MONASH University

MONASH
INFORMATION
TECHNOLOGY

SQL Intermediate

GROUP
OF EIGHT
AUSTRALIA

Aggregate Functions

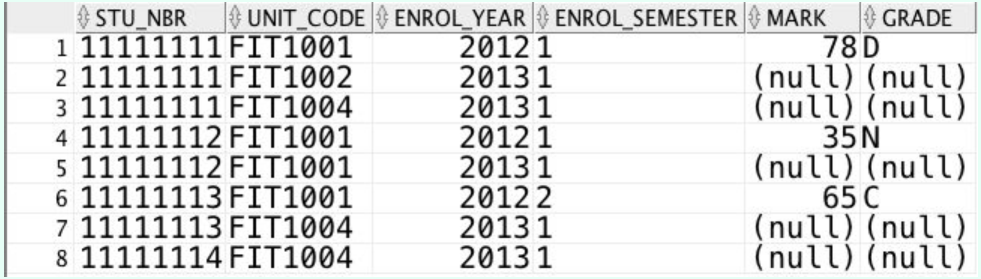
- COUNT, MAX, MIN, SUM, AVG
- Example:

```
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?

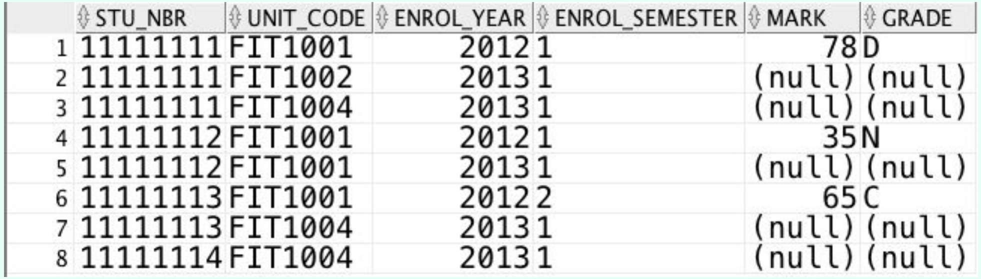
```
SELECT count(*), count(mark)
FROM enrolment;
```

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?

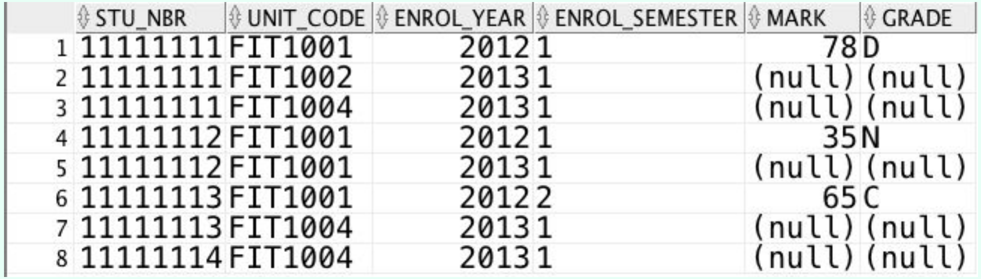
```
SELECT count(*), count(stu_nbr), count(distinct stu_nbr)
FROM enrolment;
```

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;

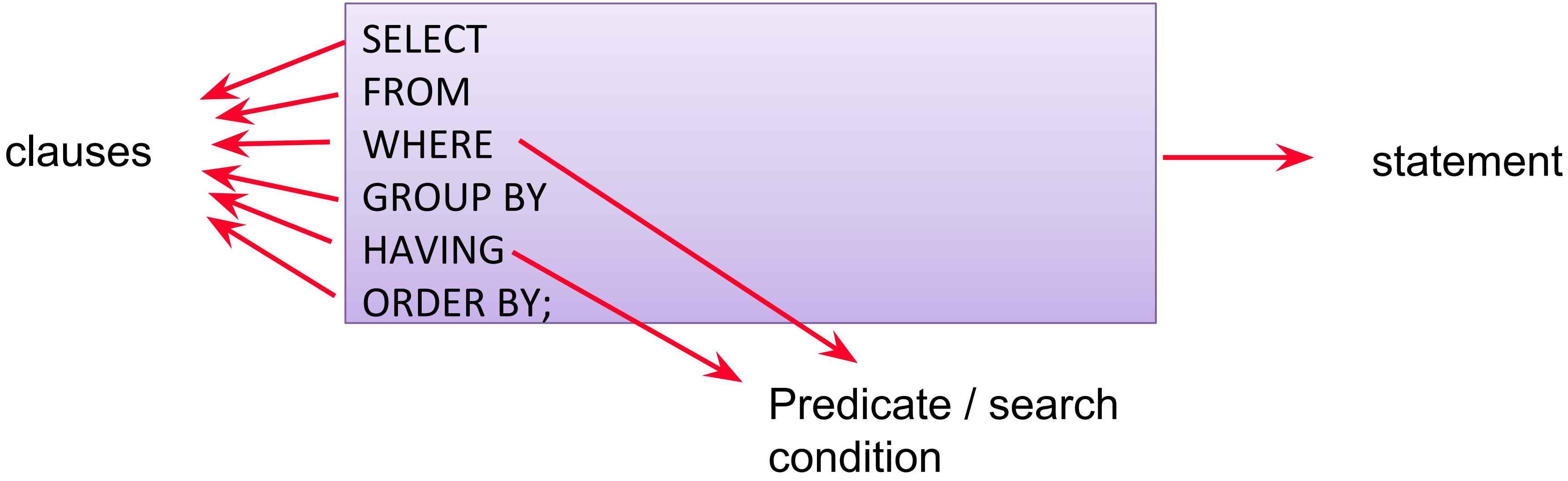
C. SELECT sum(mark)/count(*) FROM enrolment;

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



