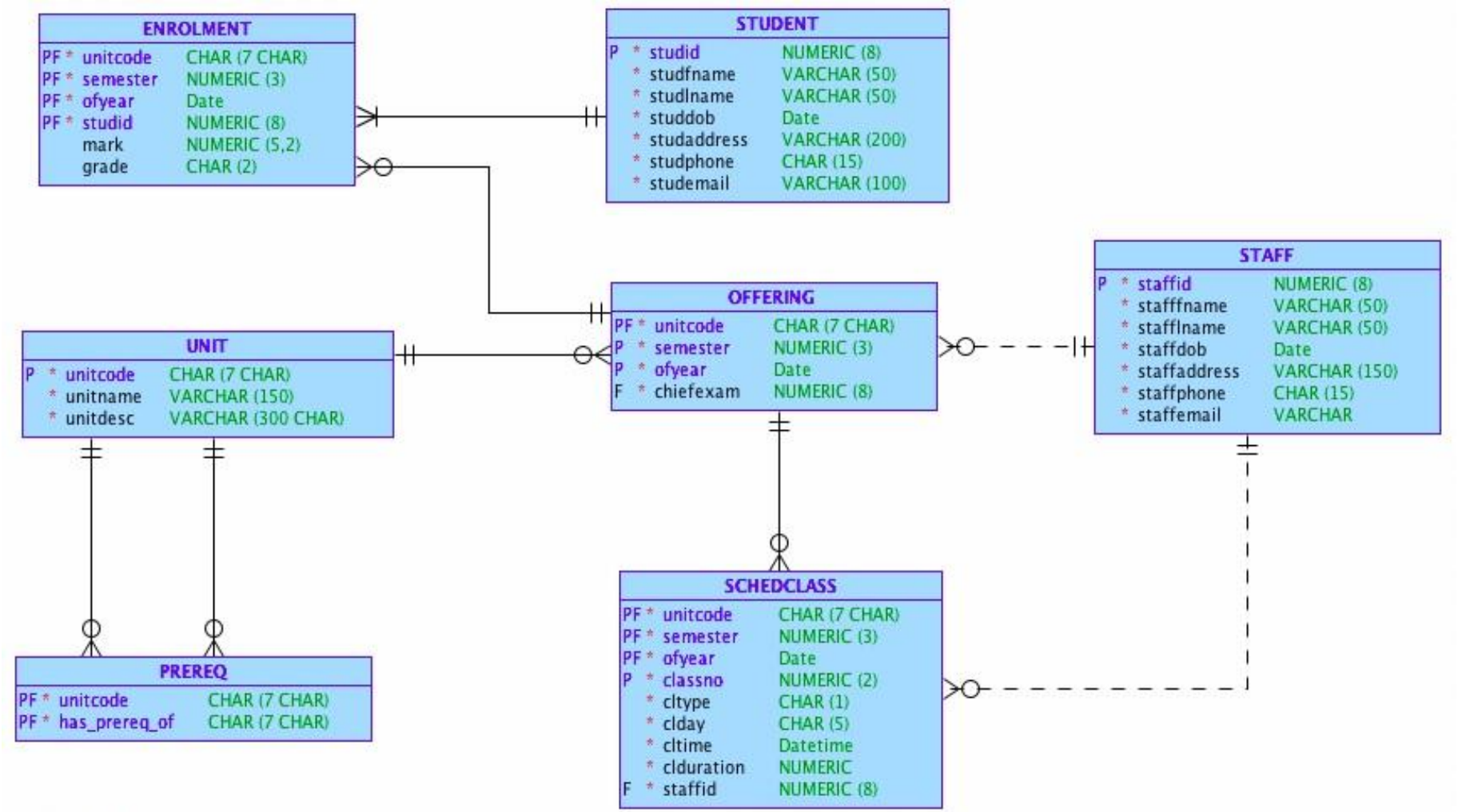
Coffee break - see you in 10 minutes.

Img src: Jazmin Quaynor @jazminantoinette at Unsplash

Summary of Intermediate SQL learnt today

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

Practice



University data model

Q1. Display the number of enrolments with a mark assigned, and the average mark for all enrolments

Q2. Select the highest mark ever in any unit

Q3. Select the highest mark ever for each unit (show the unit code only)

Q4. For each student (show the id only), select the highest mark he/she ever received

Q5. For each unit, print unit code, unit name and the highest mark ever in that unit. Print the results in descending order of highest marks.

