



Database

Lesson 5. Structured Query Language – part 2

Viet-Trung Tran
<https://is.hust.edu.vn/~trungtv/>

Learning Map

| Sequence | Title |
|----------|------------------------------------|
| 1 | Introduction to Databases |
| 2 | Relational Databases |
| 3 | Relational Algebra |
| 4 | Structured Query Language – Part 1 |
| 5 | Structured Query Language – Part 2 |
| 6 | Constraints and Triggers |
| 7 | Entity Relationship Model |
| 8 | Functional Dependency |
| 9 | Normalization |
| 10 | Storage - Indexing |
| 11 | Query Processing |
| 12 | Transaction Management – Part 1 |
| 13 | Transaction Management – Part 2 |

Outline

1. Data Manipulation: SQL Retrieval statement (Part 2)
2. View
3. Privileges and User Management in SQL

Learning objective

- Write **retrieval statement** in SQL: from simple queries to complex ones
- Create **views** and work correctly on predefined views
- Have experience with a DBMS: **manage user account and database access permissions**

Keywords

| Keyword | Description |
|------------|---|
| Query | A request (SQL statement) for information from a database |
| Subquery | A subquery (inner query, nested query) is a query within another (SQL) query. |
| Privileges | Database access permissions |
| View | A view is the result set of a stored query on the data, which the database users can query just as they would in a persistent database collection object. |

Data Manipulation: SQL Retrieval statement (Part 2)

1. Joins operators
2. Subqueries: in FROM clause and in WHERE clause
3. Union, Intersection and Difference of Queries
4. Aggregation operators
5. Grouping and aggregation in SQL , conditions in HAVING clause
6. Controlling the output: duplicate elimination, ordering the result

1. Example of a database schema

student(**student_id**, first_name, last_name, dob, gender, address, note, *clazz_id*)
clazz(**clazz_id**, name, *lecturer_id*, *monitor_id*)
subject(**subject_id**, name, credit, percentage_final_exam)
enrollment(**student id**, **subject id**, **semester**, midterm_score, final_score)
lecturer(**lecturer id**, first_name, last_name, dob, gender, address, email)
teaching(**subject id**, **lecturer id**)
grade(**code**, from_score, to_score)



Client-applications
(in C#, Java, php, ...)

List of all female students ?

First name, last name and address of class monitors ?

**List of students (id and fullname) have
enrolled subject 'Học máy' in semester 20172?**

List of students (id and fullname) having CPA >= 3.2?



1. Example of a database schema

student

| student_id | first_name | last_name | dob | gender | ... |
|------------|------------|-----------|-----------|--------|-----|
| 20160001 | Ngọc An | Bùi | 3/18/1987 | M | ... |
| ... | ... | ... | ... | ... | ... |
| 20160003 | Thu Hồng | Trần | 6/6/1987 | F | ... |
| 20160004 | Minh Anh | Nguyễn | 5/20/1987 | F | ... |

List of students (id and fullname) have enrolled subject 'Học máy' in semester 20172?

enrollment

| student_id | subject_id | semester | midterm_score | final_score |
|------------|------------|----------|---------------|-------------|
| 20160001 | IT1110 | 20171 | 9 | 8.5 |
| ... | ... | ... | ... | ... |
| 20160001 | IT4866 | 20172 | 7 | 9 |
| 20160002 | IT3080 | 20172 | 9 | |
| 20160003 | IT4866 | 20172 | 7 | 6 |

subject

| subject_id | name | credit | percentage_final_exam |
|------------|-------------------|--------|-----------------------|
| IT1110 | Tin học đại cương | 4 | 60 |
| ... | ... | ... | ... |
| IT4866 | Học máy | 2 | 70 |

1. Data Manipulation: SELECT operation

```
SELECT [all|distinct]
      { * | {table_name.*|expr[alias]} | view_name.* }
      [, {table_name.*|expr[alias]}] ... }
FROM table_name [alias] [,table_name[alias]] ...
[WHERE condition]
[GROUP BY expr [,expr] ...]
[HAVING condition]
[ {UNION|UNION ALL|INTERSECT|MINUS}
    SELECT ... ]
[ORDER BY {expr|position} [ASC|DESC]
[,expr|position] [ASC|DESC]
```