```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY;
```

clauses

statement

Predicate / search condition

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(mark)
FROM enrolment;
```

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

```
SQL>
SQL> SELECT avg(mark)
  2  FROM enrolment;

 AVG(MARK)
----------
59.3333333

SQL>
SQL> SELECT unit_code, avg(mark)
  2  FROM enrolment
  3  GROUP BY unit_code
  4  ORDER BY unit_code;

UNIT_CO  AVG(MARK)
------- ----------
FIT1001 59.3333333
FIT1002
FIT1004
```

What output is produced?

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

```
SELECT unit_code, avg(mark)
FROM enrolmentA
GROUP BY unit_code
ORDER BY unit_code;
```

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

```
SELECT unit_code, avg(mark), count(*)
FROM enrolmentA
GROUP BY unit_code
ORDER BY unit_code;
```

```
SQL> SELECT avg(mark)
  2  FROM enrolmentA;

 AVG(MARK)
----------
        56

SQL>
SQL> SELECT unit_code, avg(mark)
  2  FROM enrolmentA
  3  GROUP BY unit_code
  4  ORDER BY unit_code;

UNIT_CO  AVG(MARK)
------- ----------
FIT2004         60
FIT2094         50

SQL>
SQL> SELECT unit_code, avg(mark), count(*)
  2  FROM enrolmentA
  3  GROUP BY unit_code
  4  ORDER BY unit_code;

UNIT_CO  AVG(MARK)   COUNT(*)
------- ---------- ----------
FIT2004         60          3
FIT2094         50          2
```

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

What output is produced?

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

```
SELECT unit_code, avg(mark), count(*)
FROM enrolmentA
GROUP BY unit_code, year
ORDER BY unit_code, year;
```

```
SQL> SELECT unit_code, avg(mark), count(*)
  2  FROM enrolmentA
  3  GROUP BY unit_code, year
  4  ORDER BY unit_code, year;

UNIT_CO  AVG(MARK)   COUNT(*)
------- ---------- ----------
FIT2004         40          2
FIT2004        100          1
FIT2094         20          1
FIT2094         80          1
```

Note: attributes in the GROUP BY clause do not have to appear in the select list

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

```
SQL> SELECT unit_code, year, avg(mark), count(*)
  2  FROM enrolmentA
  3  GROUP BY unit_code, year
  4  ORDER BY unit_code, year;

UNIT_CO       YEAR  AVG(MARK)   COUNT(*)
------- ---------- ---------- ----------
FIT2004       2015         40          2
FIT2004       2016        100          1
FIT2094       2015         20          1
FIT2094       2016         80          1
```

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;
```

What output is produced?

```
SELECT unit_code, avg(mark), count(*)
FROM enrolmentA
GROUP BY unit_code
HAVING count(*) > 2
ORDER BY unit_code;
```

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

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

```
SQL> SELECT unit_code, avg(mark), count(*)
  2  FROM enrolmentA
  3  GROUP BY unit_code
  4  HAVING count(*) > 2
  5  ORDER BY unit_code;

UNIT_CO  AVG(MARK)   COUNT(*)
------- ---------- ----------
FIT2004         60          3


SQL>
SQL> SELECT unit_code, avg(mark), count(*)
  2  FROM enrolmentA
  3  GROUP BY unit_code
  4  HAVING avg(mark) > 55
  5  ORDER BY unit_code;

