

Database Systems (ISYS1001/ISYS5008)

Lecture 6

Subqueries , more on constraints

Updated: 01st September,2021

Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

CRICOS Provide Code: 00301J

Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf
of **Curtin University of Technology** pursuant to Part VB of the
Copyright Act 1968 (the Act)

The material in this communication may be subject to copyright under the
Act. Any further copying or communication of this material by you
may be the subject of copyright protection under the Act.

Do not remove this notice

Learning outcomes

- ▶ Create queries using subqueries , for comparing scalars, rows, columns or tables.
- ▶ Use subqueries with IN, ANY, ALL, EXISTS operators
- ▶ Use sub queries with UNION
- ▶ Use subqueries to create tables and do table manipulations
- ▶ Explain the importance of referential integrity constraints and policies to maintain them.
- ▶ Explain the difference between RESTRICT, CASCADE and SET-NULL policies related to referential integrity.
- ▶ Using MYSQL implement a relational schema with multiple tables, foreign key constraints and suitable referential integrity constraints.

Joining Two Relations – Story so far

- ▶ We've seen that it's possible to join two relations.
- ▶ We normally do this when a query requires information stored in more than one table.
- ▶ We JOIN the table by matching attributes up in some manner.
 - ▶ NATURAL JOIN – join common attributes
 - ▶ Theta JOIN – join based ON a condition
 - ▶ OUTER JOIN -

Story so far : Explicit tuple-variables

- ▶ Sometimes, a query needs to use two copies of the same relation.
- ▶ Distinguish copies by following the relation name by the name of a **tuple-variable**, in the FROM clause.
- ▶ It's always an option to rename relations this way, even when not essential.

Example : story so far

- From **Staff** (name, salary), find the name and salary of employees who are paid more than Jim.

```
SELECT b.name, b.salary
FROM Staff a, Staff b
WHERE a.name= 'Jim' AND a.salary < b.salary;
```

```
SELECT b.name, b.salary
FROM Staff a JOIN Staff b
ON a.salary < b.salary
WHERE a.name= 'Jim';
```

Staff a

Name	Salary
Jack	50000
Jim	45000
Kim	30000

Staff b

Name	Salary
Jack	50000
Jim	45000
Kim	30000

Subqueries or Subselects

- ▶ A query that is part of another query is called a subquery.
- ▶ Subqueries can have subqueries .. and so on.
- ▶ Subquery can be used in different ways:
 - ▶ Subquery returning a single constant (scalar) and this constant is compared with another value in a WHERE clause
 - ▶ Subquery can return a single column or a single row and used in FROM and WHERE clauses
 - ▶ Subquery can returns a relation that can be used in various ways in WHERE clauses
 - ▶ Example: in place of a relation in the FROM clause, we can place another query, and then query its result.
 - ▶ Better use a tuple-variable to name tuples of the result.
 - ▶ These are called scalar, column, row, and table subqueries.
 - ▶ They can be used in UNION, INTERSECTION, DIFFERENCE operations also.
- ▶ Subqueries that return a particular kind of result often can be used only in certain contexts
- ▶ Subquery is written as a parenthesized SELECT-FROM-WHERE statement.

Subqueries that return one Tuple

- ▶ If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
- ▶ Usually, the tuple has one component.
- ▶ A run-time error occurs if there is more than one tuple returned by the subquery.
- ▶ General form of predicates: $\text{expr } \theta \text{ (subquery)}$ where θ is one of $\{=, <>, >, >=, <, <=\}$.

Example: Subquery that produces a scalar

- ▶ From *Staff* (name, salary), find the name and salary of employees who are paid more than Jim.
- ▶ Earlier , a query was produced to answer the question as follows:

```
SELECT b.name, b.salary
FROM Staff a, Staff b
WHERE a.name = 'Jim' AND a.salary < b.salary;
```

- ▶ Same query can be answered with subquery approach:

1. First find the salary of Jim.
2. Then find the salaries greater than that.

```
SELECT b.name, b.salary
FROM Staff b
WHERE b.salary > (SELECT salary
                  FROM Staff
                  WHERE name= 'Jim');
```

Note: Any query with subqueries has to be analysed from the inside out.