Data Manipulation: Advanced SELECT

- Joins operators
- Subqueries: in FROM clause and in WHERE clause
- Aggregation operators
- Grouping and aggregation in SQL , conditions in HAVING clause
- Controlling the output: duplicate elimination, ordering the result

1.1. Joins operators

- Syntax:

```
SELECT t1.c1, t1.c2, ..., t2.c1, t2.c2  
FROM t1, t2  
WHERE condition_expression
```

- Example:

student(student_id, first_name, last_name, dob, gender, address, note, *clazz_id*)
clazz(clazz_id, name, *lecturer_id*, *monitor_id*)

```
SELECT clazz.clazz_id, name, last_name, first_name  
FROM clazz, student  
WHERE student_id = monitor_id
```

1.1. Joins operators: Operational semantics

clazz

| clazz_id | name | lecturer_id | monitor_id |
|----------|--------------|-------------|------------|
| 20162101 | CNTT1.01-K61 | 02001 | 20160003 |
| 20162102 | CNTT1.02-K61 | | |
| 20172201 | CNTT2.01-K62 | 02002 | 20170001 |
| 20172202 | CNTT2.02-K62 | | |

student

| student_id | first_name | last_name | ... | clazz_id |
|------------|------------|-----------|-----|----------|
| 20160001 | Ngọc An | Bùi | | |
| 20160002 | Anh | Hoàng | | 20162101 |
| 20160003 | Thu Hồng | Trần | | 20162101 |
| 20160004 | Minh Anh | Nguyễn | | 20162101 |
| 20170001 | Nhật Ánh | Nguyễn | | 20172201 |

List of classes with monitor names
(firstname, lastname) :

SELECT clazz.clazz_id, name,
student.last_name,
student.first_name

3

FROM clazz, student

1

WHERE student_id = monitor_id

2

result

| clazz_id | name | last_name | first_name |
|----------|--------------|-----------|------------|
| 20162101 | CNTT1.01-K61 | Trần | Thu Hồng |
| 20172201 | CNTT2.01-K62 | Nguyễn | Nhật Ánh |

Tuple-variables loop over all tuples of each relation in FROM clause

1.1. Joins operators: AS keyword in FROM clause

- Used for naming variables:

```
SELECT ...
FROM  <table_name> [AS] <variable_name>, ...
[WHERE ...]


- AS: optional,
- <variable_name>: used in the whole SQL statement

```

- Example:

```
SELECT c.clazz_id, name, s.last_name, s.first_name
FROM clazz AS c, student s
WHERE s.student_id = c.monitor_id
```

1.1. Joins operators: Self-join

subject(subject_id, name, credit, percentage_final_exam)

Find all pairs of subjects id having the *same name* but the credit of the first subject is less than the credit of the second one

```
SELECT sj1.subject_id, sj2. subject_id
FROM subject sj1, subject sj2
WHERE sj1.name = sj2.name
      AND sj1.credit < sj2.credit
```

1.1. Joins operators: Example

```
student(student_id, first_name, last_name, dob, gender, address, note, clazz_id)
subject(subject_id, name, credit, percentage_final_exam)
enrollment(student_id, subject_id, semester, midterm_score, final_score)
```

List of students have enrolled subjects in semester 20172. The list composes of student fullname, subject name, subject credit:

```
SELECT last_name || ' ' || first_name as fullname,
       sj.name as subjectname, credit
FROM student s, enrollment e, subject sj
WHERE s.student_id = e.student_id
       AND sj.subject_id = e.subject_id
       AND semester = '20172'
```

1.1. Joins operators: Join types

- Product:
 - R **CROSS JOIN** S
- Theta join:
 - R [**INNER**] **JOIN** S **ON** <condition>
- Natural join: (**Be careful!**)
 - R **NATURAL JOIN** S
- Outer join:
 - R [**LEFT|RIGHT|FULL**] [**OUTER**] **JOIN** S **ON** <condition>
 - R **NATURAL [LEFT|RIGHT|FULL] [OUTER]** **JOIN** S

