CMSC 508 Databases

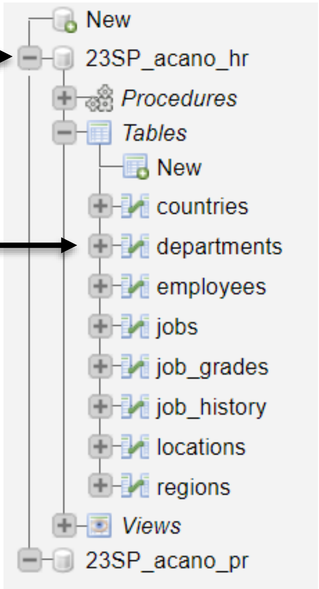
Introduction to SQL (I)

Dr. Alberto Cano
Associate Professor
Department of Computer Science

VCU College of Engineering

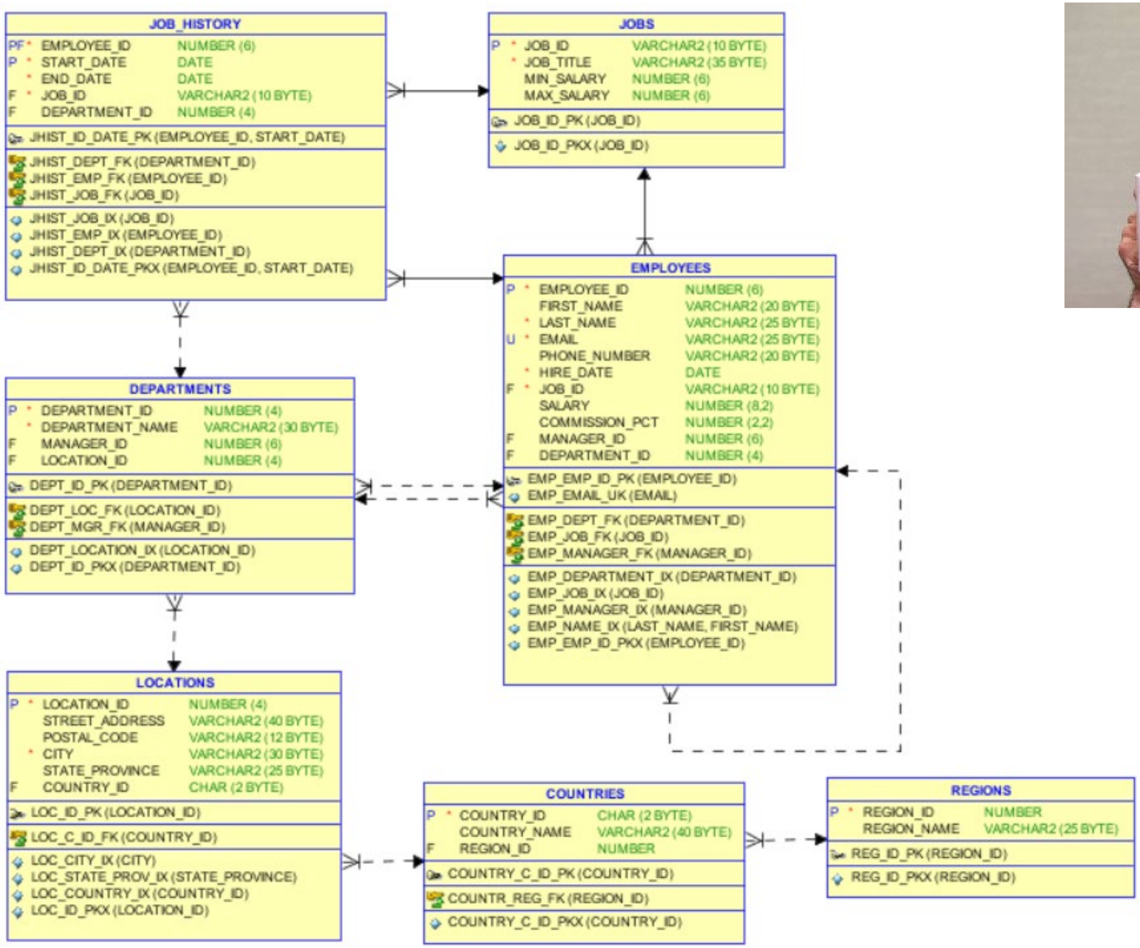
Chapter 3 from Database System Concepts, 7th Ed. by Silberschatz, Korth, Sudarshan, 2019
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003