Example: Subquery that produces a scalar

- ▶ Query with JOIN approach as created earlier:

```
SELECT b.name, b.salary
FROM Staff a JOIN Staff b
ON a.salary < b.salary
WHERE a.name= 'Jim';
```

Same query with subquery approach:

```
SELECT b.name, b.salary
FROM Staff b JOIN (SELECT salary
                    FROM Staff
                    WHERE name = 'jim') j
ON j.salary < b.salary;
```

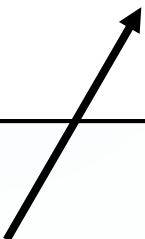
Example: Subquery that produces a scalar

11

- ▶ From *Taught* (unit, student, year, mark), find the students who took ISYS1001 in the same year as Joe.
- ▶ Two queries would surely work:
 1. Find the year Joe took ISYS1001.
 2. Find the students who took ISYS1001 in that year.

```
SELECT student
FROM Taught
WHERE unit = 'ISYS1001'
      AND year = (SELECT year
                  FROM Taught
                  WHERE student = 'Joe'
                  AND unit = 'ISYS1001' );
```

The year in
which Joe
took database
Systems



Query + Subquery Solution

Subqueries

- ▶ The main advantages of subqueries are:
 - ▶ They allow queries that are *structured* so that it is possible to isolate each part of a statement.
 - ▶ They provide alternative ways to perform operations that would otherwise require complex joins and unions.
 - ▶ Many people find subqueries more readable than complex joins or unions.

As per the MySQL documentation

(<https://dev.mysql.com/doc/refman/8.0/en/subqueries.html>)

Subqueries with conditions involving tuples

- ▶ Several operators in SQL can be applied to a relation and get a Boolean result.
- ▶ Example : IN, ANY, ALL, EXISTS
- ▶ These can be used with subqueries also.
- ▶ Example :
- ▶ In the expression `<tuple> IN <relation>`
 - ▶ `<relation>` can be a subquery.

Example: **IN** operator and subqueries

Using,

Writes(authorname, bookid, published),

Books(bookid, title)

find the title of each book that is written by Arthur Conan Doyle.

```
SELECT title
FROM Books
WHERE bookid IN (SELECT bookid
                  FROM Writes
                  WHERE authorname = 'A.C.Doyle');
```

ANY operator

- ▶ $x = ANY(<relation>)$ is a Boolean condition true if x equals at least one tuple in the relation.
- ▶ Similarly, $=$ can be replaced by any of the comparison operators.
- ▶ Example: $x > ANY(<relation>)$ means x is not the smallest tuple in the relation.
 - ▶ Note tuples must have one component only.

Example : **ANY** operator and subqueries

16

From

*Emp (empno, lastname, firstname, workdept, job, salary),
get name, dept and salary of non-managers who are paid
more than any manager.*

```
SELECT lastname, workdept, salary
FROM Emp
WHERE job <> 'MANAGER'
AND salary > ANY(SELECT salary
                  FROM Emp
                  WHERE job = 'MANAGER');
```


ALL operator

- ▶ Similarly, $x \neq \text{ALL}(\text{<relation>})$ is true if and only if for every tuple t in the relation, x is not equal to t .
 - ▶ That is, x is not a member of the relation.
- ▶ The \neq can be replaced by any comparison operator.
- ▶ Example: $x \geq \text{ALL}(\text{<relation>})$ means there is no tuple larger than x in the relation.