Q6. For each offering of a unit with marks show the unit code, unitname, offering details and and average mark

Q7. For each student that is enrolled in at least 3 different units, print his/her name and average mark. Also, display the number of units he/she is enrolled in.

Q8. For each unit, count the total number of HDs

Q9. For each unit, print the total number of HDs, Ds, and Cs. The output should contain three columns named unitcode, grade_type, num where grade_type is HD, D or C and num is the number of students that obtained the grade.

Q10. For each unit, print the student ids of the students who obtained maximum marks in that unit

PL/SQL: Triggers

Img src: @samuelzeller at Unsplash /

Oracle Triggers

- **This is Oracle-specific functionality in PL/SQL**
- "PL/SQL is a procedural language designed specifically to embrace SQL statements within its syntax. PL/SQL program units are compiled by the Oracle Database server and stored inside the database."
  - https://www.oracle.com/technetwork/database/features/plsql/index.html

- BOTH FIT2094 and FIT3171 have to study this.
- **FIT3171 WILL be examined more rigorously on Oracle Triggers due to ULO #7…**
  - **This means more quality/quantity of Q's...**
  - **ULO "7. develop … with a database backend;"**

Oracle Triggers

- A trigger is **PL/SQL code** associated with a table, which **performs an action** when a row in a table is inserted, updated, or deleted.
- Triggers are used to implement some types of **data integrity constraints that cannot be enforced at the DBMS design** and implementation levels
- A trigger is a **stored procedure/code block** associated with a table
- Triggers specify a **condition** and an **action** to be taken whenever that condition occurs
- The DBMS **automatically executes** the trigger when the condition is met ("fires")
- A Trigger can be ENABLE'd or DISABLE'd via the ALTER command
  - `ALTER TRIGGER trigger_name ENABLE;`

Oracle Triggers - general form

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
    {BEFORE | AFTER | INSTEAD OF }
    {UPDATE | INSERT | DELETE}
        [OF <attribute_name>] ON <table_name>
    [FOR EACH ROW]
    [WHEN]
DECLARE
BEGIN
        …. trigger body goes here …..
END;
```

Triggering Statement

```
BEFORE|AFTER  INSERT|UPDATE [of colname]|DELETE  ON  Table
```

- The triggering statement specifies:
  - the **type** of SQL statement that fires the trigger body.
  - the possible **options** include DELETE, INSERT, and UPDATE. One, two, or all three of these options can be included in the triggering statement specification.
  - the **table** associated with the trigger.
- Column List for UPDATE
  - if a triggering statement specifies **UPDATE**, *an optional **list of columns** can be included in the triggering statement*.
  - if you include a column **list**, the trigger is fired on an UPDATE statement only when one of the **specified** columns is updated.
  - if you **omit** a column list, the trigger is fired when **any** column of the associated table is updated

Trigger Body

```
BEGIN
   .....
END;
```

- is a PL/SQL block that can include SQL and PL/SQL statements. These statements are executed if the triggering statement is issued and the trigger restriction (if included) evaluates to TRUE.
- Within a trigger body of a row trigger, the PL/SQL code and SQL statements have access to the **old** and **new** column values of the current row affected by the triggering statement.
- Two correlation names exist for every column of the table being modified: **one for the old column value** and **one for the new column value**.

Correlation Names

- Oracle uses two correlation names in conjunction with every column value of the current row being affected by the triggering statement. These are denoted by:

  `OLD.ColumnName & NEW.ColumnName`

  - For DELETE, only OLD.ColumnName is meaningful
    - –Nothing NEW to use.
  - For INSERT, only NEW.ColumnName is meaningful
    - –No OLD value to use?
  - For UPDATE, both are meaningful
- A colon must precede the OLD and NEW qualifiers when they are used in a trigger's body, but a colon is not allowed when using the qualifiers in the WHEN clause **-- see example later.**
- Old and new values are available in both BEFORE and AFTER **row triggers**.

FOR EACH ROW Option

- The FOR EACH ROW option determines whether the trigger is a row trigger or a statement trigger. If you specify **FOR EACH ROW, the trigger fires once for each row of the table that is affected** by the triggering statement.
- The absence of the FOR EACH ROW option means that the trigger fires only once for each applicable statement, but **not separately for each row** affected by the statement.