1.1. Joins operators: OUTER JOINS

- R [LEFT|RIGHT|FULL] OUTER JOIN S ON <condition>
- R NATURAL [LEFT|RIGHT|FULL] OUTER JOIN S

R

| a | b | c |
|---|-------|---|
| 1 | An | 5 |
| 2 | Binh | 5 |
| 3 | Cuong | 7 |

S

| a | c | d |
|---|---|---|
| 1 | 5 | X |
| 1 | 7 | Y |
| 2 | 5 | Z |
| 4 | 1 | Z |

R FULL OUTER JOIN S ON (R.a = S.a)

| R.a | b | R.c | S.a | S.c | d |
|------|-------|------|------|------|------|
| 1 | An | 5 | 1 | 5 | X |
| 1 | An | 5 | 1 | 7 | Y |
| 2 | Binh | 5 | 2 | 5 | Z |
| 3 | Cuong | 7 | NULL | NULL | NULL |
| NULL | NULL | NULL | 4 | 1 | Z |

R NATURAL LEFT OUTER JOIN S

| R.a | b | R.c | S.a | S.c | d |
|-----|-------|-----|------|------|------|
| 1 | An | 5 | 1 | 5 | X |
| 2 | Binh | 5 | 2 | 5 | Z |
| 3 | Cuong | 7 | NULL | NULL | NULL |

1.1. Joins operators: OUTER JOIN Example

- List of all classes with monitor names (firstname and lastname, NULL if class has not yet a monitor)

clazz

| clazz_id | name | lecturer_id | monitor_id |
|----------|--------------|-------------|------------|
| 20162101 | CNTT1.01-K61 | 02001 | 20160003 |
| 20162102 | CNTT1.02-K61 | | |
| 20172201 | CNTT2.01-K62 | 02002 | 20170001 |
| 20172202 | CNTT2.02-K62 | | |

student

| student_id | first_name | last_name | ... | clazz_id |
|------------|------------|-----------|-----|----------|
| 20160003 | Thu Hồng | Trần | ... | 20162101 |
| 20160004 | Minh Anh | Nguyễn | ... | 20162101 |
| ... | ... | ... | ... | ... |

result

| clazz_id | name | last_name | first_name |
|----------|--------------|-----------|------------|
| 20172202 | CNTT2.02-K62 | NULL | NULL |
| 20162102 | CNTT1.02-K61 | NULL | NULL |
| 20162101 | CNTT1.01-K61 | Trần | Thu Hồng |
| 20172201 | CNTT2.01-K62 | Nguyễn | Nhật Anh |

```
SELECT c.clazz_id, name, last_name, first_name
FROM clazz c LEFT OUTER JOIN student
    ON (student_id = monitor_id);
```

1.2. Sub-queries

- A SELECT-FROM-WHERE statement can be used within a clause of another outer query. It can be
 - within a **WHERE** clause
 - within a **FROM** clause
- Creates an intermediate result
- No limit to the number of levels of nesting
- Objectives:
 - Check if an element is in a set (**IN, NOT IN**)
 - Set comparison **>ALL, >=ALL, <ALL, <=ALL, =ALL, ANY (SOME)**
 - Check if a relation is empty or not (**EXISTS, NOT EXISTS**)

1.2. Sub-queries: Subquery returns scalar value

- A sub-query provide a single value → we can use it as if it were a constant

```
SELECT *
FROM student
WHERE clazz_id = (SELECT clazz_id
                   FROM clazz
                   WHERE name = 'CNTT1.01-K61');
```

1.2. Sub-queries: IN operators

- Syntax:

<tuple> [NOT] IN <subquery>

- Example: First name, last name and address of class monitors?

student(student_id, first_name, last_name, dob, gender, address, note, *clazz_id*)
clazz(clazz_id, name, *lecturer_id*, *monitor_id*)

```
SELECT first_name, last_name, address
FROM student
WHERE student_id IN (SELECT monitor_id FROM clazz);
```

1.2. Sub-queries: EXISTS

- Syntax:
[NOT] EXISTS (<subquery>
EXISTS (<subquery>): TRUE iff <subquery> result is **not empty**
- Example: subjects having no lecturer?

```
teaching(subject_id, lecturer_id)
subject(subject_id, name, credit, percentage_final_exam)
SELECT * FROM subject s
WHERE not exists (SELECT *
                   FROM teaching
                   WHERE subject_id = s.subject_id)
```

