SQL: Nested Queries, Advanced Options, Updates, and Views

SWEN304 / SWEN435

Trimester 1, 2024

Lecturer: Kevin Shedlock

Engineering and Computer Science

VICTORIA UNIVERSITY OF WELLINGTON TE HERENGA WAKA

SWEN304/ Swen435 Course Noticeboard

1. Course Tutors are available at help desk labs in Room CO246. The days and times are:
   - Monday, 2-3pm
   - Friday 2-3pm

2. Assignment-1 is due in week 5

Outline

- Nested Queries
- Aggregate Functions
- Advanced options of the query language
  - Joined tables,
  - Aggregate functions
  - Grouping
- SQL views
- Additional features of SQL

Nested Queries

- Some queries require comparing a tuple to a collection of tuples (e.g., *students doing courses that have more than 100 students*)
- This task can be accomplished by embedding a SQL query into `WHERE` clause of another query
  - The embedded query is called **nested query,**
  - The query containing the nested query is called **outer query**
- The comparison is made by using `IN`, θ `ANY`, θ `SOME`, and θ `ALL` operators, where θ∈{ =, <, <=, >=, >, < > }
- Note: `IN` ⇔ `=ANY` and `IN` ⇔ `=SOME`

Example Nested Query

- *Retrieve first names of students that passed M214*

```
SELECT FName
FROM STUDENT s
WHERE s.StudId  IN
    (SELECT StudId  FROM GRADES
     WHERE CourId = 'M214' AND Grade IS NOT
     NULL);
```

FName
James

Example Nested Query

- A nested query defined by using `IN` (or `=ANY`) operator can be expressed as a single block query

```
SELECT FName
FROM STUDENT s, GRADES g
WHERE s.StudId = g.StudId  AND g.CourId = 'M214'
AND g.Grade IS NOT NULL;
```

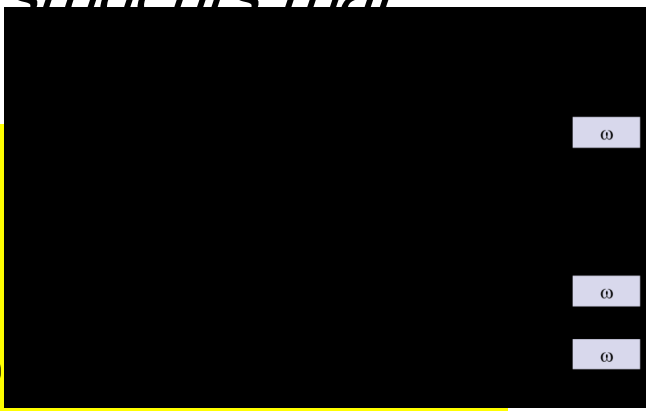
Correlated Nested Queries

- Let the variable *s* contain the current tuple of the outer query
- If the nested query doesn't refer to *s* :
  - The nested query computes the same result for each tuple in *s*
  - The outer query and the nested query are said to be **uncorrelated**
- If a condition in the `WHERE` clause of the nested query refers to some attributes of a relation declared in the outer query, the two queries are said to be **correlated**
  - Have to compute the inner query for each tuple considered by the outer query
  - Correlated nested queries consume more computer time than uncorrelated ones

Correlated Nested Query

- *Retrieve id's and surnames of those students that passed at least one course*

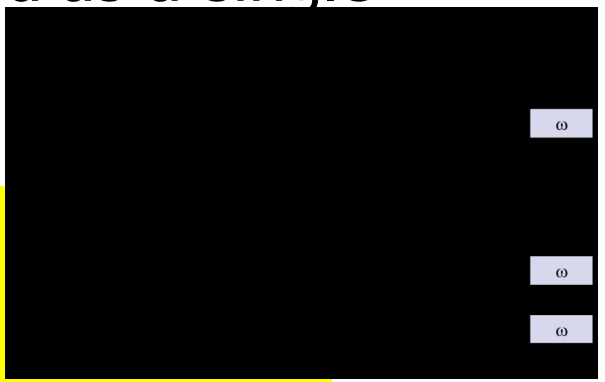
```
SELECT s.StudId, FName
FROM Student s
WHERE s.StudId  IN
    (SELECT StudId  FROM GRAD
     WHERE s.StudId = StudId AND
                Grade IS NOT NULL);
```



