

25 YEARS ANNIVERSARY  
SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY  
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

# IT3090E - Databases

## Chapter 4: Structured Query Language *part 3*

Muriel VISANI

[murielv@soict.hust.edu.vn](mailto:murielv@soict.hust.edu.vn)

# Contents

- Chapter 1: Introduction
- Chapter 2: Relational databases
- Chapter 3: Relational algebra
- **Chapter 4: Structured Query Language (SQL)**
- Chapter 5: Database Design
- Chapter 6: Indexing
- Chapter 7: Query processing and optimization
- Chapter 8: Constraints, rules and triggers
- Chapter 9: Security
- *(Optional) Chapter 10: Transactions: concurrency and recovery*

# Outline

1. Data Manipulation: SQL Data Manipulation Language for retrieving the data
2. Creating and managing views
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**

# Global Outline of Chapter 4

- Chapter 4 - Part 1:
  - 1 - Introduction to SQL
  - 2 – Definition of a Relational Schema (DDL)
  - 3 – Data Manipulation: 3.1.-3.3. Insertion, deletion, updates
- Chapter 4 - Part 2:
  - 3.4. Data Manipulation Language for Querying (simple queries)
- Chapter 4 - Part 3:
  - **3.4. Data Manipulation Language for Querying (complex queries)**

# Keywords

Keyword	Description
<b>Query</b>	A request (SQL statement) for information from a database
<b>Subquery</b>	A subquery (inner query, nested query) is a query within another (SQL) query.
<b>Privileges</b>	Database access permissions
<b>View</b>	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 Language for querying: Advanced SQL Retrieval statement

- a) Advanced joins operators
- b) Subqueries (inside FROM clauses and inside WHERE clauses)
- c) Aggregation operators (make calculations in SQL)
- d) Grouping and aggregation in SQL,
- e) HAVING clauses
- f) Controlling outputs: eliminating duplicates, ordering results
- g) Divisions



# 1. Example of a database schema

student(student\_id, first\_name, last\_name, dob, gender, address, email, *program\_id*)  
subject(subject\_id, name, credit, percentage\_final\_exam)  
lecturer(lecturer\_id, first\_name, last\_name, dob, gender, address, email)  
teaching(subject\_id, lecturer\_in\_charge)  
program(program\_id, name, *lecturer\_in\_charge*)  
scores\_per\_subject(student\_id, subject\_id, semester, midterm\_score, final\_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  $\geq 3.2$ ?



# 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?

**scores\_per\_subject**

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

- 1.1. Advanced joins operators
- 1.2. Subqueries (inside FROM clause and inside WHERE clause)
- 1.3. Aggregation operators (make calculations in SQL)
- 1.4. Grouping and aggregation in SQL
- 1.5. HAVING clauses
- 1.6. Controlling outputs: eliminating duplicates, ordering results
- 1.7. Division

# 1.1. Advanced join operators

- Syntax:

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

- Example (with a variation of the current database):

student(student\_id, first\_name, last\_name, dob, gender, address, note, *course\_id*)

course(course\_id, name, *lecturer\_id#*, *TA\_id#*)

- Question: what does the below query do?

```
SELECT course.course_id, last_name, first_name  
FROM course, student  
WHERE student_id = TA_id;
```

# 1.1. Advanced join operators: Operational semantics

course

course_id	lecturer_id	TA_id
20181101	02001	20150003
20162102		
20172201	02002	20160001
20172202		

student

student_id	first_name	last_name	...	course_id
20150001	Ngọc An	Bùi		
20150002	Anh	Hoàng		20152101
20150003	Thu Hồng	Trần		20152101
20150004	Minh Anh	Nguyễn		20152101
20160001	Nhật Ánh	Nguyễn		20162201

