FIT2094-FIT3171 2019 S1 -- Week 9 eBook

Credits and Copyrights:
Authors: FIT3171 2018 S2 UG Databases
FIT Database Teaching Team
Maria Indrawan, Manoj Kathpalia, Lindsay Smith, et al

Change Log:

- Formatted and Imported from Alexandria APRIL 2019.

9.0. SQL Part II – Maintaining data

9.0.1. UPDATE

It is common for data to change value across time. In a database, we use the SQL UPDATE statement to change the value of a cell or cells in a table.

The UPDATE statement consist of three main components:

- The name of the table where the data will be updated.
- The row or the set of rows where the value will be updated.
- The new value to replace the old value.

An example of an UPDATE statement for data in the database we have created by following the exercises in **Tutorial 7 SQL Data Definition Language DDL** is as follows:

```
UPDATE enrolment

SET enrol_mark = 60

WHERE stud_nbr = 11111111 AND

unit_code = 'FIT5132' AND

enrol_semester = '2' AND

enrol_year = 2014;
```

TASKS

1. Update the unit name of FIT9999 from ‘FIT Last Unit’ to ‘place holder unit’.

2. Enter the mark and grade for the student with the student number of 11111113 for the unit code FIT5132 that the student enrolled in semester 2 of 2014. The mark is 75 and the grade is D.
3. The university introduced a new grade classification. The new classification are:
    1. 45 – 54 is P1.
    2. 55 – 64 is P2.
    3. 65 – 74 is C.
    4. 75 – 84 is D.
    5. 85 – 100 is HD.

Change the database to reflect the new grade classification.

9.0.2. DELETE

The DELETE statement is used to remove data from the database. It is important to consider the referential integrity issues when writing a DELETE statement.

In Oracle, a table can be created with FOREIGN KEY constraints with a reference_clause ON DELETE action. The action can be set to CASCADE and SET NULL. When the reference_clause is not specified, the action will be set to RESTRICT[1], in other words, the deletion of row in a parent table (table contains the PRIMARY KEY being referred to by a FOREIGN KEY) will not be allowed when there are rows in the child table (the table with the FOREIGN KEY).

TASKS

[1] Terminology as per SQL standard.

1. A student with student number 11111114 has taken intermission in semester 2 2014, hence all the enrolment of this student for semester 2 2014 should be removed. Change the database to reflect this situation.
2. Assume that Wendy Wheat (student number 11111113) has withdrawn from the university. Remove her details from the database.
3. Add Wendy Wheat back to the database (use the INSERT statements you have created when completing **Tutorial 7 SQL Data Definition Language DDL**). Change the FOREIGN KEY constraints definition for table STUDENT so it will now include the references_clause ON DELETE CASCADE.

   Hint: You need to use the ALTER TABLE statement to drop the FOREIGN KEY constraint first and then put it back using ALTER TABLE with ADD CONSTRAINT clause. A brief description of using ALTER to drop a constraint is available here, the ADD CONSTRAINT was covered in previous tutorials. For more details, you can check the SQL Reference Manual (available from Moodle) for the full syntax and a range of examples.

   Once you have changed the table, now, perform the deletion of the Wendy Wheat (student number 11111113) row in the STUDENT table. Examine the ENROLMENT table. What happens to the enrolment records of Wendy Wheat?

9.1. Transactions Management

In these exercises, you will examine the issues involved in updating shared data.

You will work in pairs. Suppose one user is User1, and the other is User2. *Replace User1 and User2 with your respective Oracle usernames when reading the tutorial exercises - e.g. aabc0012*. **DO NOT REVEAL YOUR PASSWORD TO ANYONE.**

User1 will create a table called **account** which will be shared with User2, that is, both users will be allowed to select data from the table and also update data in the table. This table keeps the account balances of customers, where each customer is identified by a unique id.

Q1. User1 should create the account table. The table will have 2 attributes, **id** and **balance**. Both attributes will have datatype number. After creating the table, User1 should insert two rows of data so that the table looks as below:

	ID	BALANCE
1	1	100
2	2	200

Q2. **User1** can now make the account table available to User2 using the following command:

```
grant select, update on account to User2;
```

Q3. In order for **User2** to access the account table, they would normally have to prefix the account table with the name of the owner, e.g:

```
select * from User1.account;
```

However, we can remove the need to do this if we create a synonym (essentially a system maintained alias for the table) as follows (**User2** issues this command):

```
create synonym account for User1.account;
```

Q4. Make sure both users have the autocommit feature turned OFF – i.e. **BOTH users should issue the command:**

```
set autocommit off
```

Q5. Now, try the following (**maintain the order of the operations**).