- Evaluation of the query:
  - when s.Stud Id = 131313,
    ⇒ result of the nested query is StudId = {131313},
    ⇒ (131313, Susan) is in the final result
  - When s.Stud Id = 010101,
    ⇒ result of the nested query is StudId = { },
    ⇒ (010101, John) is NOT in the final result

Correlated Nested Query

- Again, the nested query can be expressed as a single block query:



```
SELECT DISTINCT s.StudId, s.LName
FROM STUDENT s, Grades g
WHERE s.StudId = g.StudId AND Grade IS NOT
NULL;
```

- Have to be careful of duplicates!
- This computes an Equi-Join of the relations

EXISTS and NOT EXISTS

- EXIST and NOT EXIST are used in conjunction with correlated nested queries
- *Retrieve Id's and surnames of students who passed at least one course:*

```
SELECT s.StudId, s.LName FROM STUDENT s
WHERE EXISTS
     (SELECT *  FROM GRADES
      WHERE s.StudId = StudId AND Grade IS NOT NULL);
```

EXISTS and NOT EXISTS

- *Retrieve Id's and surnames of students who didn't pass any course:*

```
SELECT s.StudId, s.LName FROM STUDENT s
WHERE NOT EXISTS
     (SELECT *  FROM GRADES
      WHERE s.StudId = StudId AND Grade IS NOT NULL );
```

Summary (Including Last Friday)

- SQL as DML: INSERT, UPDATE and DELETE
- SQL as a query language
  - Basic Query structure
    - Queries against a single table
    - Queries against multiple tables
    - Substring comparisons
    - Arithmetic operations
    - Sorting
  - Nested queries (outer and inner-nested queries)
    - Correlated nested queries

Joined Tables in SQL and Outer Joins

- **Joined table**
  - Permits users to specify a table resulting from a join operation in the FROM clause of a query

- To avoid mixing conditional expressions and join conditions in the `WHERE` clause, it is possible to define join in the `FROM` clause

UNIVERSITY Database

UNIVERSITY ={STUDENT(StudId, Lname, Fname, Major),
COURSE(CourId, Cname, Points, Dept),
GRADES(StudId, CourId, Grade)}

IC = {GRADES[Id] $\subseteq$ STUDENT[Id],
GRADES[Course_id] $\subseteq$ COURSE[Course_id]}

STUDENT			
StudId	**Lname**	**Fname**	**Major**
300111	Smith	Susan	COMP
300121	Bond	James	MATH
300143	Bond	Jenny	MATH
300132	Smith	Susan	COMP

COURSE			
CourId	**Cname**	**Points**	**Dept**
COMP302	DB sys	15	Engineering
COMP301	softEng	20	Engineering
COMP201	Pr & Sys	22	Engineering
MATH214	DisMat	15	Mathematics

GRADES		
StudId	**CourId**	**Grade**
300111	COMP302	A+
300111	COMP301	A
300111	MATH214	A
300121	COMP301	B
300132	COMP301	C
300121	COMP302	B+
300143	COMP201	ω
300132	COMP201	ω
300132	COMP302	C+

Joined Tables in SQL

- Q1: Retrieve first name, course id and corresponding grades of the student with Student Id = 007007

```
SELECT FName, CourId, Grade
FROM (STUDENT NATURAL JOIN GRADES )
WHERE StudId = 007007 ;
```

- The `FROM` clause contains a single joined table

Joined Tables in SQL

- Specify different types of joins
  - Inner Joins:
    - JOIN, INNER JOIN, EQUIJOIN, NATURAL JOIN,
  - Outer Joins:
    - LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN
    - The keyword OUTER may be omitted
    - CROSS JOIN is used to specify the CARTESIAN PRODUCT operation and should be used only with the utmost care

- Each join operation concatenates those tuples from two relations that have such join attribute values and satisfy the JOIN condition.