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

```
SELECT course.course_id, name,  
       student.last_name,  
       student.first name  
FROM course, student  
WHERE student_id = TA_id
```

result

course_id	last_name	first_name
20182101	Trần	Thu Hồng
20172201	Nguyễn	Nhật Ánh

# 1.1. Advanced join 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.course_id, name, s.last_name, s.first_name  
FROM course AS c, student s  
WHERE s.student_id = c.TA_id
```

## 1.1. Advanced join operators: Self-join

subject(subject\_id, name, credit, percentage\_final\_exam)

Example: 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. Advanced join operators: Example

student(student\_id, first\_name, last\_name, dob, gender, address, program\_id#)

subject(subject\_id, name, credit, percentage\_final\_exam)

scores\_per\_subject(student\_id#, subject\_id#, semester, midterm\_score, final\_score)

Example: List of students having enrolled in subjects in semester 20172: student full name, subject name, subject credit:

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

## 1.1. Advanced join operators: Join types

- **Cartesian product:**
  - R **CROSS JOIN** S
- Theta join: (INNER JOIN)
  - R [**INNER**] **JOIN** S **ON** <condition>
- Natural join: (**Be careful!**) -> my personal recommendation is to avoid using it!!!
  - 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. Advanced join operators: Join types

- Cartesian product:
  - S **CROSS JOIN** R
    - Equivalent to SELECT \* from S, R;

## PRODUCERS

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

## PRODUCTS

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

# 1.1. Advanced join operators: Join types

- Cartesian product:

- PRODUCERS

## CROSS JOIN PRODUCTS

producer_id	name	address	reference	designation	unit_price	Producer_id
1369	denecker sarl	Lyon	139	Lamp rhapsody	30.00	1623
1369	denecker sarl	Lyon	248	Female shoes camargue	75.00	1623
1369	denecker sarl	Lyon	258	Male shoes camargue	87.00	1623
1369	denecker sarl	Lyon	416	Backpack dolpo	100.00	1369
1369	denecker sarl	Lyon	426	Backpack nepal	155.00	1371
1369	denecker sarl	Lyon	765	tent	300.00	1502
1370	chen'alpe diffusion	Lyon	139	Lamp rhapsody	30.00	1623
1370	chen'alpe diffusion	Lyon	248	Female shoes camargue	75.00	1623
1370	chen'alpe diffusion	Lyon	258	Male shoes camargue	87.00	1623
1370	chen'alpe diffusion	Lyon	416	Backpack dolpo	100.00	1369
1370	chen'alpe diffusion	Lyon	426	Backpack nepal	155.00	1371
1370	chen'alpe diffusion	Lyon	765	tent	300.00	1502
1502	rodenas francois	Toulouse	139	Lamp rhapsody	30.00	1623
1502	rodenas francois	Toulouse	248	Female shoes camargue	75.00	1623
1502	rodenas francois	Toulouse	258	Male shoes camargue	87.00	1623
1502	rodenas francois	Toulouse	416	Backpack dolpo	100.00	1369
1502	rodenas francois	Toulouse	426	Backpack nepal	155.00	1371
1502	rodenas francois	Toulouse	765	tent	300.00	1502
1623	adidas	Dettwiller	139	Lamp rhapsody	30.00	1623
1623	adidas	Dettwiller	248	Female shoes camargue	75.00	1623
1623	adidas	Dettwiller	258	Male shoes camargue	87.00	1623
1623	adidas	Dettwiller	416	Backpack dolpo	100.00	1369
1623	adidas	Dettwiller	426	Backpack nepal	155.00	1371
1623	adidas	Dettwiller	765	tent	300.00	1502

## 1.1. Advanced join operators: Join types

- Cartesian product:
  - R **CROSS JOIN** S
- **Theta join: (INNER JOIN)**
  - R [**INNER**] **JOIN** S **ON** <condition>
- Natural join: (**Be careful!**) -> my personal recommendation is to avoid using it!!!
  - 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. Advanced join operators: Join types

- INNER JOIN

- R [INNER] JOIN S ON <condition>

- Equivalent to SELECT \* from S, R where <condition>;

- Example: select \* from PRODUCERS S, PRODUCTS P

where R.producer\_id=S.producer\_id

## PRODUCERS

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

## PRODUCTS

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

# 1.1. Advanced join operators: Join types

- INNER JOIN

- R [INNER] JOIN S ON <condition>

- Equivalent to SELECT \* from S, R where <condition>;

- Example: select \* from PRODUCERS S, PRODUCTS P

where R.producer\_id=S.producer\_id

producer_id	name	address	reference	designation	unit_price	Producer_id
1369	denecker sarl	Lyon	416	backpack dolpo	100.00	1369
1502	rodenas francois	Toulouse	765	tent	300.00	1502
1623	adidas	Dettwiller	139	lamp rhapsody	30.00	1623
1623	adidas	Dettwiller	248	Female shoes camargue	75.00	1623
1623	adidas	Dettwiller	258	Male shoes camargue	87.00	1623

# 1.1. Advanced join operators: Join types

- INNER JOIN

- R [INNER] JOIN S ON <condition>

- Equivalent to SELECT \* from S, R where <condition>;
    - Example: select \* from PRODUCERS S, PRODUCTS P

where R.producer\_id=S.producer\_id

PRODUCERS