UNIT_CO  AVG(MARK)   COUNT(*)
------- ---------- ----------
FIT2004         60          3
```

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

HAVING and WHERE clauses

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

- 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 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)

What output is produced?

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

```
SELECT unit_code, avg(mark), count(*)
FROM enrolmentA
WHERE year = 2015
GROUP BY unit_code
HAVING avg(mark) > 30
ORDER BY avg(mark) DESC;
```

```
SQL> SELECT unit_code, avg(mark), count(*)
  2  FROM enrolmentA
  3  WHERE year = 2015
  4  GROUP BY unit_code
  5  HAVING avg(mark) > 30
  6  ORDER BY avg(mark) DESC;

UNIT_CO  AVG(MARK)   COUNT(*)
------- ---------- ----------
FIT2004         40          2
```

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

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

A. FIT2094, 111, 50

B. FIT2004, 111, 60

C. FIT2004, 111, 60, 222, 333

D. FIT2004, 111, 100

E. Will print three rows

F. Error

```
SQL> SELECT unit_code, studid, avg(mark)
  2  FROM enrolmentA
  3  GROUP BY unit_code
  4  HAVING avg(mark) > 55
  5  ORDER BY unit_code, studid;

Error starting at line : 1 in command -
SELECT unit_code, studid, avg(mark)
FROM enrolmentA
GROUP BY unit_code
HAVING avg(mark) > 55
ORDER BY unit_code, studid
Error at Command Line : 1 Column : 19
Error report -
SQL Error: ORA-00979: not a GROUP BY expression
00979. 00000 -  "not a GROUP BY expression"
*Cause:
*Action:
```

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

```
SELECT stu_lname, stu_fname, avg(mark)
FROM enrolment e JOIN student s
        ON s.stu_nbr = e.stu_nbr
GROUP BY s.stu_nbr;
```

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 stu_nbr, whereas the display is based on stu_lname and stu_fname. The two groups may not have the same members.
- How to fix this?
  - Include the stu_lname,stu_fname 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.

Subqueries

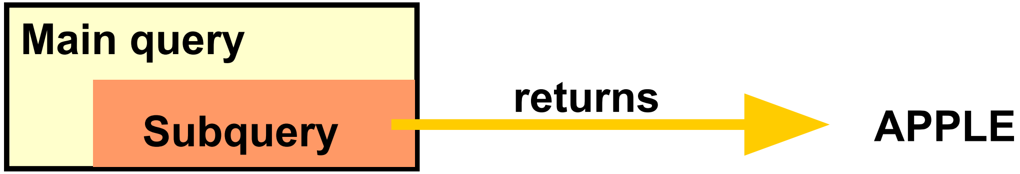
- Query within a query.

"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



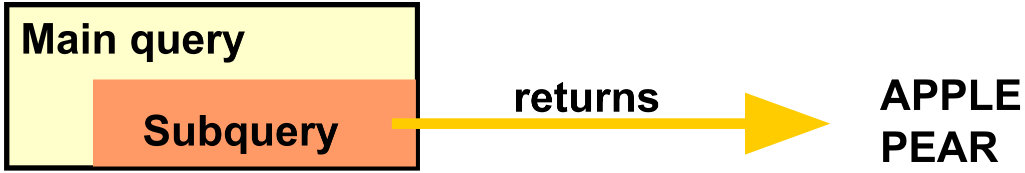
Main query

Subquery

returns

APPLE

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



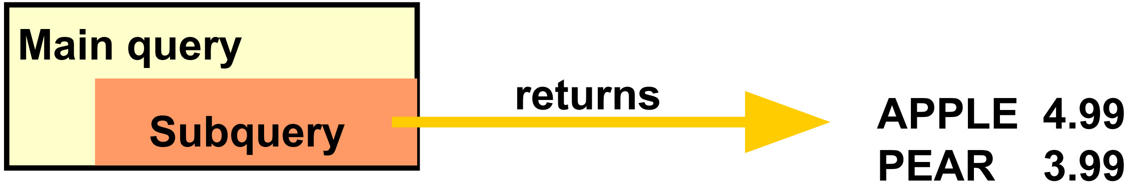
Main query

Subquery

returns

APPLE
PEAR

Multiple-column subquery (many rows, many columns)



Main query

Subquery

returns

APPLE 4.99
PEAR 3.99

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