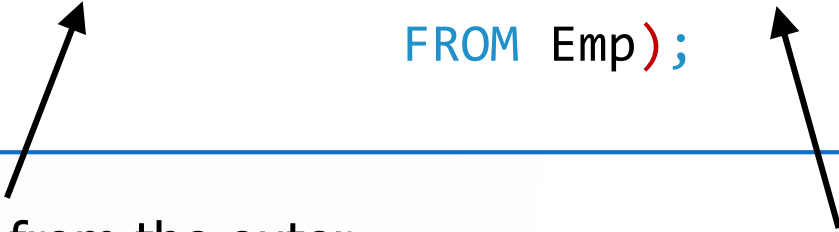
Example: **ALL** operator and subqueries

From

Emp (empno, lastname, firstname, workdept, job, salary),

find the empno and the last name of staff who get the highest salary.

```
SELECT empno, lastname
FROM Emp
WHERE salary >= ALL(SELECT salary
                     FROM Emp);
```



salary from the outer
Emp must not be
less than any salary.

Subquery resulted in a
single column

*Note: In practice ,
this query can be
answered much
easily with
MAX(salary); this
answer shows the
use of ALL*

Correlated subquery

- ▶ A simple subquery can be evaluated once and the result would be used in the higher-level query.
- ▶ In more complicated use of nested subqueries, the subquery may need to be evaluated many times, once for each assignment of a value to some term in the subquery that comes from a tuple variable outside the subquery.
- ▶ Such subqueries are called **correlated subqueries**.
- ▶ Correlated subquery: A subquery that uses data from an outer select.

Example: Correlated subquery

From *Emp* (*empno*, *lastname*, *firstname*, *workdept*, *salary*),

find the name and salary of employees paid the lowest salary in their departments.

Salary of each employee to be compared with the lowest salary of employees in the same dept.

```
SELECT lastname, workdept, salary
FROM Emp e1
WHERE salary <= ALL(SELECT salary
                     FROM Emp
                     WHERE workdept = e1.workdept);
```

empno	lastname	firstname	workdept	salary
120012	Jones	David	B01	67000
120045	Davies	Tom	B01	70000
120054	McDonald	Chris	D11	59000
120034	Smith	John	D11	63000

lastname	workdept	salary
Jones	B01	67000
McDonald	D11	59000

Example

- From *Enrolled*(unitID, studentID, year, mark), find unit and mark of each student who received the highest mark in the year he/she took the unit:

```
SELECT studentID, unitID, mark
FROM Enrolled t1
WHERE mark >= ALL(SELECT mark
                  FROM Enrolled
                  WHERE unitID = t1.unitID
                  AND year = t1.year);
```

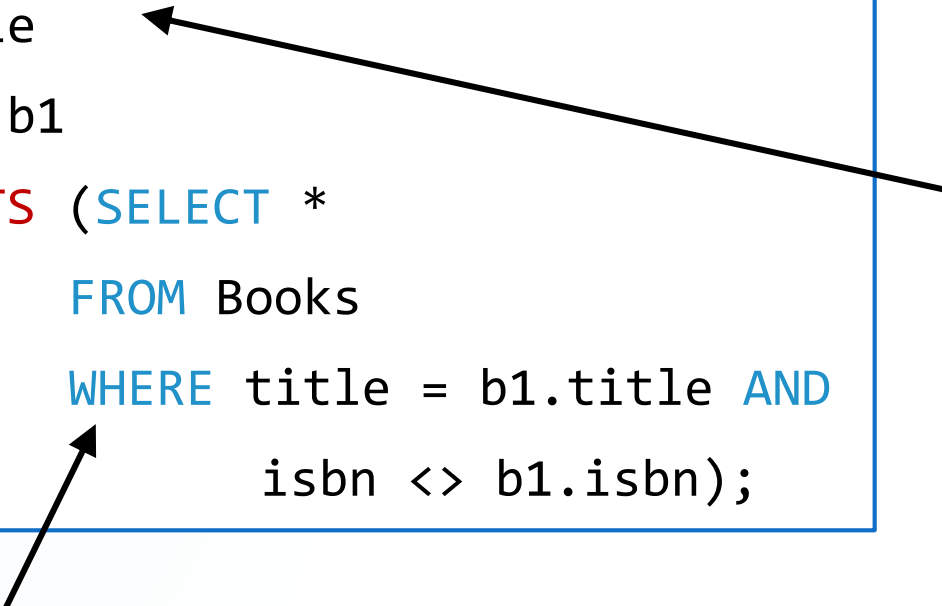
Exists operator

- ▶ **EXISTS** <relation> is a condition that is true if and only if <relation> is not empty.
- ▶ **EXISTS**(<subquery>) is true if and only if the subquery result is not empty.