- **User1** updates the balance of customer 1 from 100 to 110 (without issuing a commit).
- **User2** views the contents of the account table (do they see the new value? – if not, why not?)
- **User1** issues a commit command.
- **User2** views the contents of the account table (do you notice any difference?)

Explain what is happening in the results of the above queries, in the context of atomic transactions.

Q6. Now we will try and see what happens when we try some concurrent updates of the table (keep the order of transactions the same as below)

- **User1** updates the balance of customer 2 from 200 to 150 (without issuing a commit).

- **User2** tries to update the balance of customer 2 to 100 (what happens?)

Explain what is happening here. What should be done to allow the **User2** update to proceed?

Q7. Now try the following:

- **User1** updates the balance of customer 2 from 200 to 150 (without issuing a commit).
- **User2** tries to update the balance of customer 1 to 125 (what happens?)

How does this differ from the results of the transactions in part 6? What does this tell you about the granularity of locking in Oracle? What must be done in order for the results of both updates to be visible to both users?

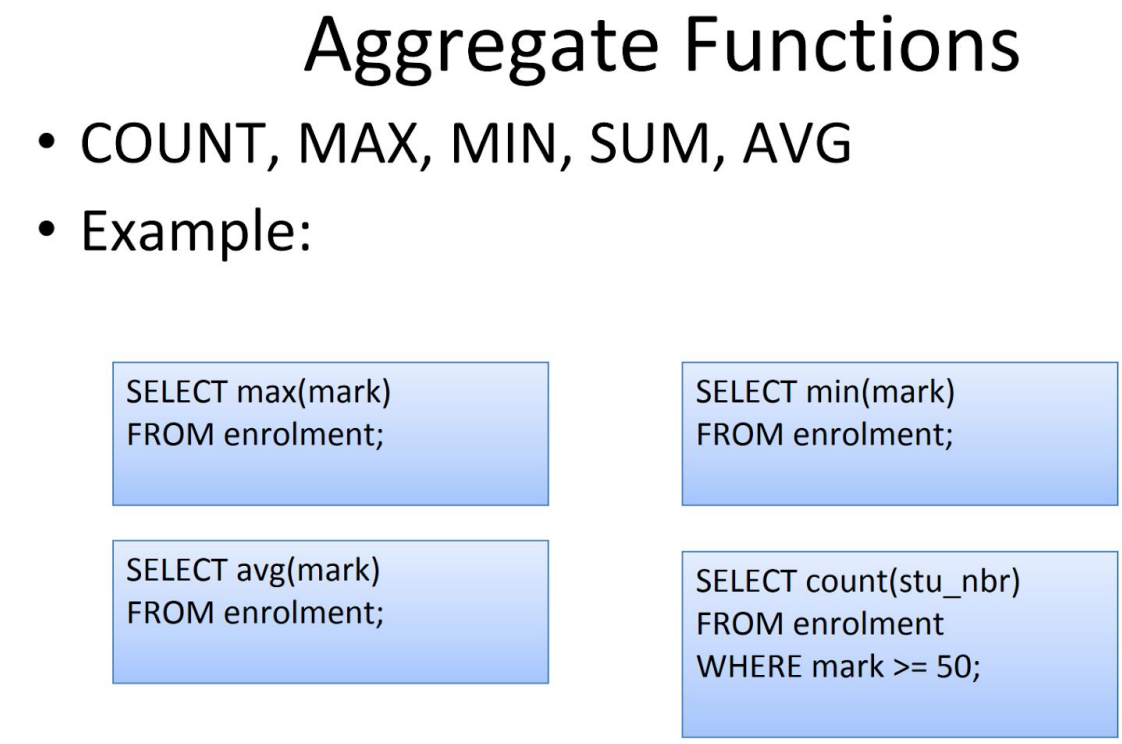
Q8. Try and generate a deadlock between the two users (hint: you will need to set up another shared table). Remember that a deadlock occurs when User1 holds a lock on table A and requests a lock on table B, but table B is locked by User2 who is also requesting a lock on table A.