```
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.

```
SQL> SELECT *
  2  FROM enrolment
  3  WHERE mark > (SELECT avg(mark)
  4           FROM enrolment
  5           GROUP BY unit_code);

Error starting at line : 1 in command -
SELECT *
FROM enrolment
WHERE mark > (SELECT avg(mark)
              FROM enrolment
              GROUP BY unit_code)
Error report -
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).

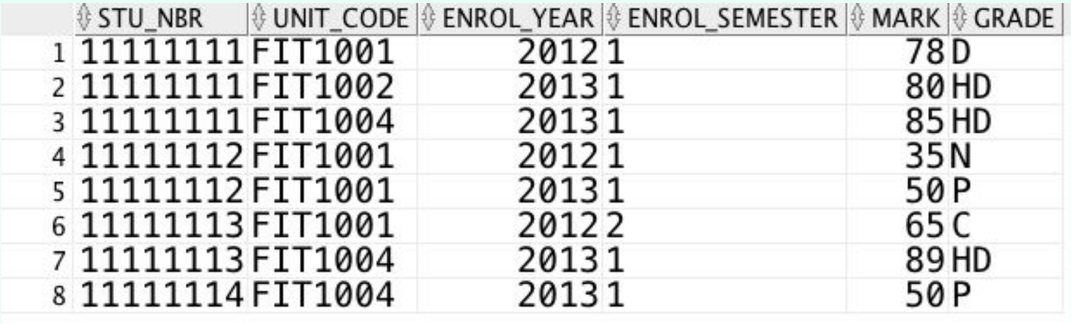
B. A list of values.

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

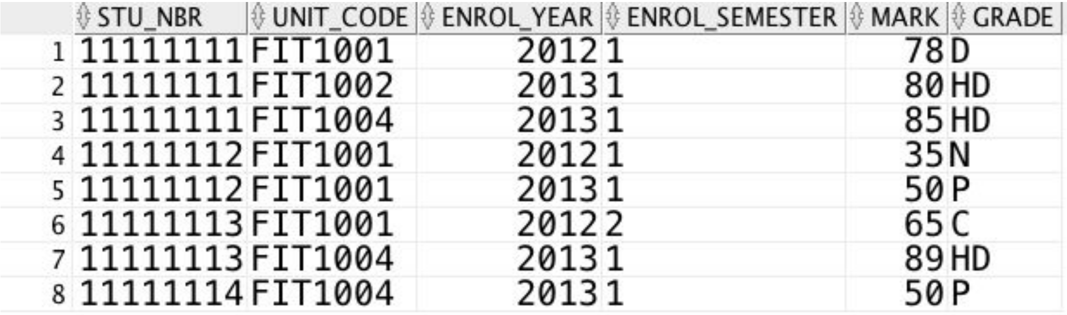
Q7. Which row(s) in ENROL2 table will be retrieved by the following SQL statement?

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

A. 1, 2, 7

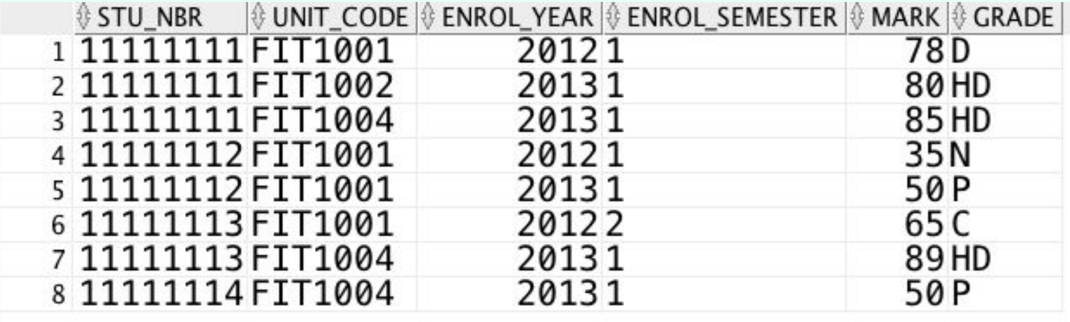
B. 7

C. 2, 3, 7

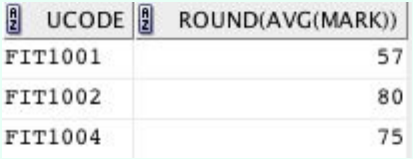


	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	80	HD
3	11111111	FIT1004	2013	1	85	HD
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	50	P
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	89	HD
8	11111114	FIT1004	2013	1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

```
SQL> SELECT * FROM enrol2
  2  WHERE mark IN (SELECT max(mark)
  3                 FROM enrol2
  4                 GROUP BY unit_code)
  5  ORDER BY stu_nbr, unit_code, enrol_year;