Joined Tables in SQL

- **Inner join**
  - Default type of join in a joined table
  - Tuple is included in the result only if a matching tuple exists in the other relation

- **Outer join** has been defined to overcome problems with null values and tuples of the referenced table not being referenced

Inner Joins

- Two relation can be joined with join conditions, comparing pairs of attribute values using operations, $\theta \in \{ =, <, <=, >, >=, <> \}$
- An equijoin is a join using only equality operator
- A NATURAL JOIN on two relations R and S
  - No join condition specified
  - Implicit EQUIJOIN condition for each pair of attributes with same name from R and S
  - Each pair of attributes appears only once

JOIN

- Using `JOIN` operator

```
SELECT * FROM R1 JOIN R2 ON R1.B = R2.B;
```

R1

A	B
a_1	ω
a_2	b_1

R2

B	C
b_1	c_1
b_2	c_1
b_3	c_1

R1 JOIN R2

A	B	B	C
a_2	b_1	b_1	c_1

- Using `NATURAL JOIN` operator

```
SELECT * FROM R1 NATURAL JOIN R2;
```

A	B	C
a_2	b_1	c_1

LEFT OUTER JOIN

- Every tuple in left table must appear in result
- If no matching tuple
  - Padded with NULL values for attributes of right table

```
SELECT * FROM R1 LEFT JOIN R2 ON R1.B = R2.B;
```

R1

A	B
a_1	ω
a_2	b_1

R2

B	C
b_1	c_1
b_2	c_1
b_3	c_1

R1 LEFT JOIN R2

A	B	B	C
a_1	ω	ω	ω
a_2	b_1	b_1	c_1

RIGHT OUTER JOIN

- Every tuple in right table must appear in result
- If no matching tuple
  - Padded with NULL values for attributes of right table

```
SELECT * FROM R1 RIGHT JOIN R2 ON R1.B = R2.B;
```

R1

A	B
a_1	ω
a_2	b_1

R2

B	C
b_1	c_1
b_2	c_1
b_3	c_1

R1 RIGHT JOIN R2

A	B	B	C
a_2	b_1	b_1	c_1
ω	ω	b_2	c_1
ω	ω	b_3	c_1

FULL OUTER JOIN

- All tuples from both relations must appear in result

```
SELECT * FROM R1 FULL JOIN R2 ON R1.B = R2.B;
```

R1

A	B
a_1	ω
a_2	b_1

R2

B	C
b_1	c_1
b_2	c_1
b_3	c_1

R1 RIGHT JOIN R2

A	B	B	C
a_1	ω	ω	ω
a_2	b_1	b_1	c_1
ω	ω	b_2	c_1
ω	ω	b_3	c_1

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- **Grouping**
  - Create subgroups of tuples before summarizing
- Built-in aggregate functions
  - `COUNT`, `SUM`, `MAX`, `MIN`, and `AVG`
- Functions can be used in the `SELECT` clause or in a `HAVING` clause

Aggregate Functions: Examples

Q2: What is the total point value of the courses in the COMP dept?

```
SELECT SUM(Points)
FROM COURSE
WHERE Dept = 'Comp';
```

sum_Points
52

Q3: How many different first names do COMP majors have?

```
SELECT COUNT(DISTINCT FName) AS NoOfNames
FROM STUDENT
WHERE Major = 'Comp';
```

NoOfNames
1

Grouping

- `GROUP BY` clause create groups of tuples used to apply an aggregate function to
- Groups are determined by means of a grouping attribute list and all attributes from that list have to appear in the query result (i.e. appear in the `SELECT` clause)
- *Example:*

  For each student, retrieve the number of courses passed

```
SELECT StudId, COUNT(*)
FROM GRADES
WHERE Grade IS NOT NULL
GROUP BY StudId ;
```

