

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 1*

Muriel VISANI

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

- Introduction to SQL
- Defining a Relational schema
- Data Manipulation

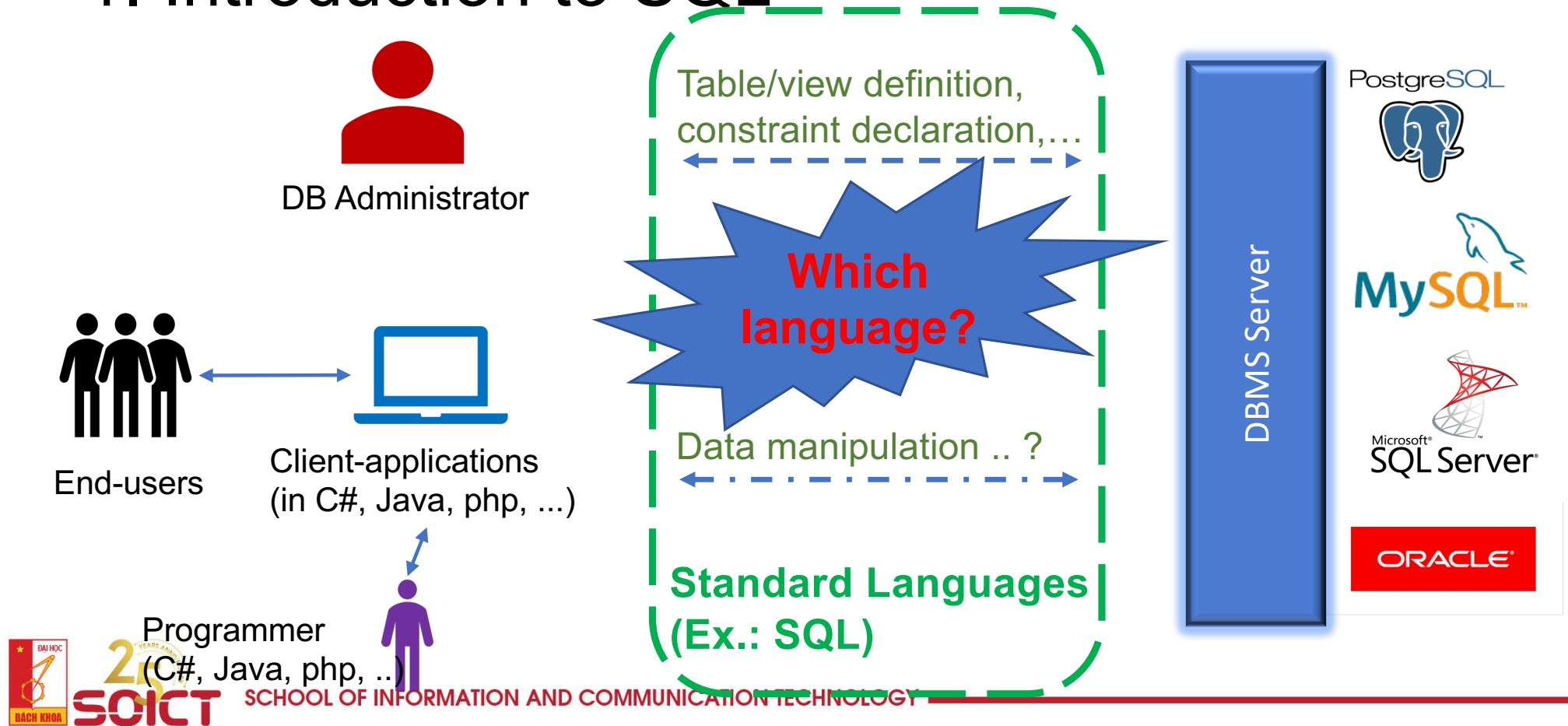
Learning objective

- Have notions about the **SQL language**
- Use SQL to **define a relational schema** in a database
- Use SQL to:
 - Create / modify / drop tables
 - Create / modify / drop constraints
 - Insert / update / delete rows in/from a table

Keywords

Keyword	Description
DBMS	Database Management System: system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data
CREATE TABLE	SQL statement to define a table into a database
ALTER TABLE	SQL statement to modify table structure if needed (add /delete/modify column(s), add/remove constraint(s))
INSERT/UPDATE/DELETE	SQL statements to add new record to a table; to change the data of one or more records in a table; to remove single record or multiple records from a table
SELECT	SQL statement to retrieve data from a database