   STU_NBR UNIT_CO ENROL_YEAR E       MARK GRA
---------- ------- ---------- - ---------- ---
  11111111 FIT1001       2012 1         78 D
  11111111 FIT1002       2013 1         80 HD
  11111113 FIT1004       2013 1         89 HD
```



	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



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

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

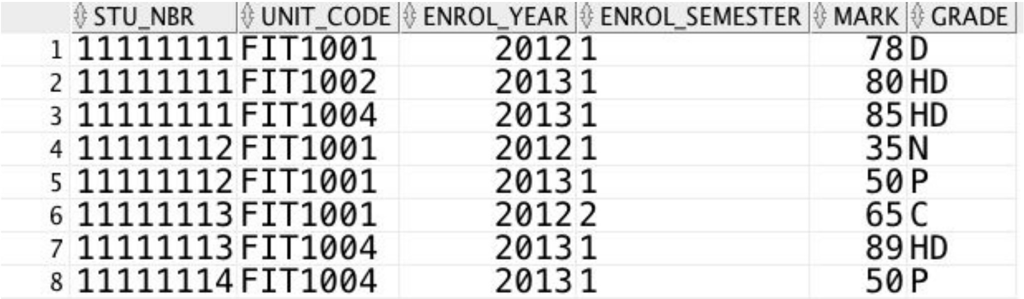
```
SELECT * FROM enrol2
WHERE mark > ANY (SELECT avg(mark)
                     FROM enrol2
                         GROUP BY unit_code);
```

A. 1, 2, 3, 6, 7

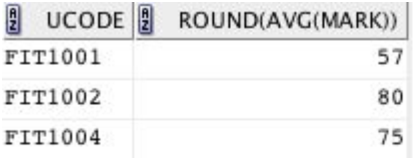
B. 2, 3, 7

C. 3, 7

D. No rows will be returned

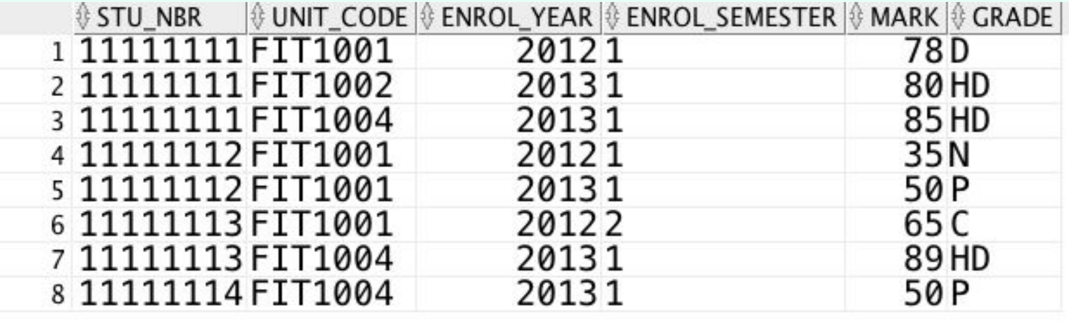


	STU_NBR	UNIT_CODE	ENROL_YEAR	ENROL_SEMESTER	MARK	GRADE
1	11111111	FIT1001	2012	1	78	D
2	11111111	FIT1002	2013	1	80	HD
3	11111111	FIT1004	2013	1	85	HD
4	11111112	FIT1001	2012	1	35	N
5	11111112	FIT1001	2013	1	50	P
6	11111113	FIT1001	2012	2	65	C
7	11111113	FIT1004	2013	1	89	HD
8	11111114	FIT1004	2013	1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

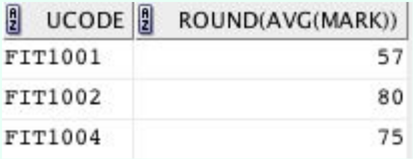


UCODE	ROUND(AVG(MARK))
FIT1001	57
FIT1002	80
FIT1004	75

```
SQL> SELECT * FROM enrol2
  2  WHERE mark > ANY (SELECT avg(mark)
  3                    FROM enrol2
  4                    GROUP BY unit_code)
  5  ORDER BY stu_nbr, unit_code, enrol_year, enrol_semester;

