ĐẠI HỌC

BÁCH KHOA

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 2

Muriel VISANI
murielv@soict.hust.edu.vn

Contents

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. Insertion
    - 3.2. Deletion, update
    - 3.3. Examples of errors
- Chapter 4 - Part 2:
    - **3.4. Data Manipulation Language for retrieving the data**

Outline of the rest of Chapter 4

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 of the rest of chapter 4

Focus of this lecture

- 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 use rs can query just as they would in a persistent database collection object.

Data Manipulation Language for querying: Simple SQL Retrieval statement

Translation Relational Algebra -> SQL queries

3.4.1.Translation relational algebra → SQL Query Language

Every relational operator may be expressed using SQL, with the SELECT statement.

SELECT [DISTINCT] <list of attributes>
FROM <list of tables>
WHERE
<predicate>

<u>Predicate</u>: selection, comparison criterion.

<u>If the predicate is verified by a record, then this record is a part of the result.</u>

3.4.1.Translation relational algebra → SQL Query Language

- *SELECTION (unary operator)*

Example: I want to extract from the table PRODUCTS only the records which unit price is ≥ 155€.

reference	designation	unit_price	producer
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	1369
765	tent	300.00	1502

Selection corresponds to: $R1 = \sigma_{unit\_price>=155}(PRODUCTS)$

reference	designation	unit_price	producer
426	Backpack nepal	155.00	1369
765	Tent	300.00	1502

3.4.1.Translation relational algebra → SQL Query Language

- *SELECTION*

Ex 1: R1= $\sigma_{\text{unit\_price} >=155}$(PRODUCTS) is expressed in SQL as:

```
SELECT *        FROM products
WHERE unit_price >=155;
```

SQL keyword for selection

Ex 2 (2 predicates):

R1= $\sigma_{\text{unit\_price} >=155}$(PRODUCTS)

R2= $\sigma_{\text{producer} = 1369}$(R1) is expressed in SQL as:

```
SELECT *  FROM products
WHERE unit_price >=155
AND producer = 1369;
```

3.4.1.Translation relational algebra → SQL Query Language

•*PROJECTION (unary operator)*

Projection $\pi_{a1,a2,a3,a4,\ldots ak}(R)$ applies to a table R and extracts only the attributes a1,a2,a3,...ak from that table.

Therefore, contrary to the selection, the columns (and not the lines) are suppressed

PRODUCTS	reference	designation
	139	Lamp rhapsody
	248	Female shoes camargue
	258	Male shoes camargue
	416	Backpack dolpo
	426	Backpack nepal
	765	tent

$R2 = \pi_{\text{reference, designation}}(\text{PRODUCTS})$

3.4.1.Translation relational algebra → SQL Query Language

- *PROJECTION*

  R1 = $\pi_{\text{(reference,designation)}}$ (PRODUCTS) is expressed in SQL as:

  Names of the attributes for projection

  ```
  SELECT reference, designation
  FROM products;
  ```

- *PROJECTION of all the attributes from a table*

  R1 = π (PRODUCTS) is expressed in SQL as:

  ```
  SELECT *
  FROM products;
  ```

3.4.1.Translation relational algebra → SQL Query Language

- *PROJECTION:*
  - *We can also:*
    - *Give names (aliases) to the projected attributes*

R1 = $\pi_{\text{(reference,designation)}}$ (PRODUCTS)

SELECT reference as number, designation
FROM products;

- *Suppress duplicates on the projected attributes*

R1 = $\pi_{\text{(designation)}}$ (PRODUCTS)

SELECT distinct designation FROM products;

3.4.1.Translation relational algebra → SQL Query Language

- Composition of selection and projection

$R1 = \sigma_{unit\_price>=155}(PRODUCTS)$

$R2 = \pi_{reference,\ designation}(R1)$

TABLE PRODUCTS

reference	designation	unit_price	producer
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	1369
765	tent	300.00	1502

TABLE R2

reference	designation
426	Backpack népal
765	Tent

3.4.1.Translation relational algebra → SQL Query Language

- *SELECTION + PROJECTION*

$R1= \sigma_{\text{unit\_price} >=155}(\text{PRODUCTS})$

$R2= \sigma_{\text{no\_four} = 1369}(R1)$

$R3= \pi_{(\text{reference,designation})} (R2)$

is expressed in SQL as:

Projection (R3)

```
SELECT reference, designation
FROM products
WHERE unit_price >=155
AND producer = 1369;
```

Selections (R1,R2)

3.4.1.Translation relational algebra → SQL Query Language

• *CARTESIAN PRODUCT (binary operator)*

The first and most important binary operator is the cartesian product x. The cartesian product between two tables R and S is denoted by: R x S and creates a new table where each record from R is associated to each record de S.