1. Introduction to SQL



1.1. Brief history of SQL

- 1975: SEQUEL: System-R
- 1976: SEQUEL 2
- 1978/79: SQL (Structured Query Language) (used in System-R)
- SQL1: The first standard for SQL defined in 1986; adopted as an international by Standards Organisation (ISO) in 1987.
- 1992: SQL2 - revised version of the processor (also called SQL 92); adopted as the formal standard language for defining and manipulating relational database.
- 1999: SQL3 - extension with additional features such as user-defined data types, triggers, user-defined functions and other Object Oriented features.
- New versions of the standard were published in 2003, 2006, 2008, 2011, 2016: more additional features: XML-based features, columns with auto-generated values, JSON,...

1.2. Languages

- Data Definition Language (DDL)
 - Define/update the logical schema (tables, constraints...), and the storage schema stored in a Data Dictionary
- Data Manipulation Language (DML)
 - Populate the tables, update / delete the records
 - Querying the content of a database for retrieving information from it
- Data Control Language (DCL)
 - permissions, access control...

1.2. Languages

Focus of this lecture

- Data Definition Language (DDL)
 - Define/update the logical schema (tables, constraints...), and the storage schema stored in a Data Dictionary
- Data Manipulation Language (DML)
 - Populate the tables, update / delete the records
 - Querying the content of a database for retrieving information from it
- Data Control Language (DCL)
 - permissions, access control...

2. Definition of a Relational Schema

- Example: Education database

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)

- Detailed description for the relation (table) **scores_per_subject**

Attribute name	Type	NOT NULL	Description
student_id	CHAR(8)	Yes	Student identification code. FOREIGN KEY references to Student(student_id)
subject_id	CHAR(6)	Yes	Subject code. FOREIGN KEY references to Subject(subject_id)
semester	CHAR(5)	Yes	Annual semester: '20171', '20172', '20173', ...
midterm_score	Float	No	Score of mid-term exam. DOM = [0,10] and (midtermScore mod 0.5) must be 0
final_score	Float	No	Score of final exam. DOM= [0,10] and (finalScore mod 0.5) must be 0
PRIMARY KEY = {student_id, subject_id, semester}			

2.1. Creating a Simple Table

- Syntax:

```
CREATE TABLE <table_name>(
    <col1> <type1>(<size1>) [NOT NULL] [DEFAULT <value>],
    <col2> <type2>(<size2>) [NOT NULL], ...,
    [[CONSTRAINT <constraint_name>] <constraint_type> clause], ...);
```

- Example:

```
CREATE TABLE student(
    student_id CHAR(8),
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    dob DATE,
    gender CHAR(6),
    address VARCHAR(50),
    email VARCHAR(30) NOT NULL,
    program_id CHAR(7) NOT NULL,
    PRIMARY KEY(student_id) );
```

2.1. Creating a Simple Table: Naming conventions

- Ordinary identifiers
 - Must begin with a letter
 - Contain only: letters with NO ACCENT / TONE (a...z), underscore (_), and digits (0...9)
 - No longer than 32 characters
- Delimited identifiers (rare)
 - Identifiers surrounded **by double quotation marks (")**
 - Can contain any characters

2.1. Creating a Simple Table: Naming conventions [2]

- Have meaning, not so long, use common abbreviations if needed:
 - use `student`, `firstname`;
 - Do not use `table1`, `abc`, `fw12re`, `student_of_the_school...`
- Avoid quotes: `student` ; not "Student" or "All Students"
- Use lowercase, underscores to separate words:
 - Use `firstname` / `first_name`;
 - Do not use "firstName" (because capital letters are not taken into account by most DBMS...)
- Avoid reserved words (keywords) for naming objects:
 - Do not use use data types such as **text**, **integer**, ... as object names
 - Do not use use **table**, **user**, ... as object names
- Tables / Views should have singular names, not plural:
 - Convention that is preferred especially for the sessions of practicals



`student` (but not students)