1.2. Sub-queries: ALL, ANY

- Syntax: <expression> <*comparison_operator*> **ALL|ANY** <subquery>
 - <*comparison_operator*>: **>**, **<**, **<=**, **>=**, **=**, **<>**
 - X **>=ALL**<subquery>: TRUE if there is **no tuple larger than X** in <subquery> result
 - X **= ANY**<subquery>: TRUE if **x equals at least one tuple** in <subquery> result
 - X **>ANY**<subquery>: TRUE if **x is not the smallest tuple** produced by <subquery>
- Example:

```
SELECT *
FROM subject
WHERE credit >= ALL (SELECT credit FROM subject);
```

1.2. Sub-queries: Example

subject

| subject_id | name | credit | perc... |
|------------|-------------------|--------|---------|
| IT1110 | Tin học đại cương | 4 | 60 |
| IT3080 | Mạng máy tính | 3 | 70 |
| IT3090 | Cơ sở dữ liệu | 3 | 70 |
| IT4857 | Thị giác máy tính | 3 | 60 |
| IT4866 | Học máy | 2 | 70 |

```
SELECT *
FROM subject
WHERE credit >= ALL (SELECT credit FROM subject);
```

result

```
SELECT *
FROM subject
WHERE credit > ANY (SELECT credit
                      FROM subject);
```

| subject_id | name | credit | perc... |
|------------|-------------------|--------|---------|
| IT1110 | Tin học đại cương | 4 | 60 |
| IT3080 | Mạng máy tính | 3 | 70 |
| IT3090 | Cơ sở dữ liệu | 3 | 70 |
| IT4857 | Thị giác máy tính | 3 | 60 |

result

| subject_id | name | credit | perc... |
|------------|-------------------|--------|---------|
| IT1110 | Tin học đại cương | 4 | 60 |

1.2. Sub-queries: Subquery in FROM Clause

- Subquery is used as a relation in a FROM clause
- Must give it a **tuple-variable alias**
- Eg.: List of lecturers teaching subject whose id is 'IT3090'

```
SELECT l.*  
FROM lecturer l,  
      (SELECT lecturer_id  
       FROM teaching  
       WHERE subject_id = 'IT3090') lid  
WHERE l.lecturer_id = lid.lecturer_id
```

1.3. Union, Intersection and Difference of Queries

- <subquery_1> **UNION** <subquery_2>
- <subquery_1> **INTERSECT** <subquery_2>
- <subquery_1> **EXCEPT** <subquery_2>
- Ex.: List of subjects having any enrollment?

```
SELECT * FROM subject  
EXCEPT  
SELECT s.*  
FROM subject s NATURAL JOIN enrollment e ;
```

1.4. Aggregation Operators

- SUM, AVG, COUNT, MIN, MAX: applied to a column in a SELECT clause
- COUNT(*) counts the number of tuples

```
SELECT AVG(credit), MAX(credit)
FROM subject
WHERE subject_id LIKE 'IT%';
```

| result | |
|--------|-----|
| AVG | MAX |
| 3.0 | 4 |

| subject | | | |
|------------|----------------------|--------|---------|
| subject_id | name | credit | perc... |
| IT1110 | Tin học đại cương | 4 | 60 |
| IT3080 | Mạng máy tính | 3 | 70 |
| IT3090 | Cơ sở dữ liệu | 3 | 70 |
| IT4857 | Thị giác máy tính | 3 | 60 |
| IT4866 | Học máy | 2 | 70 |
| LI0001 | life's happy song | 5 | |
| LI0002 | %life's happy song 2 | 5 | |

1.4. Aggregation Operators: Functions

- **Aggregate functions:** MAX, MIN, SUM, AVG, COUNT
- Functions applying on **individual tuples:**
 - Mathematic functions: ABS, SQRT, LOG, EXP, SIGN, ROUND, ..
 - String functions: LEN, LEFT, RIGHT, MID,...
 - Date/Time functions: DATE, DAY, MONTH, YEAR, HOUR, MINUTE, ...
 - Format modification: FORMAT
 - Remark:
 - In general, common functions are similar between different DBMSs,
 - Some functions have different formats or names,... especially for date, time and string data types → See documentations for each DBMS