Given R=PRODUCTS and S =PRODUCERS

reference	designation	unit_price	producer
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	1369
765	tent	300.00	1502

Producer_id	Name	address
1369	denecker sarl	Lyon
1370	chen'alpe diffussion	Lyon
1502	rodenas francois	Toulouse
1623	adidas	Dettwiller

Beware of the attributes which share the same name !!!!

CARTESIAN PRODUCT result

Producer_id	Name	address	reference	designation	unit_price	producer
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	1369
1369	denecker sarl	Lyon	765	tent	300.00	1502
1370	chen'alpe diffussion	Lyon	139	Lamp rhapsody	30.00	1623
1370	chen'alpe diffussion	Lyon	248	Female shoes camargue	75.00	1623
1370	chen'alpe diffussion	Lyon	258	Male shoes camargue	87.00	1623
1370	chen'alpe diffussion	Lyon	416	Backpack dolpo	100.00	1369
1370	chen'alpe diffussion	Lyon	426	Backpack nepal	155.00	1369
1370	chen'alpe diffussion	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	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
1623	adidas	Dettwiller	416	Backpack dolpo	100.00	1369
1623	adidas	Dettwiller	426	Backpack nepal	155.00	1369
1623	adidas	Dettwiller	765	tent	300.00	1502

3.4.1.Translation relational algebra → SQL Query Language

- *CARTESIAN PRODUCT*

R1 = products x producers   is expressed in SQL as:

```
SELECT *
FROM products , producers;
```

Cartesian product

3.4.1.Translation relational algebra → SQL Query Language

- Composition of Cartesian product and selection: INNER JOIN

  R1= PRODUCTS x PRODUCER

  $R2 = \sigma_{producer=producer\_id}(R1)$

Producer_id	Name	Address	Reference	designation	Unit_price	producer
1369	denecker sarl	Lyon	416	backpack dolpo	100.00	1369
1369	denecker sarl	Lyon	426	backpack nepal	155.00	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

INNER JOIN: cartesian product, then selection based on the equality of 2 attributes (here the foreign key of PRODUCTS and the primary key of PRODUCER)

3.4.1.Translation relational algebra → SQL Query Language

- *INNER JOIN*

R1 = Products (producer) ▶◀ Producers (producer_id) **can** be expressed in SQL as:

SELECT *

Cartesian product

FROM products , producers

Selection (with equality predicate)

WHERE products.producer = producers.producer_id;

- So, if you use the above syntax and get too many records, it is probably because…
  - … You have done something wrong during the selection (where statement)!!!
  - … And your result is just a Cartesian product!

3.4.1.Translation relational algebra → SQL Query Language

- For this reason, it is sometimes preferable to use the following syntax

R1 = Products (producer) ⋈ Producers (producer_id) **can also** be expressed in SQL as:

```
SELECT *
FROM products INNER JOIN producers
        ON products.producer = producers.producer_id;
```

INNER JOIN

- But, for some students, this syntax might be a bit more complicated to handle when there are more than 2 tables to join.
- For instance, when combined with a projection:

```
SELECT c.name, p.designation, s.name
    FROM products p
    INNER JOIN product_categories c ON c.category_id = p.category_id
    INNER JOIN produced s ON p.producer = s.producer_id;
```

3.4.1.Translation relational algebra → SQL Query Language

- Both syntaxes (with Cartesian product + WHERE, or with INNER JOIN) are OK **in my opinion**, as long as you're comfortable with it...
  - For the practicals, you should follow the instructions given by the lecturer in charge

3.4.1.Translation relational algebra → SQL Query Language

- Composition of INNER JOIN, projection and selection

Example: What are unit prices of the products produced by Adidas?

Producer_id	Name	Address	Reference	designation	Unit_price	producer
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

R1= Products (producer) $\bowtie$ Producers (producer_id)

R2 = $\sigma_{\text{Producer.name="Adidas"}}$ (R1)

R3= $\pi_{\text{unit\_price}}$(R2)

Only the records for which the producer's name is «Adidas » are extracted.