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

2.1. Creating a Simple Table: Data Types (SQL 92)

boolean	logical boolean (true/false)
character(n)	fixed-length character string
varchar(n)	variable-length character string
smallint	signed two-byte integer
int, integer	signed 4-byte integer
float(p)	floating-point number with precision p
real, double precision	double-precision floating-point number
decimal(p,s), numeric(p,s)	user-specified precision, exact; recommended for storing monetary amounts p: number of digits in the whole number, s: number of digits after the decimal point.
date	calendar date without time of day
time	time of day
timestamp with time zone	date/time

2.1. Creating a Simple Table: Data Types (SQL 92)

FLOAT(p)

The FLOAT data type accepts approximate numeric values, for which you may define a precision up to a maximum of 64. If no precision is specified during the declaration, the default precision is 64. Attempting to assign a value larger than the declared precision will cause an error to be raised.

Examples

FLOAT(8)

- Valid

12345678
1.2
123.45678
-12345678
-1.2
-123.45678

- Invalid

123456789
123.456789
-123456789
-123.456789

2.1. Creating a Simple Table: Data Types (SQL 92)

DECIMAL [(p[,s])] or DEC [(p[,s])]

The DECIMAL data type accepts numeric values, for which you may define a precision and a scale in the data type declaration. The precision is a positive integer that indicates the number of digits that the number will contain. The scale is a positive integer that indicates the number of these digits that will represent decimal places to the right of the decimal point. The scale for a DECIMAL cannot be larger than the precision.

DECIMAL data types can be declared in one of three different ways. The declaration of it controls how the number is presented to an SQL query, but not how it is stored.

- DECIMAL - Precision defaults to 38, Scale defaults to 0
- DECIMAL(p) - Scale defaults to 0
- DECIMAL(p, s) - Precision and Scale are defined by the user

In the above examples, *p* is an integer representing the precision and *s* is an integer representing the scale.

NOTE: If you exceed the number of digits expected to the left of the decimal point, an error is thrown. If you exceed the number of expected digits to the right of the decimal point, the extra digits are truncated.

Examples

DECIMAL(10,3)

- Valid

```
1234567
1234567.123
1234567.1234 (Final digit is truncated)
-1234567
-1234567.123
-1234567.1234 (Final digit is truncated)
```

- Invalid

```
12345678
12345678.12
12345678.123
-12345678
-12345678.12
-12345678.123
```



2.1. Creating a Simple Table: Data Types (SQL 92)

- Example: Education database

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)

- Questions:
 - What should be the type of the attribute *credit* in the table SUBJECT?
 -
 - What should be the type of the attribute *percentage_final_exam* in the table SUBJECT?
 -

2.1. Creating a Simple Table: NOT NULL, Default value

- NOT NULL

- The NOT NULL attributes MUST have a known value
- By default, if an attribute is not declared as NOT NULL, then it can receive NULL values...
- Primary Keys are by default NOT NULL; if we correctly declare them, we don't need to add NOT NULL constraint (but if we do, it's no problem)

- Default value

- Enables to specify the value that will be assigned by default for an attribute, if no other value is given when inserting the data

2.1. Creating a Simple Table: NOT NULL, Default value

- Example: Education database

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)

- Questions:
 - Which attributes really need to be declared as NOT NULL?
 - Give the corresponding extract of the CREATE TABLE code
 - Give one attribute that could assigned a default value (and which default value could be used)
 - Give the corresponding extract of the CREATE TABLE code

2.1. Creating a Simple Table: NOT NULL, Default value

- Solutions:
 - Which attributes really need to be declared as NOT NULL?
 -

2.1. Creating a Simple Table: NOT NULL, Default value

- Solutions:
 - Give the example of one attribute that could assigned a default value (and which default value could be used)
 -

2.1. Creating a Simple Table: NOT NULL, Default value

- Question:
 - Why do we declare default values for some attributes?

•

2.2. Constraints

- Entity Integrity:
 - PRIMARY KEY constraint
 - UNIQUE constraint
 - Valid values on an attribute or between attributes in a tuple: CHECK constraint
- Referential Integrity:
 - FOREIGN KEY constraint

2.2. Constraints

- Entity Integrity:
 - **PRIMARY KEY** constraint
 - UNIQUE constraint
 - Valid values on an attribute or between attributes in a tuple: CHECK constraint
- Referential Integrity:
 - FOREIGN KEY constraint

2.2. Constraints: PRIMARY KEY

- Syntax:

[**CONSTRAINT** <constraint_name>] **PRIMARY KEY** (<fk1>, <fk2>, ...)