1.4. Aggregation Operators: Functions

- Example

```
SELECT sjid, name, MIN(score), MAX(score), AVG(score), stddev_pop(score)
FROM (SELECT student_id sid, e.subject_id sjid, name,
             (midterm_score*(1-1.0*percentage_final_exam/100) +
              final_score*1.0*percentage_final_exam/100) score
      FROM enrollment e, subject sj
     WHERE sj.subject_id = e.subject_id) AS t
WHERE upper(sjid) LIKE 'IT%'
GROUP BY sjid, name;
```

result

| sjid | name | min | max | avg | stddev |
|--------|-------------------|------|------|------|--------|
| IT1110 | Tin học đại cương | 5.4 | 8.7 | 7.05 | 1.254 |
| IT3080 | Mạng máy tính | | | | |
| IT3090 | Cơ sở dữ liệu | 8.1 | 8.1 | 8.1 | 0 |
| IT4857 | Thị giác máy tính | 8.25 | 8.25 | 8.25 | 0 |
| IT4866 | Học máy | 8.4 | 8.4 | 8.4 | 0 |

1.4. NULL's ignored in Aggregation

- NULL: no contribution
- no non-NULL values in a column → the result: NULL
 - Exception: COUNT of an empty set is 0

```
SELECT AVG(percentage_final_exam)
FROM subject; ➔ 64=(60x2+70x3)/5
```

```
SELECT AVG(percentage_final_exam),
       count(percentage_final_exam)
FROM subject
WHERE subject_id NOT LIKE 'IT%';
```

result

| AVG | COUNT |
|------|-------|
| NULL | 0 |

subject

| subject_id | name | credit | percentage_final_exam |
|------------|----------------------|--------|-----------------------|
| IT1110 | Tin học đại cương | 4 | 60 |
| IT3080 | Mạng máy tính | 3 | 70 |
| IT3090 | Cơ sở dữ liệu | 3 | 70 |
| IT4857 | Thị giác máy tính | 3 | 60 |
| IT4866 | Học máy | 2 | 70 |
| LI0001 | life's happy song | 5 | |
| LI0002 | %life's happy song 2 | 5 | 30 |

1.5. Grouping results

- Syntax:

```
SELECT ...
FROM ...
[WHERE condition]
GROUP BY expr [,expr]...
```

student

| student_id | first_name | last_name | ... | gender | ... | clazz_id |
|------------|------------|-----------|-----|--------|-----|----------|
| 20160001 | Ngọc An | Bùi | ... | M | ... | |
| 20160002 | Anh | Hoàng | ... | M | ... | 20162101 |
| 20160003 | Thu Hồng | Trần | ... | F | ... | 20162101 |
| 20160004 | Minh Anh | Nguyễn | ... | F | ... | 20162101 |
| 20170001 | Nhật Ánh | Nguyễn | ... | F | ... | 20172201 |

- Example and Operational semantic:

```
SELECT clazz_id, count(student_id) 3
```

```
FROM student
```

```
WHERE gender = 'F' 1
```

```
GROUP BY clazz_id; 2
```

result

| clazz_id | count |
|----------|-------|
| 20162101 | 2 |
| 20172201 | 1 |

1.5. Grouping results

- Each element of the SELECT list must be either:
 - Aggregated, or
 - An attribute on the GROUP BY list

```
SELECT clazz_id, count(student_id), first_name
FROM student
WHERE gender = 'F'
GROUP BY clazz_id;
```

1.5. Grouping results: HAVING

- Syntax:

```
SELECT ...
FROM ...
[WHERE condition]
GROUP BY expr [,expr]...
HAVING <condition on group>
```