9.2. Pre-Lecture Notes

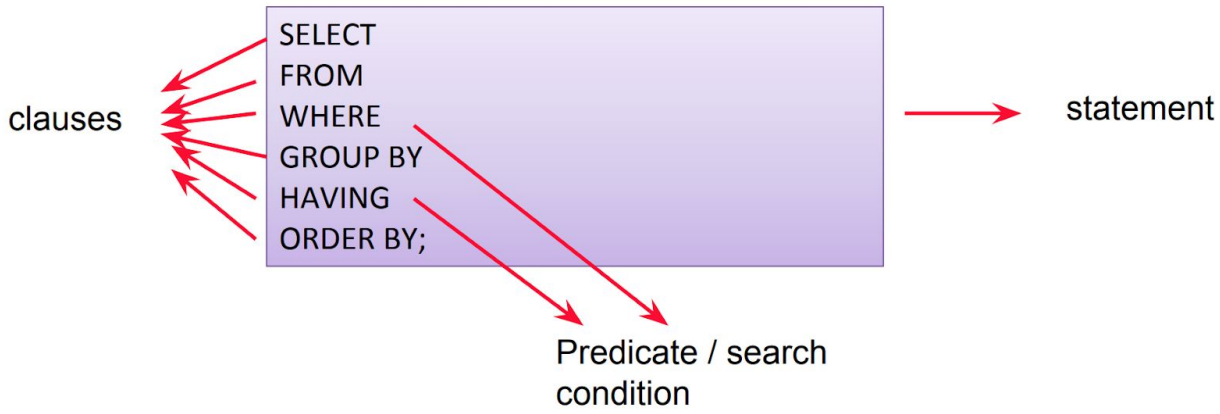
These are notes you may find useful to read through before the lecture, adapted from Lindsay Smith's lecture material. NOTE: THESE ARE NOT THE FINAL LECTURE SLIDES.

Read up the pre-lecture notes below.

IMPORTANT: AS THE LECTURE FOCUSES ON THE 'FLIPPED CLASSROOM' APPROACH - I.E. MORE INTERACTIVE DISCUSSION AND LESS ON DISCOVERING NEW MATERIAL - THE THEORY SLIDES BELOW ARE PROVIDED FOR YOUR READING CONVENIENCE BEFORE THE LECTURE.



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(mark)
FROM enrolment;
```

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

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

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

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;
```