StudId	COUNT(*)
300111	3
300121	2
300132	2

HAVING Clause

- HAVING clause is used to choose from groups according to a condition specified on aggregate function values
- Whereas a conditional expression in the WHERE clause filters individual tuples, the HAVING clause filters groups of tuples
- *Example:*

  Retrieve the number of courses passed for students that passed at least three courses

```
SELECT StudId, COUNT(*)
FROM STUDENT s NATURAL JOIN GRADES g
WHERE Grades IS NOT NULL
GROUP BY s.StudId
HAVING COUNT(*) > 2;
```

StudId	COUNT(*)
300111	3

Summary of SQL queries

- An SQL query can consist of up to six clauses
- but only SELECT and FROM are mandatory
- the HAVING-clause can only be specified with a GROUP BY-clause
- The clauses are specified in the following order:

```
SELECT  ⟨attribute_and_function_list⟩
FROM  ⟨table_list⟩
[WHERE  ⟨condition⟩ ]
[GROUP BY  ⟨grouping_attribute_list⟩ ]
[HAVING  ⟨group_condition⟩ ]
[ORDER BY  ⟨attribute_list⟩ ]
```

Set Theoretic Operations

- SQL has directly implemented **set operations**
  - `UNION`, `EXCEPT` (difference), and `INTERSECT`
- Operations on union compatible relations (same attributes, in the same order), results sets of tuples; (demo)
- e.g.
  - *Retrieve student ids of the students that didn't enroll in M214*

```
(SELECT StudId
 FROM STUDENT )
EXCEPT
(SELECT StudId FROM GRADES
 WHERE CourId  = 'M214');
```

StudId
300132
300121
300143

Queries

- *Query: Retrieve student ids of the students who got A+ for **all** the grades she/he achieved so far*

- *Query: Retrieve student ids of the students who has never got A+ so far*

GRADES		
StudId	**CourId**	**Grade**
300111	COMP302	A+
300111	COMP301	A
300111	MATH214	A
300121	COMP301	B
300132	COMP301	C
300121	COMP302	B+
300143	COMP201	ω
300132	COMP201	ω
300132	COMP302	C+

Views in SQL

- A SQL view is a virtual table that is derived from other base or virtual tables
- Base tables are defined by `CREATE TABLE` command and are permanently stored in a database
- Virtual tables are defined by the `CREATE VIEW` command to avoid defining complex SQL retrieval expressions repeatedly
- The definition of a view is stored in the Catalog, but it is not stored in the database itself, so it is computed every time it is used in a query

SQL Views

- A possible view definition

```
CREATE VIEW StudOccupied AS
    SELECT g.StudId, SUM(Hours ) AS Occupied
    FROM Grades g, Course p
    WHERE g.CourId = p.CourId AND Grade IS NULL
    GROUP BY StudId ;
```

- Deleting a view

```
DROP VIEW StudOccupied;
```

Additional Features of SQL

- **Assertions** as general constraints (`CREATE ASSERTION` – a DDL command that may use DML SELECT command)
- **Triggers** as procedures stored with tables
- `GRANT` and `REVOKE` commands to deal with database user privileges
- **Embedded** SQL and `CURSOR`
- SQL transaction control commands (`COMMIT`, `ROLLBACK`)
- User Defined Functions (UDF):
  - SQL Functions
  - Procedural Language (C, PL/pgSQL, Java) Functions

Summary

- The relational database language has commands to define:
  - database structure (schema, domain, table, and constraints) (`CREATE SCHEMA, CREATE DOMAIN, CREATE TABLE`)
  - queries (`SELECT… FROM… WHERE… GROUP BY…HAVING… ORDER BY…`)
  - update operations (`INSERT, DELETE, UPDATE`)
  - views (`CREATE VIEW`)
  - additional features (`ASSERTION, TRIGGER, CURSOR, GRANT, REVOKE, COMMIT, ROLLBACK, DEFINE FUNCTION`)
- SQL is defined by a standard, with implementations that have some dialects and exceptions

Next Lecture

- SQL tutorial

- And then...
  - Relational Algebra with Hui! (Bye ☹)
  - Chapter 6