- Example:

```
SELECT clazz_id, count(student_id) 4
FROM student
WHERE gender = 'F' 1
GROUP BY clazz_id 2
HAVING count(student_id) >= 2; 3
```

result

| clazz_id | count |
|----------|-------|
| 20162101 | 2 |

1.5. Grouping results: HAVING

- Requirements on HAVING conditions:
 - Anything goes in a **subquery**
 - Outside subqueries, they may refer to attributes only if they are:
 - either a **grouping attribute**
 - or **aggregated**

```
SELECT subject_id, semester, count(student_id)
FROM enrollment
GROUP BY subject_id, semester
HAVING count(student_id) >= ALL
          (SELECT count(student_id)
           FROM enrollment
           GROUP BY subject_id, semester)
```

1.5. Grouping results: HAVING

- Which subject in which semester has it the most enrollments?

```
{ SELECT subject_id, semester, count(student_id)  
  FROM enrollment  
 GROUP BY subject_id, semester  
 HAVING count(student_id) >= ALL  
       (SELECT count(student_id)  
        FROM enrollment  
        GROUP BY subject_id, semester);
```

result

| subject_id | semester | count |
|------------|----------|-------|
| IT4857 | 20172 | 1 |
| IT3090 | 20172 | 1 |
| IT4866 | 20172 | 1 |
| IT3080 | 20172 | 2 |
| IT1110 | 20171 | 4 |

result

| subject_id | semester | count |
|------------|----------|-------|
| IT1110 | 20171 | 4 |

1.6. Controlling the output: Eliminating Duplicates

- Remove duplicate tuples: **DISTINCT**

```
SELECT DISTINCT student_id FROM enrollment;
```

- UNION | INTERSECT | EXCEPT: remove duplicate rows
- UNION | INTERSECT | EXCEPT **ALL**:
 - does not remove duplicate rows

1.6. Controlling the output: Eliminating Duplicates in an Aggregation

- Use **DISTINCT** inside aggregation

```
SELECT count(*) a,  
       count(distinct percentage_final_exam) b,  
       AVG(credit) c,  
       AVG(distinct credit) d  
FROM subject;
```

result

| a | b | c | d |
|---|---|------|-----|
| 7 | 3 | 3.57 | 3.5 |

subject

| subject_id | name | credit | percentage_final_exam |
|------------|----------------------|--------|-----------------------|
| IT1110 | Tin học đại cương | 4 | 60 |
| IT3080 | Mạng máy tính | 3 | 70 |
| IT3090 | Cơ sở dữ liệu | 3 | 70 |
| IT4857 | Thị giác máy tính | 3 | 60 |
| IT4866 | Học máy | 2 | 70 |
| LI0001 | life's happy song | 5 | |
| LI0002 | %life's happy song 2 | 5 | 37 |

1.6. Controlling the output: Ordering results

- Syntax and operational semantic:

```
SELECT ...
FROM ...
[WHERE condition]
[GROUP BY expr [,expr]... ]
[HAVING ...]
ORDER BY {expr|position} [ASC|DESC]
      [,expr|position} [ASC|DESC]
```

1

1.6. Controlling the output: Ordering results

- Example:

```
SELECT subject_id, semester, count(student_id)
FROM enrollment
GROUP BY subject_id, semester
ORDER BY semester,
         count(student_id) DESC, subject_id;
```

result

| subject_id | semester | count |
|------------|----------|-------|
| IT4857 | 20172 | 1 |
| IT3090 | 20172 | 1 |
| IT4866 | 20172 | 1 |
| IT3080 | 20172 | 2 |
| IT1110 | 20171 | 4 |

result

| subject_id | semester | count |
|------------|----------|-------|
| IT1110 | 20171 | 4 |
| IT3080 | 20172 | 2 |
| IT3090 | 20172 | 1 |
| IT4857 | 20172 | 1 |
| IT4866 | 20172 | 1 |

View

1. View definition
2. Accessing views
3. Updatable views
4. Materialized views

2.1. View definition

- A **view** is a relation defined in terms of stored tables (called **base tables**) and other views
- Two kinds:
 - **Virtual** = not stored in the database; just a query for constructing the relation
 - **Materialized** = actually constructed and stored
- Declaring views:
`CREATE [MATERIALIZED] VIEW <name> AS <query>;`
 - Default is **virtual**