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
HAVING count(*) > 2;
```

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

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)

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;
```

```
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 "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

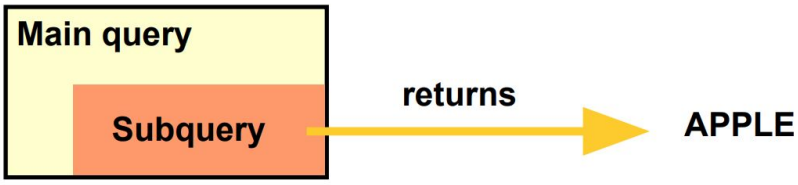
- Query within a query.

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

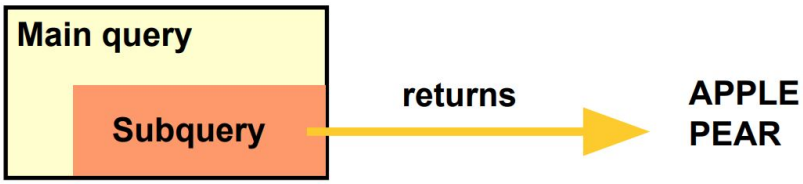
```
SELECT *
FROM enrolment
WHERE mark > (SELECT avg (mark)
                FROM enrolment );
```

Types of Subqueries

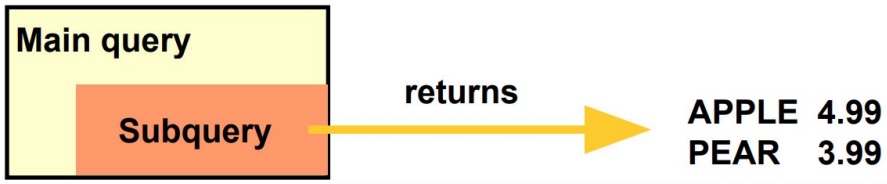
Single-value



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



Multiple-column subquery (many rows, many columns)



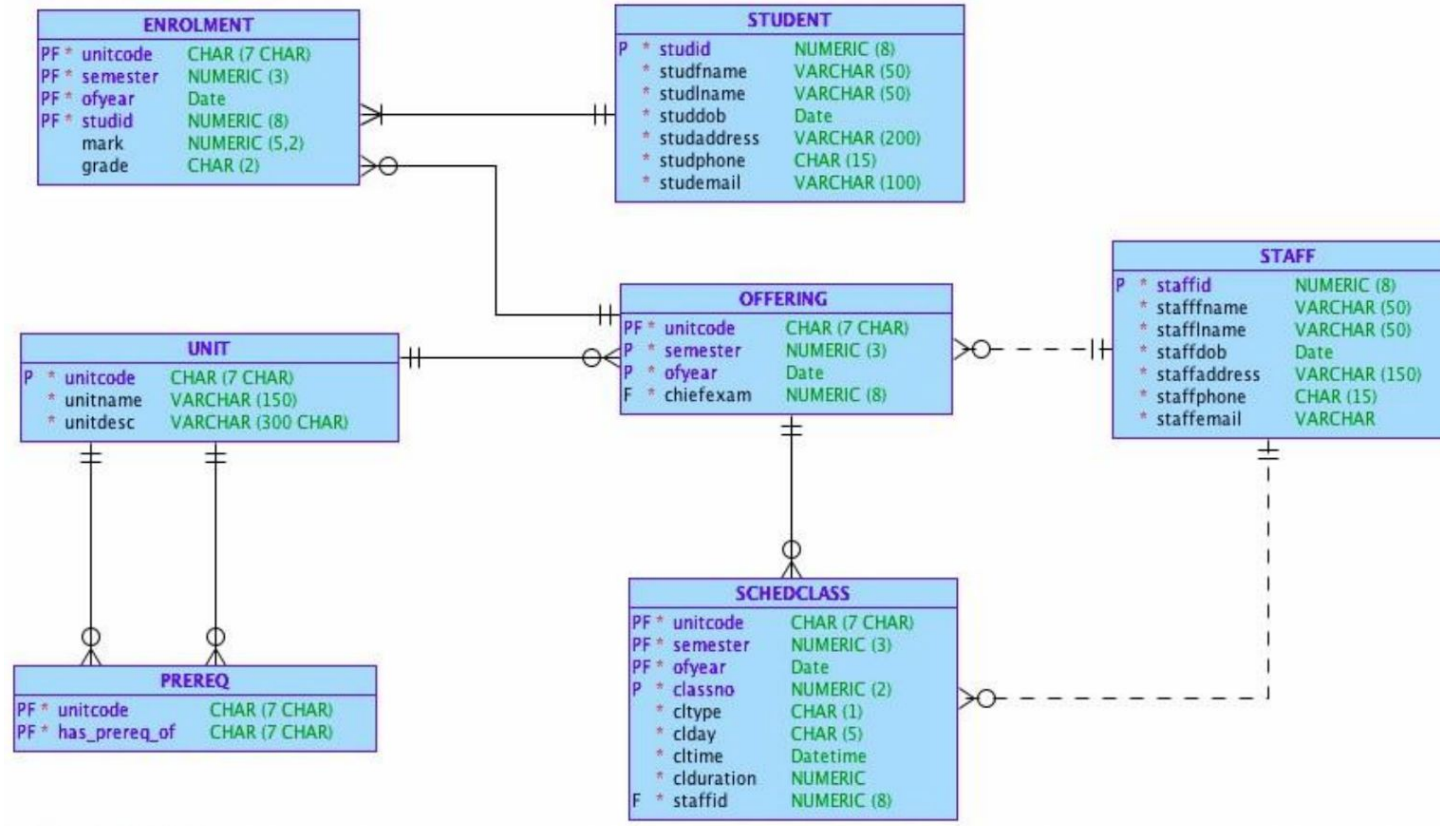
Comparison Operators for Subquery

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

Summary

- 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

Continued, PTO.

Selected questions to practice live in the actual Lecture:

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.

EOF.