Example: **Exists** operator and subqueries

- ▶ Example: From *Books(isbn, title)*, find the titles of all books that have more than one isbn (i.e., the books that have multiple editions with the same title)

```
SELECT title
FROM Books b1
WHERE EXISTS (SELECT *
              FROM Books
              WHERE title = b1.title AND
                    isbn <> b1.isbn);
```



*Notice scope rule:
title refers
to closest nested
FROM with
a relation having
that attribute.*

Set of Books with the same title
as b1, but not the same isbn

Union, Intersection, and Difference

- ▶ Union, intersection, and difference of relations are expressed in SQL by the following forms, each involving subqueries:
 - ▶ (<subquery>) UNION (<subquery>)
 - ▶ (<subquery>) INTERSECT (<subquery>)
 - ▶ (<subquery>) MINUS (<subquery>)
- ▶ *Note that MySQL does not support **INTERSECT** (use INNER JOIN) or **MINUS** (use NOT IN subquery).*

Example : UNION

```
(SELECT lastname, salary * 0.25 AS tax, 'tax at lower  
rate' AS remark  
FROM Emp  
WHERE Salary <= 65000)  
  
UNION  
  
(SELECT lastname, salary * 0.30 AS tax, 'tax at higher  
rate' AS remark  
FROM Emp  
WHERE Salary > 65000)  
  
ORDER BY tax DESC;
```

- ▶ Result of each SELECT must have the same number of columns.
- ▶ Corresponding entries of each select must be of compatible type.
- ▶ UNION eliminates duplicates, use UNION ALL to retain duplicates.
- ▶ If **ORDER BY** is present, it must be the last clause.

Example

LASTNAME	TAX	REMARK
-----	-----	-----
HAAS	21825	tax at higher rate
THOMPSON	15312	tax at lower rate
GEYER	15043	tax at lower rate
KWAN	14562	tax at lower rate
LUCCHESI	14125	tax at lower rate
PULASKI	14042	tax at lower rate
STERN	13888	tax at lower rate
LUTZ	12460	tax at lower rate
.....

Database Modifications

Database Modifications

- ▶ A *modification* command does not return a result (as a query does), but changes the database in some way.
- ▶ Three kinds of modifications:
 1. *Insert* a tuple or tuples.
 2. *Delete* a tuple or tuples.
 3. *Update* the value(s) of an existing tuple or tuples.
- ▶ We have already used some of these operations

Adding default values

29

- ▶ In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
- ▶ When an inserted tuple has no value for that attribute, the default will be used.
- ▶ Example:

```
CREATE TABLE Purchases(  
    name VARCHAR(30),  
    book CHAR(13),  
    when DATE,  
    cost DECIMAL(6,2)  
        DEFAULT 99.99,  
);
```

```
INSERT INTO  
    Purchasers(name,book)  
VALUES('Sally',9780786965625);
```

Resulting table:

name	book	when	cost
Sally	9780786965625	NULL	99.99

Inserting many tuples with a subquery

30

- ▶ We may insert the entire result of a query into a relation, using the form:

```
INSERT INTO <relation>  
    ( <subquery> );
```

- ▶ *Example:*

*From **Proj** (projno, projname, deptno, respemp, prstaff, stdate, endate), insert completed projects into Cproj.*

```
INSERT INTO Cproj  
SELECT *  
FROM Proj  
WHERE SYSDATE > endate;
```

Example: Deleting with a subquery

- ▶ *Task: Delete customers that haven't actually purchased books.*

```
DELETE FROM Customers c
WHERE NOT EXISTS
(SELECT *
 FROM Purchases
 WHERE name=c.name);
```

- ▶ Note use of correlation variable c.