```
CREATE OR REPLACE TRIGGER display_salary_increase
AFTER UPDATE OF empmsal ON employee
FOR EACH ROW
WHEN (new.empmsal > 1000)
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Employee:  '|| :new.empno ||' Old
  salary:  '|| :old.empmsal || ' New salary:  '||
  :new.empmsal);
END;
```

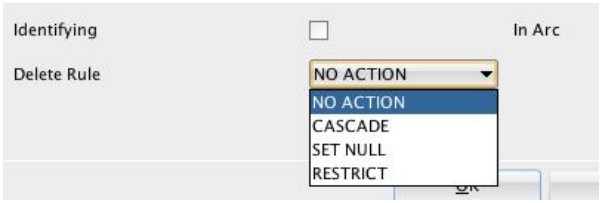
NB: A colon must precede the OLD and NEW qualifiers when they are used in a trigger's body, but a colon is not allowed when using the qualifiers in the WHEN clause

Statement Level Trigger

- Executed once for the whole table but will have to check all rows in the table.
- In many cases, it will be inefficient.
- **No access to the correlation values :new and :old.**

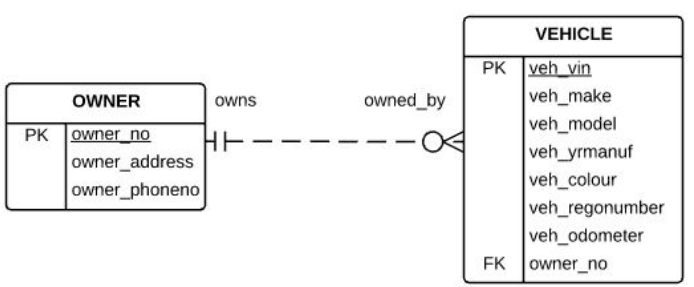
Oracle Data FK Integrity

- Oracle offers the options:
  - UPDATE
    - no action (the default - not specified)
  - DELETE
    - no action (the default - not specified)
    - cascade
    - set null



- Subtle difference between "no action" and "restrict"
  - RESTRICT - will not allow action if child records exist, checks first
  - NO ACTION - allows action and any associated triggers, *then* checks integrity
- Databases implementations vary, for example:
  - **Oracle has no RESTRICT :(**
  - IBM DB2, SQLite implement both as above

Common use of triggers



- In the model above OWNER is the PARENT (PK end) and VEHICLE is the CHILD (FK end)
- What should the database do to maintain integrity if the user:
  - attempts to UPDATE the owner_no of the owner (parent)
  - attempts to DELETE an owner who still has vehicles in the vehicle table
- Oracle, by default, takes the safe approach
  - UPDATE NO ACTION (no update of PK permitted if child records)
  - DELETE NO ACTION (no delete permitted if child records)
  - what if you as the developer want UPDATE CASCADE?
  - **MySQL but not Oracle supports UPDATE CASCADE :(**
    https://dev.mysql.com/doc/refman/8.0/en/create-table-foreign-keys.html

Implement UPDATE CASCADE rule
OWNER 1 ---- has --- M VEHICLE
:new.owner_no – value of owner_no after update
:old.owner_no – value of owner_no before update

Oracle Triggers