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rouenas francois	Toulouse
1623	adidas	Dettwiller

PRODUCTS

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
418	Backpack dorpo	100.00	1370
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

Not in the INNER JOIN's result



## 1.1. Advanced join operators: Join types

- Cartesian product:
  - R **CROSS JOIN** S
- Theta join: (INNER JOIN)
  - R [**INNER**] **JOIN** S **ON** <condition>
- **Natural join:** (**Be careful!**) -> my personal recommendation is to avoid using it!!!
  - 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. Advanced join operators: Join types

## – R NATURAL JOIN S

- Finds automatically the “natural” attribute to make the join
- Usually, the first attribute with the same name in the two tables!
  - Here, the result is the same as with an inner join
  - But, in general, VERY uncertain! -> I recommend you not to use it
    - » Example: what if “designation” was called “name”?

### PRODUCERS

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

### PRODUCTS

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

## 1.1. Advanced join operators: Join types

- Cartesian product:
  - R **CROSS JOIN** S
- Theta join: (INNER JOIN)
  - R [**INNER**] **JOIN** S **ON** <condition>
- Natural join: (**Be careful!**) -> my personal recommendation is to avoid using it!!!
  - 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. Advanced join operators: Join types

- Outer join:
  - R [LEFT|RIGHT|FULL] [OUTER] JOIN S ON <condition>
  - R NATURAL [LEFT|RIGHT|FULL] [OUTER] JOIN S
- Example:
  - SELECT \* from S LEFT OUTER JOIN P  
ON S.producer\_id=P.producer\_id;

PRODUCERS

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

PRODUCTS

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

# 1.1. Advanced join operators: Join types

- Example:
  - SELECT \* from S **LEFT** OUTER JOIN P  
ON S.producer\_id=P.producer\_id;

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

producer_id	name	address	reference	designation	unit_price	Producer_id
1369	denecker sarl	Lyon	416	backpack dolpo	100.00	1369
1502	rodenas francois	Toulouse	765	tent	300.00	1502
1623	adidas	Dettwiller	139	lamp rhapsody	30.00	1623
1623	adidas	Dettwiller	248	Female shoes camargue	75.00	1623
1623	adidas	Dettwiller	258	Male shoes camargue	87.00	1623
1370	Chen'alpe diffusion	Lyon	NULL	NULL	NULL	NULL

Was not in the INNER JOIN result

# 1.1. Advanced join operators: Join types

- Example:
  - SELECT \* from S **RIGHT** OUTER JOIN P  
ON S.producer\_id=P.producer\_id;

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

producer_id	name	address	reference	designation	unit_price	Producer_id
1369	denecker sarl	Lyon	416	backpack dolpo	100.00	1369
1502	rodenas francois	Toulouse	765	tent	300.00	1502
1623	adidas	Dettwiller	139	lamp rhapsody	30.00	1623
1623	adidas	Dettwiller	248	Female shoes camargue	75.00	1623
1623	adidas	Dettwiller	258	Male shoes camargue	87.00	1623
NULL	NULL	NULL	426	Backpack nepal	155.00	1371

Was not in the INNER  
JOIN result

# 1.1. Advanced join operators: Join types

- Example:
  - SELECT \* from S **FULL** OUTER JOIN P  
ON S.producer\_id=P.producer\_id;

producer_id	name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffusion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

reference	designation	unit_price	producer_id
139	Lamp rhapsody	30.00	1623
248	Female shoes camargue	75.00	1623
258	Male shoes camargue	87.00	1623
416	Backpack dolpo	100.00	1369
426	Backpack nepal	155.00	1371
765	tent	300.00	1502

producer_id	name	address	reference	designation	unit_price	Producer_id
1369	denecker sarl	Lyon	416	backpack dolpo	100.00	1369
1502	rodenas francois	Toulouse	765	tent	300.00	1502
1623	adidas	Dettwiller	139	lamp rhapsody	30.00	1623
1623	adidas	Dettwiller	248	Female shoes camargue	75.00	1623
1623	adidas	Dettwiller	258	Male shoes camargue	87.00	1623
NULL	NULL	NULL	426	Backpack nepal	155.00	1371
1370	Chen'alpe	Lyon	NULL	NULL	NULL	NULL

Were not in the INNER JOIN result

## 1.1. Advanced join operators: Join types

- NATURAL LEFT JOIN, NATURAL RIGHT JOIN
  - In this case, same result as OUTER LEFT JOIN, OUTER RIGHT JOIN
  - But more risk to make errors, I would not recommend it!



## 1.2. Sub-queries (nested 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 returning a scalar value

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

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

## 1.2. Sub-queries: IN operators

- Syntax: <tuple> [**NOT** ] **IN** <subquery>
  - Can be used as INNER JOIN!!!
- Example: First name, last name and address of TAs? (variation of our DB)

student(student\_id, first\_name, last\_name, dob, gender, address, note, *course\_id*)  
course(course\_id, name, lecturer\_id#, TA\_id#)