3.4.1.Translation relational algebra → SQL Query Language

- Composition of INNER JOIN, projection and selection

R1= Products (producer) ▶◀ Producers (producer_id)

R2 = $\sigma_{\text{Producer.name="Adidas"}}$ (R1)

R3= $\pi_{\text{unit\_price}}$(R2)

PROJECTION

```
SELECT unit_price
FROM products  P, producers S
WHERE P.producer = S.producer_id
AND S.name='Adidas';
```

INNER JOIN

SELECTION

3.4.1.Translation relational algebra → SQL Query Language

- Question: Re-write the following composition of INNER JOIN, projection and selection Using the INNER JOIN SQL keyword

$$R1 = \text{Products (producer)} \bowtie \text{Producers (producer\_id)}$$

$$R2 = \sigma_{\text{Producer.name="Adidas"}}(R1)$$

$$R3 = \pi_{\text{unit\_price}}(R2)$$

- Solution:

3.4.1.Translation relational algebra → SQL Query Language

- *UNION*

In our example, the schemes of the two tables are different =>
we first have to build two tables with the same scheme

- Products which price >=155: *SELECTION*.

  $R1 = \sigma_{unit\_price>=155}(PRODUCTS)$

- Products which are produced by Adidas: *JOIN + SELECTION*

  $R2 = Products\ (producer) \bowtie Producers\ (producer\_id)$

- Selection of the products produced by Adidas

  $R3 = \sigma_{name='Adidas'}(R2)$

- Building the same scheme:

  $R4 = \pi_{reference,designation}(R1)$

  $R5 = \pi_{reference,designation}(R3)$

- Union of the two tables:

  $R6 = R4 \cup R5$

3.4.1.Translation relational algebra → SQL Query Language

- *UNION*

R4

SELECT reference, designation
FROM products WHERE unit_price >= 155

UNION

R5

SELECT reference, designation
FROM products,producers
WHERE products.producer = producers.producer_id
AND name = 'Adidas';

3.4.1.Translation relational algebra → SQL Query Language

- *INTERSECTION*

$R1 = \sigma_{unit\_price >=155}(PRODUCTS)$

$R2 = Products(producer) \bowtie Producers\ (producer\_id)$

$R3 = \sigma_{Producers.name="Adidas"}\ (R2)$

$R4 = R1 \cap R3$

```
SELECT *
FROM products WHERE unit_price >= 155
INTERSECT
SELECT  *
FROM  products,producers
WHERE products.producer = producers.producer_id
AND name = 'Adidas';
```

3.4.1.Translation relational algebra → SQL Query Language

- *INTERSECTION*
- Non-standard (*e.g.* in ACCESS it doesn't exist)

⇒Can also be expressed using JOINS or NESTED QUERIES *(see later*).

3.4.1.Translation relational algebra → SQL Query Language

- *DIFFERENCE*

Example: Extracting the set of products which have never been ordered:

R1 = $\pi_{\text{(reference)}}$ (PRODUCTS).

R2 = $\pi_{\text{(reference)}}$ (ORDER_DETAILS)

R3 = R1 - R2

- SELECT reference FROM products

EXCEPT

select reference FROM order_details;

- N.B.: Non-standard keyword (*e.g.* in ORACLE it is called MINUS)

⇒ Can also be expressed using NESTED QUERIES (*see later*).

3.4.1.Translation relational algebra → SQL Query Language

- *Exercises: exercise list n°3*

Data Manipulation Language for querying: Simple SQL Retrieval statement

Expressing more refined **selections** in SQL

3.4.2. More refined selections using SQL

- Comparative operators for SQL **selections**:
  - =, !=, <>, <, >, <=, >= , IS NULL, IS NOT NULL
- Logic operation: NOT, AND, OR
- Other operation: BETWEEN, IN, LIKE
  - Digital / string/ date data type
    - attr **BETWEEN** val1 **AND** val2(⇔ (attr>=val1) and (attr<=val2))
    - attr **IN** (val1, val2, ...) (⇔ (attr=val1) or (attr=val2) or ...)
  - String data type: pattern matching
    - **LIKE**: _ instead of one character

      % instead of any characters (string)

    attr **LIKE** '_IT%'

    attr **LIKE** 'IT%'

3.4.2. More refined selections using SQL

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

[https://www.w3schools.com/sql/sql_like.asp]