- A relation can have **only one primary key**

Table: *student*(*student_id*, *first_name*, *last_name*, *dob*, *gender*, *address*, *email*, *program_id#*)

SQL:

- **CREATE TABLE** *student* (
 student_id **CHAR**(8) ,
 first_name **VARCHAR**(20) **NOT NULL**,
 last_name **VARCHAR**(20) **NOT NULL**,
 dob **DATE**,
 gender **CHAR**(6) ,
 address **VARCHAR**(50) ,
 email **VARCHAR**(30) **NOT NULL**,
 program_id **CHAR**(7) **NOT NULL**,
 PRIMARY KEY(*student_id*));

2.2. Constraints: PRIMARY KEY [2]

- Other possible SQL declarations:

SQL:

```
CREATE TABLE student(  
    student_id CHAR(8) ,  
    first_name VARCHAR(20) NOT NULL,  
    ...  
    CONSTRAINT student_pk PRIMARY KEY(student_id)  
);
```

Name of the constraint

```
CREATE TABLE student(  
    student_id CHAR(8) PRIMARY KEY ,  
    first_name VARCHAR(20) NOT NULL,  
    ...  
);
```

Only if the PK
is **one**
attribute

2.2. Constraints: PRIMARY KEY [2]

- Question 1:
 - According to you, why is it mostly useful to name yourself your constraints?

2.2. Constraints: PRIMARY KEY [2]

- Question 2:
 - According to you, why can't we use this type of declarations when the PK is composed of multiple attributes?

```
CREATE TABLE student (  
    student_id CHAR(8) PRIMARY KEY,  
    first_name VARCHAR(20) NOT NULL,  
    ...  
);
```

2.2. Constraints

- Entity Integrity:
 - PRIMARY KEY constraint
 - **UNIQUE constraint**
 - Valid values on an attribute or between attributes in a tuple: CHECK constraint
- Referential Integrity:
 - FOREIGN KEY constraint

2.2. Constraints: UNIQUE

- Several attributes can be unique in a table
- A unique attribute is not necessary NOT NULL
- A combination of attributes can also be unique
- For primary keys, it is not necessary to declare that they are unique
 - It's implied (by definition of the PK)
- Question:
 - What kind of key is an attribute which is both unique and not null?
 -

2.2. Constraints: UNIQUE

- Question:
 - Give an example of ONE attribute that should be UNIQUE in our schema

2.2. Constraints

- Entity Integrity:
 - PRIMARY KEY constraint
 - UNIQUE constraint
 - Valid values on an attribute or between attributes in a tuple: CHECK constraint
- Referential Integrity:
 - FOREIGN KEY constraint

2.2. Constraints: CHECK

- Syntax:

[**CONSTRAINT** <constraint_name>] **CHECK** <condition>

- Declaring check constraint when defining table

Tables: student(student_id, first_name, last_name, dob, gender, address, email, program_id#)

SQL: **CREATE TABLE** student (
 student_id **CHAR**(8),
 ...
 CONSTRAINT student_pk **PRIMARY KEY** (student_id),
 CONSTRAINT student_unique_email **UNIQUE**(email),
 CONSTRAINT student_chk_gender **CHECK** (gender='Female' **OR** gender='Male')
);

2.2. Constraints: CHECK

- Important remarks:
 - CHECK constraints do not fail when the value is NULL
 - One can use the keywords **AND**, **OR** in CHECK constraints
 - Depending on the DBMS, more refined constraints can be expressed using the keywords LIKE, BETWEEN...
 - This will be studied mainly during the practicals
 - Question: which other attribute would you recommend adding a CHECK constraint on?
 -

2.2. Constraints

- Entity Integrity:
 - PRIMARY KEY constraint
 - UNIQUE constraint
 - Valid values on an attribute or between attributes in a tuple: CHECK constraint
- Referential Integrity:
 - FOREIGN KEY constraint

2.2. Constraints: FOREIGN KEY

- Syntax:

```
[CONSTRAINT <constraint_name>] FOREIGN KEY (<fk1>,<fk2>,...)  
    REFERENCES <tab>(<k1>,<k2>, ...)  
    [ON UPDATE <option>] [ON DELETE <option>]
```