Example: Update several tuples with a subquery

32

- *Task: Reduce the price of soft-cover books that no-one has purchased.*

```
UPDATE Softcovers
SET price = 0.75*price
WHERE isbn NOT IN
    (SELECT DISTINCT book
     FROM Purchases);
```


Creating tables from subqueries

- ▶ Saving the query result as a table

```
CREATE TABLE tablename  
[(columnname  
{, columnname})]  
AS subselect;
```

- ▶ Example: Create a table of all employees who are managers.

```
CREATE TABLE Deptmanager  
AS SELECT empno, lastname, firstname, edlevel,  
          deptno, deptname, salary, comm  
FROM Emp JOIN Dept  
ON mgrno=empno;
```

Changing Columns

34

- ▶ Add an attribute of relation R with :

```
ALTER TABLE R ADD <column declaration>;
```

- ▶ Example:

```
ALTER TABLE Emp  
ADD phone CHAR(16)  
    DEFAULT 'unlisted';
```

DEFAULT value
will be used
when no other
value is given
for phone.

- ▶ Columns may also be dropped.

```
ALTER TABLE R DROP <column declaration>;
```

- ▶ Example:

```
ALTER TABLE Emp DROP column comm;
```

More on constraints

Constraints

- ▶ A *constraint* is a relationship among data elements that the DBMS is required to enforce.
 - ▶ Example: key constraints.
- ▶ Types of constraints:
 - ▶ *Keys*.
 - ▶ *Foreign-key*, or referential-integrity.
 - ▶ *Value-based* constraints.
 - ▶ Constrain values of a particular attribute.
 - ▶ *Tuple-based* constraints.
 - ▶ Relationship among components.
- ▶ We have used some of these constraints earlier.

Sample relations

37

Student(studentID, firstName, lastName, country, courseID)

FK courseID REF Course(courseID)

Course(courseID, duration, dept)

Students

<u>studentID</u>	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	CS3UG
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	EG5PG

Courses

<u>courseID</u>	duration	dept
CS3UG	4	Computing
CS4UG	3	Computing
EG5PG	5	Engineering

Foreign key constraints and maintaining referential integrity

- ▶ What would happen if :
- ▶ We try to **insert** a new tuple to Students, whose courseID is not same as any courseIDs in the Courses table?
- ▶ We try to **update** a tuple in Students with a value not in courseIDs of the Courses table?
- ▶ We try to **delete** a tuple in Courses, which appears as a courseID in one or more Students tuples?
- ▶ We try to **update** a courseID in Courses table while there are courseIDs matching the old courseIDs in the Students table?

Foreign key constraints and maintaining referential integrity

- ▶ Above actions will be prevented by the DBMS to maintain the integrity of the database.
- ▶ We can specify the action taken when a violation occurs.
- ▶ There are three alternative ways to enforce it.
 1. *The Default policy*: **Reject** the modification; no change would be allowed if referential integrity violates. (This is the default, if no action is specified)
 2. *The cascade policy* (**CASCADE**) : Changes in the referenced attributes are similarly made in the foreign key. (changes in parent table reflects in the child table)
 3. *The set-NULL policy* (**SET NULL**) : When the value in the referenced attribute is modified, corresponding foreign key value is set to NULL.(when parent table changes, child table value set to NULL)

Note: **NO ACTION** : This is the same as RESTRICT

- ▶ Included for compatibility with SQL
- ▶ Is subtly different from RESTRICT in standard SQL

Foreign key constraints and maintaining referential integrity

- ▶ These options can be chosen for update and delete independently.
- ▶ The options are indicated with the declaration of the foreign key.