3.4.2. More refined selections using SQL: pattern matching

student

student_id	first_name	last_name	dob	gender	address	note	program_id
20160001	Ngọc An	Bùi	3/18/1987	M	15 Lương Định Của,Đ. Đa, HN		20162101
20160002	Anh	Hoàng	5/20/1987	M	513 B8 KTX BKHN		20162101
20160003	Thu Hồng	Trần	6/6/1987	F	15 Trần Đại Nghĩa, HBT, Hà nội		20162101
20160004	Minh Anh	Nguyễn	5/20/1987	F	513 TT Phương Mai, Đ. Đa, HN		20162101
20170001	Nhật Ánh	Nguyễn	5/15/1988	F	214 B6 KTX BKHN		20172201
20170002	Nhật Cường	Nguyễn	10/24/1988	M	214 B5 KTX BKHN		20172201
20170003	Nhật Cường	Nguyễn	1/24/1988	M	214 B5 KTX BKHN		20172201
20170004	Minh Đức	Bùi	1/25/1988	M	214 B5 KTX BKHN		20172201

```
SELECT student_id, first_name, dob, address FROM student
WHERE address LIKE '%KTX%' AND gender = 'F';
```

result

student_id	first_name	last_name	dob	address
20170001	Nhật Ánh	Nguyễn	5/15/1988	214 B6 KTX BKHN

3.4.2. More refined selections using SQL: pattern matching

- Special character in the pattern: single quote ('), %, _
  - To retrieve single quotes (') ➔ use 2 single quotes: `title LIKE '%''%'`

```
SELECT * FROM subject
WHERE name LIKE '%''%';
```

result

subject_id	name	credit	….
LI0001	life's happy song	5	
LI0002	%life's happy song 2	5	

3.4.2. More refined selections using SQL: pattern matching

- Special character in the pattern: single quote ('), %, _
  - To retrieve symbols % or _ ➔ use escape characters: `title LIKE 'x%%x_' ESCAPE 'x'`

```
SELECT * FROM subject
WHERE name LIKE '\%%' ESCAPE '\';
```

Or, equivalently, in SQL Server:

```
SELECT * FROM subject
WHERE name LIKE '[%]%';
```

result

subject_id	name	credit	....
LI0002	%life's happy song 2	5	

3.4.2. More refined selections using SQL: NULL values

- Comparative operations with a NULL value:

  =, !=, <>, <, >, <=, >= with a NULL ➔ UNKNOWN

  (UNKNOWN: a truth-value as TRUE, FALSE)

- Check if an attribute has NULL value: IS NULL, IS NOT NULL

- Arithmetic operators : NULL is not a constant

  NULL +-/x any value ➔ NULL

  - If x is NULL then **x + 3 results NULL**

  - **NULL + 3** : not a legal SQL expression

  – N.B. Not only used for selections, can also be used for projections, aggregation functions, etc.

3.4.2. More refined selections using SQL: NULL values

- Comparative operations: with a NULL → UNKNOWN
- Logic operation: AND ~MIN, OR ~MAX, NOT(x) ~ 1-x

X	Y	X AND Y Y AND X	X OR Y Y OR X	NOT Y
UNKNOWN	TRUE	UNKNOWN	TRUE	FALSE
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	TRUE

- Conditions in WHERE clauses apply on each tuples of some relation

→ Only the tuples for which the condition has the **TRUE** value become part of the answer

3.4.2. More refined selections using SQL: NULL values

- Example:

subject

subject_id	name	credit	per..
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	

```
SELECT * FROM subject
WHERE credit >= 4 AND
percentage_final_exam <= 60;
```

result

subject_id	name	credit	per..
IT1110	Tin học đại cương	4	60

```
SELECT * FROM subject
WHERE percentage_final_exam = NULL;
```

result

subject_id	name	credit	....

```
SELECT * FROM subject
WHERE percentage_final_exam IS NULL;
```

result

subject_id	name	credit	per..
LI0001	life's happy song	5	
LI0002	%life's happy song 2	5	

Remark

- Each DBMS has its own implementation. So the syntax for each statement can vary from one database system to another:
  - Meaning of special characters used (%, _, *, ", '),
  - less or more options
  - standard part & extension part
- More options for each statement: see documentations of the DBMS used in your system

Data Manipulation Language for querying: Simple SQL Retrieval statement

Expressing more refined **projections** in SQL

3.4.3. More refined projections using SQL: sorting results

- *SORTING*

```
SELECT *
FROM products
WHERE unit_price >= 155
AND  producer = 1369
ORDER BY reference ASC;
```

The default option is ASC.

<u>The option ORDER is always at the end of the query</u>

3.4.3. More refined projections using SQL

- Renaming output attributes
- Syntax:

  SELECT <col_name> **AS** <alias_name>, <expr> **AS** <alias_name>…

  FROM … **WHERE** ...