- Options:

- **CASCADE**

- Delete/update all tuples with matching foreign key values, if the table attribute value is deleted/updated in the parent table
 - I **personally** recommend you NOT to use the CASCADE option until you are very confident with databases

- **NO ACTION / RESTRICT** (syntax depending on the DBMS)

- can't delete primary key tuple whilst a foreign key tuple matches
 - default action -> will return an error, but you might be happy to see the error realize that you were about to make a mistake ☺ -> my personal recommendation

- **SET NULL:** if a record in the parent table is deleted, then the corresponding records in the child table will have the

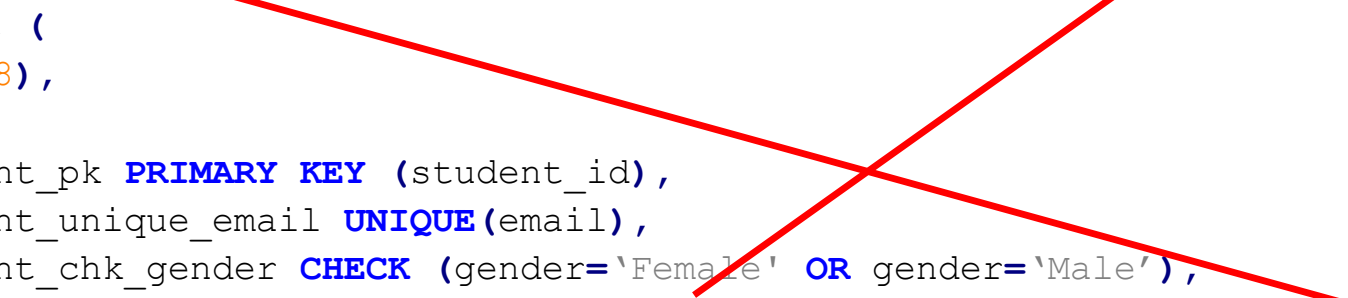
foreign key fields set to NULL -> still a loss of information, I personally would not recommend it in general

2.2. Constraints: FOREIGN KEY

- Declaring a foreign key constraint when defining a table

Table: `student(student_id, first_name, last_name, dob, gender, address, email, program_id#)`
`program(program_id, name, lecturer_in_charge#)`

SQL: **CREATE TABLE** student (
 student_id **CHAR**(8),
 ...
 CONSTRAINT student_pk **PRIMARY KEY** (student_id),
 CONSTRAINT student_unique_email **UNIQUE** (email),
 CONSTRAINT student_chk_gender **CHECK** (gender='Female' **OR** gender='Male'),
 CONSTRAINT student_fk_program **FOREIGN KEY** (program_id) **REFERENCES** program(program_id));



The diagram consists of two red arrows. One arrow originates from the `program_id` attribute in the `student` table definition and points to the `program_id` attribute in the `program` table definition. The second arrow originates from the `program_id` attribute in the `FOREIGN KEY clause of the SQL statement and points to the program_id attribute in the program table definition.`

2.3. Modifying Relation Schema: Columns

- Add column(s)

```
ALTER TABLE <table_name> ADD COLUMN  
<column_name> <datatype> [NOT NULL] [DEFAULT <default_value>;
```

- Delete column(s)

```
ALTER TABLE <table_name> DROP COLUMN <column_name>;
```

- Modify column(s)

```
ALTER TABLE <table_name> ALTER COLUMN <column_name> <datatype>;
```

- Examples:

```
ALTER TABLE student ADD COLUMN  
emergency_contact CHAR(15) DEFAULT '(+84) 000-000-000';
```

```
ALTER TABLE student DROP COLUMN emergency_contact;
```

2.3. Modifying Relation Schema: Constraints

- Add new constraint(s)

```
ALTER TABLE <table_name>  
ADD CONSTRAINT <constraint_name> <constraint_type> clause;
```

Example:

```
ALTER TABLE student ADD CONSTRAINT student_fk_program  
FOREIGN KEY (program_id) REFERENCES program(program_id);
```

- Delete existing constraints

```
ALTER TABLE <table_name> DROP CONSTRAINT <constraint_name>;
```