   STU_NBR UNIT_CO ENROL_YEAR E       MARK GRA
---------- ------- ---------- - ---------- ---
  11111111 FIT1001       2012 1         78 D
  11111111 FIT1002       2013 1         80 HD
  11111111 FIT1004       2013 1         85 HD
  11111113 FIT1001       2012 2         65 C
  11111113 FIT1004       2013 1         89 HD
```



	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



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

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

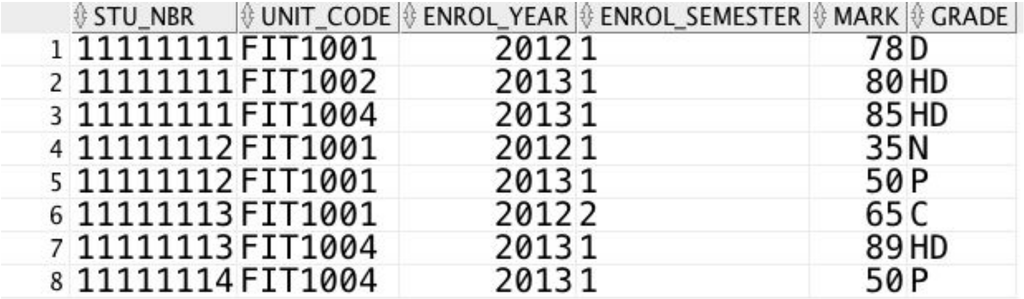
```
SELECT * FROM enrol2
WHERE mark > ALL (SELECT avg(mark)
                  FROM enrol2
                        GROUP BY unit_code);
```

A. 1, 2, 3, 6, 7

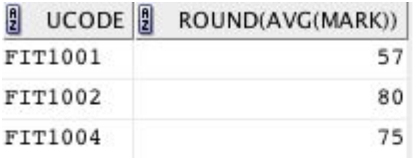
B. 2, 3, 7

C. 3, 7

D. No rows will be returned



	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



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

```
SQL> SELECT * FROM enrol2
  2  WHERE mark > ALL (SELECT avg(mark)
  3                    FROM enrol2
  4                    GROUP BY unit_code)
  5  ORDER BY stu_nbr, unit_code, enrol_year, enrol_semester;

   STU_NBR UNIT_CO ENROL_YEAR E       MARK GRA
---------- ------- ---------- - ---------- ---
  11111111 FIT1004       2013 1         85 HD
  11111113 FIT1004       2013 1         89 HD
```

Q10. Find all students whose mark in any enrolled unit is lower than Wendy Wheat's lowest mark for all units she is enrolled in. What would be a possible inner query statement for the above query (assume Wendy Wheat's name is unique)?

A.
```
SELECT min(mark)
        FROM enrol2
        WHERE stu_lname='Wheat' AND stu_fname='Wendy';
```

B.
```
SELECT min(mark)
        FROM enrol2 e JOIN student s on e.studid = s.studid
        WHERE stu_lname='Wheat' AND stu_fname='Wendy';
```

C.
```
SELECT min(mark) FROM enrol2;
```

D.
```
SELECT mark
        FROM enrol2 e JOIN student s on e.studid = s.studid
        WHERE stu_lname='Wheat' AND stu_fname='Wendy';
```

Summary

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

PART 2
PL/SQL - Triggers (FIT3171)

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
  - For INSERT, only NEW.ColumnName is meaningful
  - 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.
- 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.

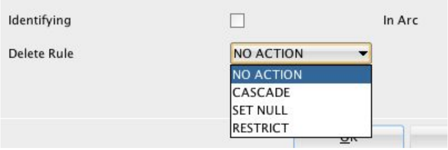
```
CREATE OR REPLACE TRIGGER display_salary_increase
AFTER UPDATE OF empmsal ON employee
FOR EACH ROW
WHEN (new.empmsal > 1000)
BEGIN
    DBMS_OUTPUT.PUT_LINE (‘Employee:  ‘|| :new.empno ||’ Old salary:  ‘||
     :old.empmsal || ‘ New salary:  ‘|| :new.empmsal);
END;
```

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

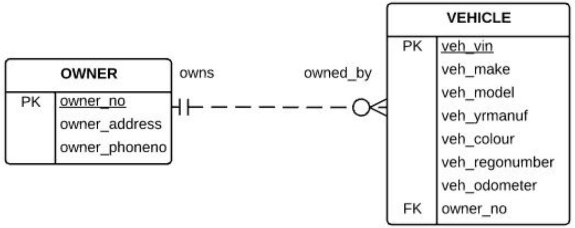
- Oracle offers the options:
  - UPDATE
    - no action (the default - not specified)
  - DELETE
    - no action (the default - not specified)
    - cascade
    - set null
- Subtle difference between "no action" and "restrict"
  - RESTRICT - will not allow action if child records exist, checks first
  - NO ACTION - allows action and any associated triggers, *then* checks integrity
- Databases implementations vary, for example:
  - Oracle no RESTRICT
  - IBM DB2, SQLite implement both as above



Identifying □

Delete Rule NO ACTION

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?

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;
/
```

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

- SQL Window: To CREATE triggers, include the RUN command (/) after the last line of the file

Common use of triggers - data integrity

- A trigger can be used to enforce user-defined integrity by triggering on a preset condition, carrying out some kind of test and then if the test fails, the trigger can raise an error (and stop the action) via a call to **`raise_application_error`**

  The syntax for this call is:

  ```
  raise_application_error(-20000, 'Error message to display');
  ```

  the -20000 is the error number which is reported to the user, the error message is the error message the user will see. The error number can be any number less than or equal to -20000.