- Linux + Apache + MySQL + PHP server
  - MySQL
  - phpMyAdmin https://www.cmsc508.com/phpMyAdmin/
  - To use any other client, use the connection information:
    - **Host:** cmsc508.com
    - **Username**: 23SP_*VCUEID* (e.g. 23SP_acano)
    - **Password**: same as the username
  - There are two databases for each user:
    - 23SP_*VCUEID*_hr -> for learning SQL in class and the tests
    - 23SP_*VCUEID*_pr -> for developing the course project



- Linux + Apache + MySQL + PHP server
  - For developing the code of the project you will need to deploy your website on the server
  - Linux
    - ssh 23SP_*VCUEID*@cmsc508.com -p 7822 (same 23SP_*VCUEID* passwd)
    - Use Filezilla to connect and transfer files to your **public_html** folder
  - Apache
    - Visit https://www.cmsc508.com/~23SP_VCUEID
  - PHP 8 + MySQL + JavaScript + jQuery + Bootstrap + D3.js FTW

Human Resources Database



JOB_HISTORY

PF	*	EMPLOYEE_ID	NUMBER (6)
P	*	START_DATE	DATE
	*	END_DATE	DATE
F	*	JOB_ID	VARCHAR2 (10 BYTE)
F		DEPARTMENT_ID	NUMBER (4)

JHIST_ID_DATE_PK (EMPLOYEE_ID, START_DATE)

JHIST_DEPT_FK (DEPARTMENT_ID)
JHIST_EMP_FK (EMPLOYEE_ID)
JHIST_JOB_FK (JOB_ID)

JHIST_JOB_IX (JOB_ID)
JHIST_EMP_IX (EMPLOYEE_ID)
JHIST_DEPT_IX (DEPARTMENT_ID)
JHIST_ID_DATE_PKX (EMPLOYEE_ID, START_DATE)

JOBS

P	*	JOB_ID	VARCHAR2 (10 BYTE)
	*	JOB_TITLE	VARCHAR2 (35 BYTE)
		MIN_SALARY	NUMBER (6)
		MAX_SALARY	NUMBER (6)

JOB_ID_PK (JOB_ID)

JOB_ID_PKX (JOB_ID)

EMPLOYEES

P	*	EMPLOYEE_ID	NUMBER (6)
		FIRST_NAME	VARCHAR2 (20 BYTE)
	*	LAST_NAME	VARCHAR2 (25 BYTE)
U	*	EMAIL	VARCHAR2 (25 BYTE)
		PHONE_NUMBER	VARCHAR2 (20 BYTE)
	*	HIRE_DATE	DATE
F	*	JOB_ID	VARCHAR2 (10 BYTE)
		SALARY	NUMBER (8,2)
		COMMISSION_PCT	NUMBER (2,2)
F		MANAGER_ID	NUMBER (6)
F		DEPARTMENT_ID	NUMBER (4)

EMP_EMP_ID_PK (EMPLOYEE_ID)
EMP_EMAIL_UK (EMAIL)

EMP_DEPT_FK (DEPARTMENT_ID)
EMP_JOB_FK (JOB_ID)
EMP_MANAGER_FK (MANAGER_ID)

EMP_DEPARTMENT_IX (DEPARTMENT_ID)
EMP_JOB_IX (JOB_ID)
EMP_MANAGER_IX (MANAGER_ID)
EMP_NAME_IX (LAST_NAME, FIRST_NAME)
EMP_EMP_ID_PKX (EMPLOYEE_ID)

DEPARTMENTS

P	*	DEPARTMENT_ID	NUMBER (4)
	*	DEPARTMENT_NAME	VARCHAR2 (30 BYTE)
F		MANAGER_ID	NUMBER (6)
F		LOCATION_ID	NUMBER (4)

DEPT_ID_PK (DEPARTMENT_ID)

DEPT_LOC_FK (LOCATION_ID)
DEPT_MGR_FK (MANAGER_ID)

DEPT_LOCATION_IX (LOCATION_ID)
DEPT_ID_PKX (DEPARTMENT_ID)

LOCATIONS

P	*	LOCATION_ID	NUMBER (4)
		STREET_ADDRESS	VARCHAR2 (40 BYTE)
		POSTAL_CODE	VARCHAR2 (12 BYTE)
	*	CITY	VARCHAR2 (30 BYTE)
		STATE_PROVINCE	VARCHAR2 (25 BYTE)
F		COUNTRY_ID	CHAR (2 BYTE)

LOC_ID_PK (LOCATION_ID)

LOC_C_ID_FK (COUNTRY_ID)

LOC_CITY_IX (CITY)
LOC_STATE_PROV_IX (STATE_PROVINCE)
LOC_COUNTRY_IX (COUNTRY_ID)
LOC_ID_PKX (LOCATION_ID)