```
CREATE OR REPLACE TRIGGER Owner_Upd_Cas
BEFORE UPDATE OF owner_no ON owner
FOR EACH ROW
BEGIN
    UPDATE vehicle
    SET    owner_no = :new.owner_no
    WHERE  owner_no = :old.owner_no;
    DBMS_OUTPUT.PUT_LINE ('Corresponding owner number in the
  VEHICLE table has also been updated');
END;
/
```

NB: we want it to affect each changed row to UPDATE CASCADE

- SQL Window: To CREATE triggers, include the RUN command (/) after the last line of the file
- **NB: lines in red resemble the SQL you're used to!**

Mutating Table

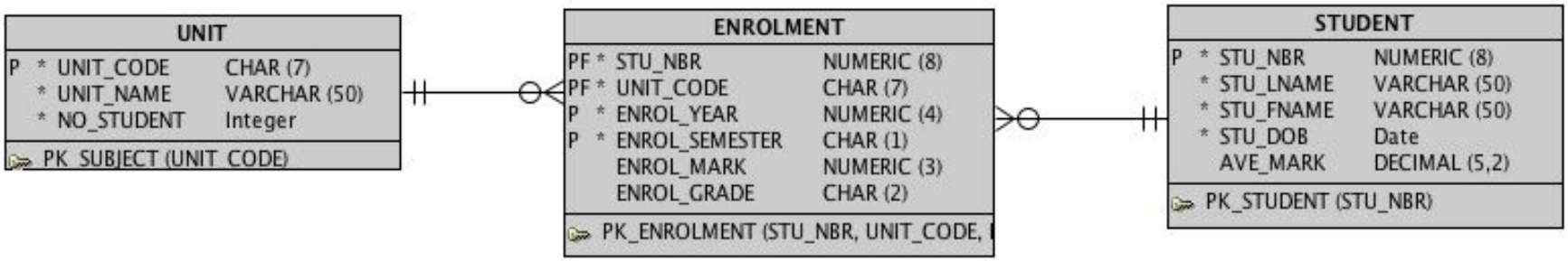
- A table that is currently being modified through an INSERT, DELETE or UPDATE statement **SHOULD NOT be read from or written to because it is in a transition state** between two stable states (before and after - recall Wk8) where data integrity can be guaranteed.
  - Such a table is called **mutating table.**

```
CREATE OR REPLACE TRIGGER Owner_Upd_Cas BEFORE
  UPDATE OF owner_no ON owner
  FOR EACH ROW

  DECLARE
    owner_count NUMBER;

  BEGIN

    SELECT COUNT(*) INTO owner_count
    FROM owner
    WHERE owner_no = :old.owner_no;

    IF owner_count = 1 THEN
      UPDATE vehicle
      SET owner_no = :NEW.owner_no
      WHERE owner_no = :OLD.owner_no;
      DBMS_OUTPUT.PUT_LINE ('Corresponding owner number in the VEHICLE table '
      || 'has also been updated');
    END IF;

  END;
```

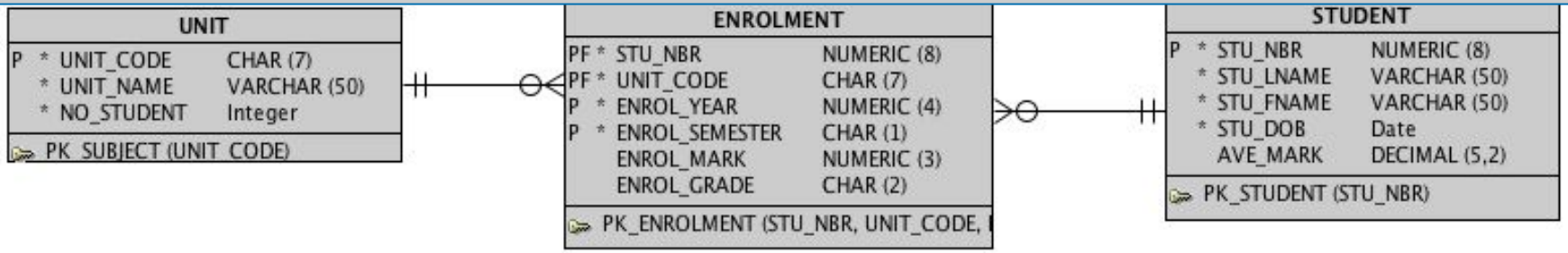
```
update owner set owner_no = 1 where owner_no = 2
Error report -
SQL Error: ORA-04091: table LSMI1.OWNER is mutating, trigger/function may not see it
ORA-06512: at "LSMI1.OWNER_UPD_CAS", line 6
ORA-04088: error during execution of trigger 'LSMI1.OWNER_UPD_CAS'
04091. 00000 -  "table %s.%s is mutating, trigger/function may not see it"
*Cause:    A trigger (or a user defined plsql function that is referenced in
           this statement) attempted to look at (or modify) a table that was
           in the middle of being modified by the statement which fired it.
*Action:   Rewrite the trigger (or function) so it does not read that table.
```

PL/SQL: Triggers
Case Study - Student database

Img src: @samuelzeller at Unsplash /



- The student enrolment database contains **two derived attributes** no_student (total number of students) and ave_mark (average mark) -- assume they're there due to strict business rules e.g. Auditors.
- The total number of students is updated when an enrolment is added or deleted.
- The average mark is updated when an update on attribute mark is performed.
- For audit purpose, any deletion of enrolment needs to be recorded. The recorded information includes the username who performed the deletion, the date and time of the deletion, the student no and unit code.



Q10. Based on the rule to maintain the integrity of the no_student attribute in the UNIT table as well as keeping the audit record, a trigger needs to be created for __ table. The trigger will update a value on __ table and insert a row to __ table.

A. UNIT, ENROLMENT, AUDIT

B. ENROLMENT, UNIT, AUDIT

C. STUDENT, ENROLMENT, AUDIT

D. AUDIT, UNIT, ENROLMENT

Walkthrough - Student Auditing 1