- Example:

```
SELECT subject_id AS id, name,
    credit "ETC"
FROM subject;
```

- Keyword **AS**: optional
  - <alias_name>: used in ORDER BY clause,
  - <alias_name>: not used in WHERE or HAVING clauses

Subject

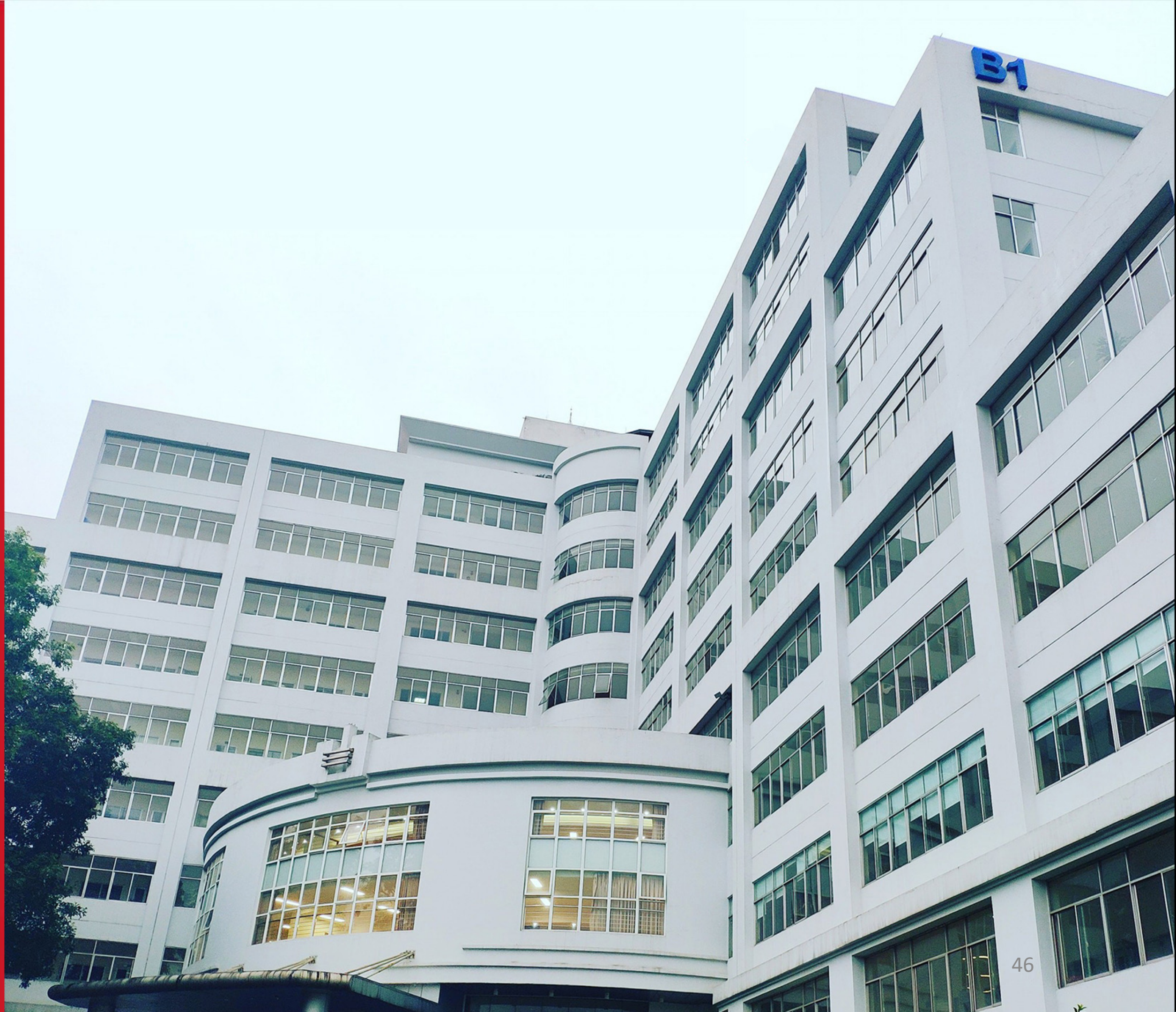
id	name	Credit
IT1110	Tin học đại cương	4
IT3080	Mạng máy tính	3
IT3090	Cơ sở dữ liệu	3
IT4857	Thị giác máy tính	3
IT4866	Học máy	2
LI0001	life's happy song	5
LI0002	%life's happy song 2	5

ĐẠI HỌC BÁCH KHOA

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/ fb.com/groups/soict