2.1. View definition: View Removal

- Dropping views: `DROP VIEW <name>;`
`DROP VIEW female_student;`
- Affection:
 - Deleting the definition of views: the `female_student` view no longer exists
 - No tuples of the base relation (student relation) is affected

2.2. Accessing views

- Declare:

```
CREATE VIEW monitor AS  
    SELECT student_id, first_name, last_name, dob, clazz_id  
    FROM student, clazz  
    WHERE student_id = monitor_id ;
```

- Query a view as if it were a base table

```
SELECT student_id, first_name, last_name, dob  
FROM monitor  
WHERE clazz_id = '20172201' ;
```

- A limited ability to modify views

2.3. Updatable views

- The SQL rules are complex
- They permit modifications on views that are defined by selecting (using **SELECT**, **not SELECT DISTINCT**) some attributes from one relation R (which may itself be an updatable view):
 - The **WHERE** clause must **not involve R in a subquery**
 - The **FROM** clause can only consist of **one occurrence of R** and **no other relation**
 - The **list in the SELECT clause** must include **enough attributes** that for every tuple inserted into the relation R (other attributes filled with NULL values or the proper default)
 - There is **no GROUP BY** clause

2.3. Updatable views: Example

- Base table: `student (student_id, first_name, last_name, dob, gender, address, note, clazz_id)`
- Updatable view

```
CREATE VIEW female_student AS  
    SELECT student_id, first_name, last_name FROM student  
    WHERE gender = 'F';
```

- Insert into views:

```
INSERT INTO female_student VALUES('20160301', 'Hoai An', 'Tran');
```

means

```
INSERT INTO student(student_id, first_name, last_name)  
VALUES ('20160301', 'Hoai An', 'Tran');
```

2.3. Updatable views: Example

- Delete from views:

```
DELETE FROM female_student WHERE first_name LIKE '%An' ;
```

means

```
DELETE FROM student  
WHERE first_name LIKE '%An' AND gender = 'F';
```

- Update views:

```
UPDATE female_student SET first_name = 'Hoài Ân'  
WHERE first_name = 'Hoai An' ;
```

means

```
UPDATE student SET first_name = 'Hoài Ân'  
WHERE first_name = 'Hoai An' AND gender = 'F';
```

2.3. Updatable views: Views and INSTEAD OF trigger

- Generally, it is impossible to modify a virtual view, because it doesn't exist.
- But an **INSTEAD OF** trigger (next lesson) lets us interpret view modifications in a way that makes sense

```
CREATE TRIGGER delete_viewtrigger
  INSTEAD OF DELETE ON monitor
  FOR EACH ROW
  BEGIN
    UPDATE clazz SET monitor_id = NULL
    WHERE clazz_id = OLD.clazz_id;
  END;
```

2.4. Materialized Views

- Results of a query can be stored
- This enables much more efficient access
- Problems:
 - each time **a base table changes**, the materialized view may change
- Solutions:
 - Periodic reconstruction (**REFRESH**) of the materialized view
 - **Triggers** (next lesson)

Privileges and User Management in SQL

1. Privileges
2. Creating users
3. Granting privileges
4. Revoking privileges

3.1. Privileges

- **SELECT, INSERT, DELETE, UPDATE**: privileges on table/view
- **REFERENCES**: privilege on a relation; the right to refer to that relation in an integrity constraint
- **USAGE**: the right to use that element in one's own declarations
- **TRIGGER**: privilege on a relation; the right to define triggers on that relation
- **EXECUTE**: the right to execute a piece of code, such as a procedure or function
- **UNDER**: the right to create subtypes of a given type

3.2. Creating users

- Syntax: variations in different database platforms

- Creating an user in Oracle, MySQL:

`CREATE USER username IDENTIFIED BY password;`

- Creating an user in PostgreSQL:

`CREATE USER username`

`[[WITH] options] PASSWORD password;`

- Deleting:

`DROP USER username [CASCADE];`

- Example:

`CREATE USER toto IDENTIFIED BY pwdtoto`

3.3. Granting privileges

- Syntax:

```
GRANT <privilege list> ON <database element> TO <user list>  
[WITH GRANT OPTION] ;
```