```
SELECT first_name, last_name, address  
FROM student  
WHERE student_id IN (SELECT TA_id FROM course) ;
```

## 1.2. Sub-queries: EXISTS

- Syntax:

[NOT] EXISTS (<subquery>)

EXISTS (<subquery>): TRUE iff <subquery> result is **not empty**

- Question: what does this query return?

-> 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: EXISTS

- Can also be used to express INNER JOIN!
- Ex: First name, last name and address of TAs? (variation of our DB)

student(student\_id, first\_name, last\_name, dob, gender, address, note, course\_id)

course(course\_id, name, lecturer\_id#, TA\_id#)

```
SELECT first_name, last_name, address
FROM student
WHERE student_id IN (SELECT TA_id FROM course);
```

- Question: write the same query, but using the EXISTS operator

```
SELECT first_name, last_name, address
FROM student
WHERE EXISTS (SELECT TA_id FROM course WHERE TA_id = student_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
    - Can be used to express **maximum, minimum**
  - `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: what does this query return? -> subjects having the maximum number of credits

**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);

**SELECT \***

**FROM** subject

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

result

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

- So far, we talked about using subqueries in WHERE clauses
- But, subquery can also be 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. 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	

# 1.3. 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, **UPPER**...
  - Date/Time functions: DATE, DAY, MONTH, YEAR, HOUR, MINUTE, ...
  - ...
  - 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**

# 1.3. Aggregation Operators: Functions

- Example

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

ssid	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

# 1.3. N.B. NULL is 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; → 66 = (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	