```
CREATE OR REPLACE TRIGGER triggername

BEFORE|AFTER
   INSERT|UPDATE [of colname]|DELETE  [OR ...]
   ON  Table

FOR EACH ROW
DECLARE
   var_name   datatype [, ...]
BEGIN
   .....
END;
```

Q11. What would be an appropriate condition for the trigger described on the previous slide?

A. BEFORE INSERT OR DELETE ON enrolment.

B. AFTER INSERT OR DELETE ON enrolment.

C. BEFORE UPDATE OF mark ON enrolment.

D. AFTER UPDATE OF mark ON enrolment.

Walkthrough - Student Auditing 2

```
CREATE OR REPLACE TRIGGER change_enrolment
AFTER INSERT OR DELETE ON ENROLMENT
FOR EACH ROW
DECLARE
   ??????
BEGIN
      ????????

END;
```

Q12. What would be the logic to update the no_student attribute in the UNIT table when a new row is INSERT-ed to ENROLMENT?

A.
```
UPDATE unit
  SET no_student = no_student + 1
  WHERE unit_code = unit code of the inserted row
```

B.
```
UPDATE unit
  SET no_student = (SELECT count (stu_nbr)
        FROM enrolment
        WHERE unit_code= unit code of the inserted row)
  WHERE unit_code = unit code of the inserted row
```

C.
```
UPDATE unit
  SET no_student = no_student -1
  WHERE unit_code = unit code of the inserted row
```

D.
```
UPDATE unit
```

Walkthrough - Student Auditing 3

```
CREATE OR REPLACE TRIGGER change_enrolment
AFTER INSERT OR DELETE ON ENROLMENT
FOR EACH ROW
DECLARE
   ??????
BEGIN
      IF INSERTING THEN
      UPDATE unit
      SET no_student = no_student + 1
      WHERE unit_code = :new.unit_code
   ENDIF;
   ????
END;
```

Q13. What would be the logic for the trigger to deal with a deletion of a row in enrolment? Assume that a table audit_trail contains audit_time, user, sno and unitcode attributes.
Recall business rule: "For audit purpose, any deletion of enrolment needs to be recorded."

A.
```
UPDATE unit
  SET no_student = no_student -1
  WHERE unit_code = :old.unit_code;
```

B.
```
INSERT INTO audit_trail VALUES
  (SYSDATE, USER,
        :old.stu_nbr, :old.unit_code);
```

C.
```
UPDATE unit
  SET no_student = no_student – 1
  WHERE unit_code = :new.unit_code;
```

D. a and b.

E. b and c.

Walkthrough - Student Auditing Final.

```
CREATE OR REPLACE TRIGGER change_enrolment
AFTER INSERT OR DELETE ON ENROLMENT
FOR EACH ROW

BEGIN
    IF INSERTING THEN
        UPDATE unit
        SET no_student = no_student + 1
        WHERE unit_code = :new.unit_code;
    END IF;
    IF DELETING THEN
        UPDATE unit
        SET no_student = no_student -1
        WHERE unit_code = :old.unit_code;

        INSERT INTO audit_trail VALUES (SYSDATE, USER,
            :old.stu_nbr, :old.unit_code);
    END IF;
END;
```

Homework: This business rule still needs an auditing trigger…
"The average mark is updated when an update on attribute mark is performed."

Summary of Oracle PL/SQL Triggers

- Use triggers where:
  - a specific operation is performed, to ensure related actions are also performed
  - to enforce integrity where data has been denormalised
  - to maintain an audit trail
  - global operations should be performed, regardless of who performs the operation
  - they do NOT duplicate the functionality built into the DBMS
  - their size is reasonably small (< 50 - 60 lines of code)
- Do not create triggers where:
  - they are recursive
  - they modify or retrieve information from triggering tables