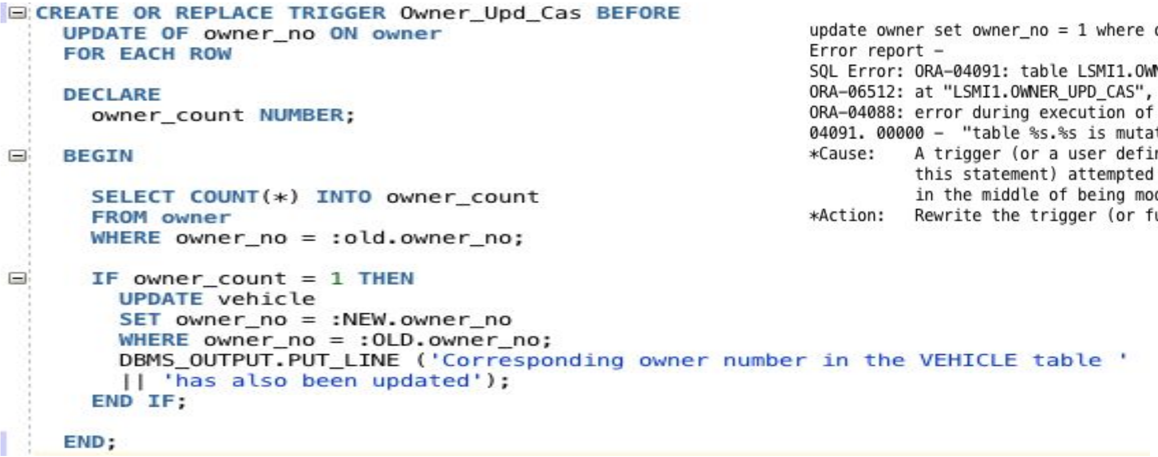
Common use of triggers - data integrity - example

For example: a trigger which will ensure any unit added (ie. inserted) to the UNIT table has a unit code which starts with 'FIT'. Test your trigger and ensure it works correctly and shows your error message.

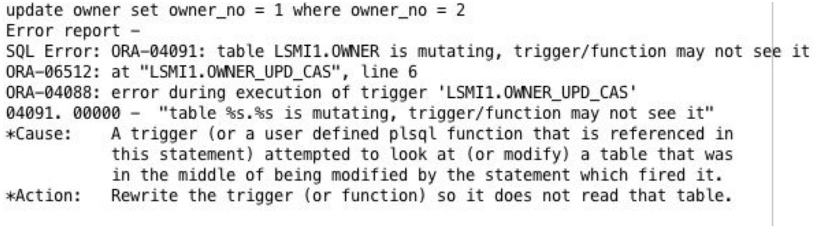
```
CREATE OR REPLACE TRIGGER check_unit_code BEFORE
    INSERT ON unit
    FOR EACH ROW
BEGIN
    IF :new.unit_code NOT LIKE 'FIT%' THEN
        raise_application_error(-20000, 'Unit code must begin with FIT');
    END IF;
END;
/
-- Test Harness
-- display before value
select * from unit;

insert into unit values ('ABC0001','Test Insert',6);

-- display after value
select * from unit;
-- closes transaction
rollback;
```

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) where data integrity can be guaranteed.
    - Such a table is called **mutating table**.

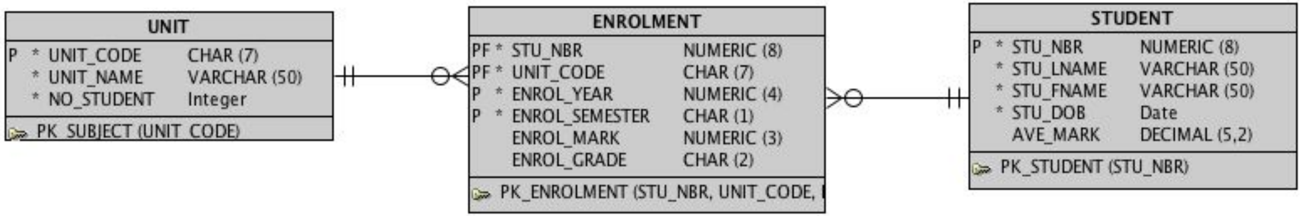


```
CREATE OR REPLACE TRIGGER Owner_Upd_Cas BEFORE
  UPDATE OF owner_no ON owner
  FOR EACH ROW

  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.
```