Example:

```
ALTER TABLE student DROP CONSTRAINT student_fk_program;
```


2.3. Modifying Relation Schema: Constraints

- N.B. If you don't name your constraints, then the DBMS will do it for you
 - According to some naming convention that depends on the DBMS you use
 - Usually, you can see the constraint list in some window of the DBMS
 - So, you can still drop these constraints by using their names if needed...
- ... But, as explained earlier, it's always best to name your constraints yourself!
 - Example of naming convention (many conventions are possible!):
 - `pk_<table_name>_<attribute_name>_<domain>`
 - `pk_<table_name>_<attribute_name>`
 - `fk_<referencing_table_parent_table>_<referencing_attribute>`
 - ...

2.4. Drop a Relation from Database

- Syntax: **DROP TABLE** <table_name> [**CASCADE** | **RESTRICT**] ;
 - **CASCADE**: allows to remove all dependent objects together with the table automatically
 - I **personally** recommend you NOT to use the CASCADE option until you are very confident with databases
 - **RESTRICT**: refuses to drop table if there is any object depends on it; default value

2.4. Drop a Relation from Database

- Example:

```
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)
```

```
DROP TABLE program;
```

```
ERROR: cannot drop table program because other objects depend on it
```

```
DROP TABLE subject CASCADE;
```

```
NOTICE: drop cascades to 2 other objects
```

3. Data Manipulation

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

Modifying address?

Adding new student / new program?

Deleting student data?

Retrieving list of all students?

program

program_id	name	lecturer_in_charge
20162101	CNTT1.01-K61	02001
20162102	CNTT1.02-K61	
20172201	CNTT2.01-K62	02002
20172202	CNTT2.02-K62	



SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

3.1. Insertion

- Syntax:

```
INSERT INTO <table1>[(<col1>,<col2>,...)] VALUES (<exp1>,<exp2>,...);
```

- Examples (if the column lecturer_in_charge is not NOT NULL):

```
INSERT INTO program(program_id, name) VALUES ('20162101', 'CNTT1.01-K61');
```

```
INSERT INTO program(name, program_id) VALUES ('CNTT2.02-K62', '20172202');
```

```
INSERT INTO program VALUES ('20172201', 'CNTT2.01-K62', NULL);
```

- One can also use data from other tables to insert in a table:

```
INSERT INTO <table1>[(<col1>,<col2>,...)]
```

```
SELECT <col1>, <col2>, ...
```

```
FROM <tab1>, <tab2>, ...
```

```
WHERE <condition>;
```

3.2. Deletion, Update

- Deletion:

```
DELETE FROM <table_name> [WHERE <condition>];
```

```
DELETE FROM student WHERE student_id = '20160002';
```

- Update:

```
UPDATE    <table_name>  
SET <col1> = <exp1>,  
      <col2> = <exp2>, ...  
[WHERE    <condition>];
```

```
UPDATE    student  
SET address = '179 Le Thanh Nghi, HBT, HN'  
WHERE    student_id = '20170003';
```



SOICT

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

3.3. Examples of errors due to constraint violations

```
INSERT INTO program VALUES ('20172201', 'CNTT3.01-K62', NULL);
```

ERROR: duplicate key value violates unique constraint "program_pk"

DETAIL: Key (program_id)=(20172201) already exists. SQL state: 23505

```
UPDATE program SET lecturer_in_charge = '20160022' WHERE program_id = '20162102';
```

ERROR: insert or update on table "program" violates foreign key constraint "program_fk_lecturer"

DETAIL: Key (lecturer_in_charge)=(20160022) is not present in table "lecturer". SQL state: 23503

```
DELETE FROM program WHERE program_id = '20162101';
```

ERROR: update or delete on table "program" violates foreign key constraint "student_fk_program" on table "student" DETAIL: Key (program_id)=(20162101) is still referenced from table "student". SQL state: 23503

```
UPDATE student SET gender = 'Woman' WHERE student_id = '20160003';
```

ERROR: new row for relation "student" violates check constraint "student_chk_gender"

DETAIL: Failing row contains (20160003, Thu Hồng, Trần, 1987-06-06, Woman, 15 Trần Đại Nghĩa, HBT, Hà nội, null, 20162101). SQL state: 23514

Exercise

- Make the exercise list number 2





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