# 1.4. Grouping results

- Syntax:

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

student

student_id	first_name	last_name	...	gender	...	program_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 program_id, count(student_id) 3  
FROM student 1  
WHERE gender = 'F'  
GROUP BY program_id; 2
```

result

program_id	count
20162101	2
20172201	1

## 1.4. Grouping results

- Each element of the SELECT list must be either:

- Aggregated, or
- An attribute on the GROUP BY list
- Example:

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

- Question: why???

## 1.5. Grouping results: HAVING

- The aggregate functions can be used in a SELECT...
  - ... or in the HAVING clause of a SELECT statement that includes a GROUP BY clause
- The HAVING clause enables to express conditions on aggregate functions
  - Aggregate functions are not allowed in WHERE statements
- Example:

SELECT Country

FROM Customers

GROUP BY Country

HAVING COUNT(CustomerID) > 5;

## 1.5. Grouping results: HAVING

- Aggregate functions in HAVING statement can be used with or without aggregate function in the select statement
- If there is an aggregate function in the SELECT statement and one in the HAVING clause, the two aggregate functions **do not** need to be the same
- Examples:

```
SELECT COUNT(city), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;
```

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;
```



# 1.5. Grouping results: HAVING

- Syntax:

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

- Example:

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

result

program_id	count
20162101	2

## 1.5. Grouping results: HAVING

- Question: which question does the following query answer to?

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

## 1.5. Grouping results: HAVING

- Which subject in which semester had the most students?

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

## 1.5. Grouping results: HAVING

- HAVING clauses can also express a condition on the groups in the GROUP BY
  - HAVING clauses do **not** necessarily contain aggregation functions!
- Example:  

```
SELECT year(birthdate), count(*)  
FROM Employees  
group by year(Birthdate)  
Having year(Birthdate)>1960 ;
```
- Question: Give a SQL statement which will give the same output, but with a where instead of a having.

## 1.6. Controlling the output: Eliminating Duplicates

- Remove duplicate tuples: **DISTINCT**

```
SELECT DISTINCT student_id FROM scores_per_subject;
```

- 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	

# 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 scores_by_subject
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



## 1.7. Division

- Example from the relational schema in Chapter 2

SHOPS (no\_shop, city, name\_manager)

CUSTOMERS (no\_customer, name, country, city, type)

ITEMS(no\_item, name, weight, color, qty\_stock, BuyingPrice, SellingPrice, no\_supplier#)

DELIVERY (no\_delivery, date\_del, no\_customer#, no\_shop#)

ORDERS (no\_order, date, no\_customer#, no\_shop#)

SUPPLIERS (no\_supplier, supplier\_name)

DELIVERY\_DETAILS (no\_delivery#, no\_item#, qty, no\_order#)

ORDER\_DETAILS (no\_order#, no\_item#, qty, no\_delivery#, total\_amount)

## 1.7. Division

- Introductory example (this is not a division yet):

*Items which do not appear in any order line*

$$S = \{ a \in \text{Items} \mid \neg a \in \text{Order\_Details} \}$$

$$\Leftrightarrow S = \{ a \in \text{Items} \mid \forall c \in \text{Order\_Details}, \\ \neg \text{Order\_Details}(a, c) \}$$

*How to express  $\forall$ ?*

*$\Rightarrow$  Using its negation.*

$$S = \{ a \in \text{Items} \mid \nexists c \in \text{Order\_Details}, \\ \text{Order\_Details}(a, c) \}$$

## 1.7. Division

- Introductory example (this is not a division yet):

$S = \{ a \in \text{Items} \mid \nexists c \in \text{ORDER\_Details},$   
 $\text{ORDER\_Details}(a, c) \}$

May be translated in SQL by:

```
select * from ITEMS I
where not exists ( select *
                  from ORDER_Details OL
                  where OL.item_no = I.item_no);
```

## 1.7. Division

- Introductive example (this is a division): Which items have been ordered in every order?