- <privilege list> : INSERT, SELECT, ..., ALL PRIVILEGES
- <database element>: a table, a view
- WITH GRANT OPTION:
 - the user may grant the privilege to other user

- Example:

```
GRANT SELECT, INSERT ON student TO tom WITH GRANT OPTION;
```

3.4. Revoking privileges

- Syntax:

```
REVOKE <privilege list> ON <database element> FROM <user list>  
[CASCADE| RESTRICT] ;
```

- **CASCADE** : revoke any privileges that were granted only because of the revoked privileges
 - **RESTRICT**: the revoke statement cannot be executed if the revoked privileges have been passed on to others

```
REVOKE GRANT OPTION FOR .....; : remove the grant option
```

- Example:

```
REVOKE INSERT ON student FROM tom CASCADE;
```

Remark

- Complex query
 - Clauses in SQL statement are not exchangeable
 - A SQL statement executed successfully, it's not sure that this statement provides the correct result
 - A query provides correct result at a moment, it may not be the correct query for a demand
 - Be careful with "natural join"
- Virtual vs. materialized view
- Privileges and User Management
 - Superuser account is not for everybody
 - An user no need to access all database objects

Quiz 1.

| Quiz Number | 1 | Quiz Type | OX | Example Select |
|-------------|---|-----------|----|----------------|
| Question | What does the following SQL statement result? SELECT * FROM student WHERE (1=0); | | | |
| Example | <ul style="list-style-type: none">A. An empty relation with the same structure of "student"B. A relation with the same structure and data as "student"C. The query raises error | | | |
| Answer | A | | | |
| Feedback | Expression (1=0) gives false value, so all tuples of student do not satisfy this condition. There is no tuple in result relation. | | | |

Quiz 2.

| Quiz Number | 2 | Quiz Type | OX | Example Select |
|-------------|--|-----------|----|----------------|
| Question | We must always have join conditions if there are more than one relation in FROM clause ? | | | |
| Example | A. Yes B. No | | | |
| Answer | B | | | |
| Feedback | No, it is as cross join (called a Cartesian product), but the product by itself is rarely a useful operation | | | |

Quiz 3.

| Quiz Number | 3 | Quiz Type | OX | Example Select |
|-------------|--|-----------|----|----------------|
| Question | Can we put the condition in HAVING clause into the WHERE clause ? | | | |
| Example | <p>A. Sometimes yes B. No, never C. Yes, we can</p> | | | |
| Answer | A | | | |
| Feedback | <p>Conditions in HAVING clause and in WHERE clause are not the same meaning. Conditions in HAVING clause apply to groups as a whole. Conditions in WHERE clause apply to individual tuples.</p> <ul style="list-style-type: none">- If condition in HAVING clause refers to grouping attribute, then this condition can be placed in WHERE clause.- If condition in HAVING clause refers to aggregated attributes, it can not | | | |

Quiz 4.

| Quiz Number | 4 | Quiz Type | OX | Example Select |
|-------------|--|-----------|----|----------------|
| Question | What does the following SQL statement result? SELECT student_id FROM enrollment WHERE subject_id = 'IT3090' AND subject_id = 'IT4859' | | | |
| Example | <ul style="list-style-type: none">A. Empty relationB. List of student_ids that have enrolled both two subjects IT3090 and IT4859.C. List of student_ids that have enrolled at least one subject whose subject_id is IT3090 or IT4859 | | | |
| Answer | A | | | |
| Feedback | The condition in WHERE clause is always false. | | | |

Summary

1. Data manipulation (part 2)

- Joins operators
- Subqueries: in FROM clause and in WHERE clause
- Aggregation operators
- Grouping and aggregation in SQL , conditions in HAVING clause
- Controlling the output: duplicate elimination, ordering the result

2. View

- View definition
- View accessing
- Updatable view
- Materialized view

3. Privileges and User Managements

- Privileges
- Creating user
- Granting / Revoking privileges



Next lesson: Constraints and Triggers

- Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom. Database Systems: The Complete Book. Pearson Prentice Hall. the 2nd edition. 2008: Chapter 6, 8
- Nguyen Kim Anh, Nguyên lý các hệ cơ sở dữ liệu, NXB Giáo dục. 2004: Chương 3