Triggers Case Study

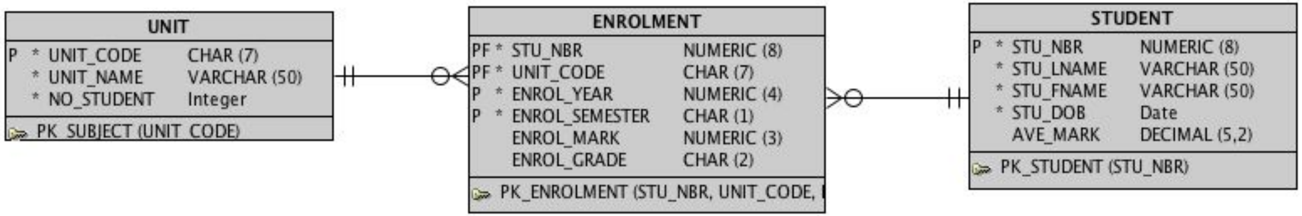


```
UNIT
P  * UNIT_CODE      CHAR (7)
   * UNIT_NAME      VARCHAR (50)
   * NO_STUDENT     Integer
PK_SUBJECT (UNIT_CODE)
```

```
ENROLMENT
PF * STU_NBR          NUMERIC (8)
PF * UNIT_CODE        CHAR (7)
P  * ENROL_YEAR       NUMERIC (4)
P  * ENROL_SEMESTER   CHAR (1)
     ENROL_MARK       NUMERIC (3)
     ENROL_GRADE      CHAR (2)
PK_ENROLMENT (STU_NBR, UNIT_CODE,
```

```
STUDENT
P  * STU_NBR      NUMERIC (8)
   * STU_LNAME    VARCHAR (50)
   * STU_FNAME    VARCHAR (50)
   * STU_DOB      Date
     AVE_MARK     DECIMAL (5,2)
PK_STUDENT (STU_NBR)
```

- The student enrolment database contains two derived attributes no_student (total number of students) and ave_mark (average mark) .
- 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 in an audit table. The recorded information includes the username who performed the deletion, the date and time of the deletion, the student no and unit code.



UNIT			
P	*	UNIT_CODE	CHAR (7)
	*	UNIT_NAME	VARCHAR (50)
	*	NO_STUDENT	Integer
		PK_SUBJECT (UNIT_CODE)	

ENROLMENT			
PF	*	STU_NBR	NUMERIC (8)
PF	*	UNIT_CODE	CHAR (7)
P	*	ENROL_YEAR	NUMERIC (4)
P	*	ENROL_SEMESTER	CHAR (1)
		ENROL_MARK	NUMERIC (3)
		ENROL_GRADE	CHAR (2)
		PK_ENROLMENT (STU_NBR, UNIT_CODE,	

STUDENT			
P	*	STU_NBR	NUMERIC (8)
	*	STU_LNAME	VARCHAR (50)
	*	STU_FNAME	VARCHAR (50)
	*	STU_DOB	Date
		AVE_MARK	DECIMAL (5,2)
		PK_STUDENT (STU_NBR)	

Q5. 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

Oracle Triggers

```
CREATE OR REPLACE TRIGGER triggername

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

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

Q6. 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.

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

Q7. What would be the logic to update the no_student attribute in the UNIT table when a new row is inserted 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
```

```
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;
```

Q8. 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.

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.

```
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;
```

Test Harness

- it is not sufficient to code a trigger only, a suitable test harness must be developed at the same time and used to ensure the trigger is working correctly.

```
-- display before value
select * from unit;

-- test the trigger for insertion
insert into enrolment values (11111111,’FIT2001’,2013,2,null,null);

-- display after value
select * from unit;

-- test the trigger for deletion
delete from enrolment where stu_nbr = 11111111 and unit_code = ’FIT2001’and enrol_year =
2013 and enrol_semester = 2;

-- display after value
select * from unit; select * from audit_trail;
-- closes transaction
rollback;
```

Statement Level Trigger

```
create or replace
TRIGGER DELETE_STATEMENT
AFTER DELETE ON ENROLMENT
BEGIN
    INSERT INTO enrol_history VALUES (SYSDATE, USER, 'Deleted');
END;
```

Row Level Trigger

```
create or replace
TRIGGER DELETE_ENROLMENT
AFTER DELETE ON ENROLMENT
FOR EACH ROW
BEGIN
    INSERT INTO audit_trail VALUES
                 (SYSDATE, USER, :old.stu_nbr, :old.unit_code);
END;
```

Oracle 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