1) *PROJECTION of all the order numbers  $\Rightarrow D$*

2) *PROJECTION of all the items along with the order numbers where the item appear  $\Rightarrow N$*

3.4.1. Division:  $R = N / D$

## 1.7. Division

- Introductive example (this is a division): Which items have been ordered in every order?

$$S = \{ a \in \text{Items} \mid \forall c \in \text{Orders}, \text{Order\_Details}(a, c) \}$$

$$\Leftrightarrow S = \{ a \in \text{Items} \mid \nexists c \in \text{Orders}, \neg \text{Order\_Details}(a, c) \}$$

$$\Leftrightarrow S = \{ a \in \text{Items} \mid \nexists c \in \text{Orders}, \forall lc \in \text{Order\_Details}, \neg lc(a, c) \}$$

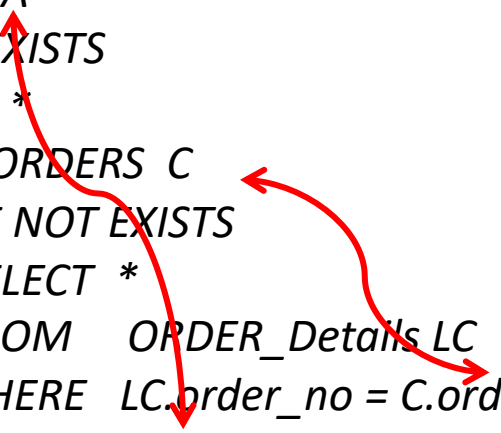
$$\Leftrightarrow S = \{ a \in \text{Items} \mid \nexists c \in \text{Orders}, \nexists lc \in \text{Order\_Details}, lc(a, c) \}$$

*List of all the items such that there is (exists) no order such that this item does not appear (exists) in at least one order line of this order*

## 1.7. Division

- Introductive example (this is a division): Which items have been ordered in every order?

```
SELECT *  
FROM ITEMS A  
WHERE NOT EXISTS  
  ( SELECT *  
    FROM ORDERS C  
    WHERE NOT EXISTS  
      ( SELECT *  
        FROM ORDER_Details LC  
        WHERE LC.order_no = C.order_no  
              AND A.item_no = LC.item_no )  
    );
```

A diagram with three red arrows illustrating the division query structure. The first arrow starts from the 'WHERE NOT EXISTS' clause of the outer query and points to the 'WHERE NOT EXISTS' clause of the first nested query. The second arrow starts from the 'WHERE NOT EXISTS' clause of the first nested query and points to the 'WHERE NOT EXISTS' clause of the second nested query. The third arrow starts from the 'WHERE NOT EXISTS' clause of the second nested query and points to the 'WHERE LC.order\_no = C.order\_no' condition.

## 1.7. Division

```
SELECT names
FROM ITEMS A
WHERE NOT EXISTS
  ( SELECT *
    FROM SHOPS M
    WHERE NOT EXISTS
      ( SELECT *
        FROM ORDERS CD,
          ORDER_Details LCD
        WHERE CD.order_no = LCD.order_no
        WHERE LCD.item_no = A.item_no
        AND CD.shop_no = M.shop_no )
    )
```

*/\* Question: What does this query return?\*/*

## 1.7. Division

```
SELECT name
FROM ITEMS A
WHERE NOT EXISTS
  ( SELECT *
    FROM SHOPS M
    WHERE city= 'La Rochelle' and NOT EXISTS
      ( SELECT *
        FROM ORDERS CD,
          ORDER_Details LCD
        WHERE CD.order_no = LCD.order_no
        WHERE LCD.item_no = A.item_no
        AND CD.shop_no = M.shop_no )
  )
```