Example: Cascade

- ▶ Delete tuple with `couseID = CS4DU` from `Courses`:
 - ▶ Then delete all tuples from `Students` who are enrolled in the courses with `couseID CS4DU`.
- ▶ Update the `couseID = CS4DU` tuple of `Courses` to `IT4DU`
 - ▶ Then all the tuple in `Students` with `courseID = CS4DU` to `IT4DU`
- ▶ *Note that some MySQL servers (using NDB cluster) don't support CASCADE if the REFERENCES column is the parent table's primary key.*

Example: ON DELETE CASCADE

42

```
CREATE TABLE Courses(  
    courseName VARCHAR(30) PRIMARY KEY,  
    courseType VARCHAR(10)  
);  
  
CREATE TABLE Students(  
    studentID VARCHAR(8) PRIMARY KEY,  
    firstName VARCHAR(25) NOT NULL,  
    lastName VARCHAR(25),  
    courseName VARCHAR(30)  
    FOREIGN KEY (courseName) REFERENCES  
        Courses(courseName) ON DELETE CASCADE  
);
```

Example: ON DELETE SET NULL

```
CREATE TABLE Dept (  
    dno CHAR(3) PRIMARY KEY,  
    dname CHAR(15));  
  
CREATE TABLE Staff (  
    sid INTEGER PRIMARY KEY,  
    name CHAR(15),  
    workdept CHAR(3),  
    FOREIGN KEY(workdept) REFERENCES  
        Dept(dno) ON DELETE SET NULL  
);
```

- ▶ Example: When deleting the tuple of department D01 from Dept table, all tuples of Staff table with workdept = 'D01' will be changed to workdept = **NULL**.

Example: ON DELETE CASCADE and ON UPDATE RESTRICT

```
CREATE TABLE Enrolled (  
    unitID INTEGER,  
    studentID INTEGER,  
    mark REAL,  
    PRIMARY KEY (unitID, studentID),  
    FOREIGN KEY (unitID) REFERENCES Units(uID)  
        ON DELETE CASCADE ON UPDATE RESTRICT,  
    FOREIGN KEY (student) REFERENCES Students(sID)  
        ON DELETE CASCADE);
```

- ▶ Is it possible to implement SET NULL policy instead of ON DELETE CASCADE for unitID in Enrolled table?
- ▶ This is not possible.

Choosing a Policy

- ▶ When we declare a foreign key in MySQL, we may choose policies **SET NULL** or **CASCADE** for **deletions** and **updates**.
- ▶ In some situation (when using NDB data engine) on update **CASCADE** may not supported.
- ▶ Follow the foreign-key declaration by:
ON DELETE [SET NULL | CASCADE]
ON UPDATE [SET NULL | CASCADE]
- ▶ Otherwise, the default (**RESTRICT**) applies.

CHECK constraints

- ▶ SQL allows the use of the CHECK constraint for an attribute or tuple.
- ▶ Example:

```
CREATE TABLE Enrolled (  
    unit INT,  
    student INT,  
    mark REAL CHECK (mark <=100));
```

- ▶ Old versions of MySQL does not support CHECK constraints (New versions – v 8.0 .16 above supports CHECK constraints)

Summary

- ▶ A query that is part of another query is called a subquery.
- ▶ Subqueries can be used to get a scalar, row, column or a table.
- ▶ Subqueries can be combined with IN, ANY, ALL, EXISTS
- ▶ UNION is based on subqueries
- ▶ Subqueries can be used to create and modifying tables also.
- ▶ Referential integrity constraints can be added to table definitions to indicate how to handle foreign key violations.
- ▶ Referential integrity constraints can be set for update, delete independently.
- ▶ Default policy is not allowing modifications, if the referential integrity violates; other variations are CASCADE and SET NULL.

Happy Database systems

Next week : No lectures
(Tuition free week)

Week after the tuition free week : still there's no lecture as we will be having the mid term test

Next lecture will be on triggers

Practical worksheet 5 & 6 will be covered after the tuition free week but will be available during the tuition free week.