*/\* Question: What does this query return?\*/*



## 1.7. Division

```
SELECT name
FROM ITEMS A
WHERE color = 'green' and NOT EXISTS
    ( SELECT *
      FROM SHOPS M
      WHERE NOT EXISTS
          ( SELECT *
            FROM ORDERS CD,
              ORDER_Details LCD
            WHERE CD.order_no = LCD.order_no
              WHERE LCD.item_no = A.item_no
              AND CD.shop_no = M.shop_no )
    )
```

*/\* Question: What does this query return?\*/*

# Views

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

## 2.1. View definition

- A **view** is a relation defined on the basis of stored tables (called **base tables**) and/or other views
- Two kinds:
  - **Virtual** (by default) = not stored in the database; just some kind of pre-defined query
    - There is a SQL standard of defining a view
  - **Materialized** = actually built and stored in the database
    - SQL does not provides any standard way of defining materialized view, however some database management system provides custom extensions to use materialized views.
- Declaring views:

**CREATE [MATERIALIZED] VIEW** <name> **AS** <query>;

## 2.1. View definition

- A view can be queried exactly like a table (select \* from <view\_name> - see later)
- The main difference between a view and a table is that the records in the views are computed on-the-fly every time the view is queried
  - The records in the view do not need to be updated every time the base tables are updated
- The views are often created to fit the specific needs of some end-user of the database, without modifying the database itself

## 2.1. View definition: View Removal

- Dropping views: **DROP VIEW** <name>;  
**DROP VIEW** female\_student;
- Consequences of dropping a view:
  - Deleting the definition of views: the female\_student view no longer exists
  - **No record** of the base relation (student relation) is affected

## 2.2. Accessing views

- Declare:

```
CREATE VIEW TA AS
```

```
SELECT student_id, first_name, last_name, dob, program_id
```

```
FROM student s, class c
```

```
WHERE s.student_id = c.TA_id ;
```

- Query a view (we do that exactly as if it were a base table)

```
SELECT student_id, first_name, last_name, dob
```

```
FROM TA
```

```
WHERE class_id = '20172201' ;
```

## 2.3. Updatable views

- The SQL rules are complex for updating views
  - Because the views depend on the base tables!!!
  - So, an update in a view should trigger an update in the base table(s)
  - Therefore, the matching between the row in the view and the row in the base table must be exact
  - That's the main reasons why there are strict rules for updating views in SQL

## 2.3. Updatable views

- SQL rules for updating views
  - Updates are only possible 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)
    - If the view contains joins between multiple tables, you can only insert and update records from one table in the 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
  - There must be no **GROUP BY** clause
  - The **list of attributes** to update /insert must cover all NOT NULL attributes from  $R$  with no default value which value is not specified in the view
    - In particular, the view must include the PRIMARY KEY attribute of  $R$
  - If the view you want to update is based upon another view, that view should be updatable



## 2.3. Updatable views: Example

- Base table: `student (student_id, first_name, last_name, dob, gender, address, note, program_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');
```

Results in:

```
INSERT INTO student(student_id, first_name, last_name)
```

```
VALUES ('20160301', 'Hoai An', 'Tran');
```

## 2.3. Updatable views: Example

- Update views:

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

Results in

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

## 2.3. Delete from views: Views and INSTEAD OF trigger

- Generally, it is impossible to delete from a virtual view
- But an **INSTEAD OF** trigger (which we'll cover in next lectures) can help us control the effects of the delete on the base table

```
CREATE TRIGGER delete_view_trigger
  INSTEAD OF DELETE ON TA
  FOR EACH ROW
  BEGIN
    UPDATE class SET TA_id = NULL
    WHERE class_id = OLD.class_id;
  END;
```

## 2.4. Materialized Views

- A materialized view is the results of a query that is 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)



25 YEARS ANNIVERSARY  
**SOICT**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**  
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

**Thank you for  
your attention!**

 [soict.hust.edu.vn/](http://soict.hust.edu.vn/)  [fb.com/groups/soict](https://fb.com/groups/soict)



# Questions