COUNTRIES

P	*	COUNTRY_ID	CHAR (2 BYTE)
		COUNTRY_NAME	VARCHAR2 (40 BYTE)
F		REGION_ID	NUMBER

COUNTRY_C_ID_PK (COUNTRY_ID)

COUNTR_REG_FK (REGION_ID)

COUNTRY_C_ID_PKX (COUNTRY_ID)

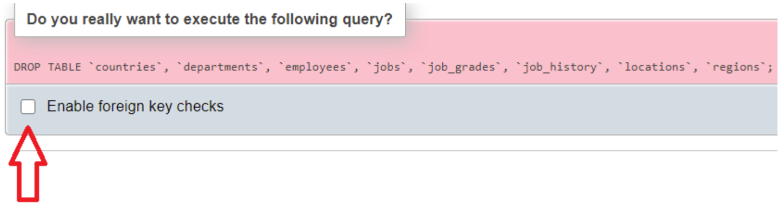
REGIONS

P	*	REGION_ID	NUMBER
		REGION_NAME	VARCHAR2 (25 BYTE)

REG_ID_PK (REGION_ID)

REG_ID_PKX (REGION_ID)

- Human Resources Database
  - DO **NOT** make permanent changes to the tables of the database. If you do, the results of the queries for the exercises may change.
  - If you need to rebuild the original database:
    - Drop the current content (procedures, views, tables (in that order) and make sure to **untick** “Enable foreign key checks”



Do you really want to execute the following query?

```
DROP TABLE `countries`, `departments`, `employees`, `jobs`, `job_grades`, `job_history`, `locations`, `regions`;
```

☐ Enable foreign key checks

    - Rebuild the original content using this script. Go to the “Import” tab, select the .sql file and import the data.
  - Finally, refresh the panel on the left and you will see the tables recreated

- Data Definition Language
  - The SQL data definition language (DDL) allows the specification of information about relations, including:
    - The schema for each relation
    - The domain of values associated with each attribute
    - Integrity constraints
    - Other information such as:
      - The set of indices to be maintained for each relation
      - Security and authorization information for each relation
      - The physical storage structure of each relation on disk

- Data types (MySQL)
  - CHAR(size)

    The length of a CHAR column is **fixed** to the length that you declare when you create the table. The length can be any value from 0 to 255.

  - VARCHAR(size)

    Values in VARCHAR columns are **variable-length** strings. The length can be specified as a value from 0 to 65535.

String	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

- Data types (MySQL)
  - Integer: INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

  - Floating-point: DECIMAL/NUMERIC, FLOAT, DOUBLE
    - NUMERIC(precision, scale) e.g. NUMERIC(5,2) ranges -999.9 to 999.99
    - FLOAT 32-bit - DOUBLE 64-bit

- Data types (MySQL)
  - Date and Time: DATE, DATETIME, TIMESTAMP, TIME, YEAR
    - DATE is used for dates in 'YYYY-MM-DD' format
    - DATETIME is used for dates and time in 'YYYY-MM-DD hh:mm:ss' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
    - TIMESTAMP is used for dates and time in 'YYYY-MM-DD hh:mm:ss' format. Has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC. **Self-updates** when rows are edited.
    - TIME in 'hh:mm:ss' format
    - YEAR in 'YYYY' format

Data Type	**"Zero" Value**
DATE	'0000-00-00'
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	'0000-00-00 00:00:00'
YEAR	0000

- Data types (MySQL)
  - BLOB

    A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the values they can hold.
  - TEXT

    TEXT values are treated as nonbinary strings (character strings). The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT.

    It’s **not** recommended to unnecessarily increase the database with huge items. Preferred to externalize content and reference using a link (e.g. path to file)

- CREATE TABLE
  - A SQL relation is defined using the CREATE TABLE command:

    CREATE TABLE *relation* (C_1 D_1, C_2 D_2, ..., C_n D_n,
    (integrity-constraint$_1$), ..., (integrity-constraint$_k$))

    - *relation* is the name of the table
    - C_i is the i-th column name
    - D_i is the data type of values in the domain of column C_i
  - Constraints
    - AUTO_INCREMENT
    - NOT NULL
    - UNIQUE
    - CHECK (predicate)
    - PRIMARY KEY (Cn, ..., Cm)
    - FOREIGN KEY (Cj, ..., Ck) REFERENCES relation([Cq, ..., Cw])

    *Referenced foreign keys **must** be on an indexed attribute

Main data types
- CHAR(size)
- VARCHAR(size)
- INT
- FLOAT
- DECIMAL(P,S)
- DATE
- DATETIME
- TIMESTAMP

- CREATE TABLE (MySQL Reference Manual)

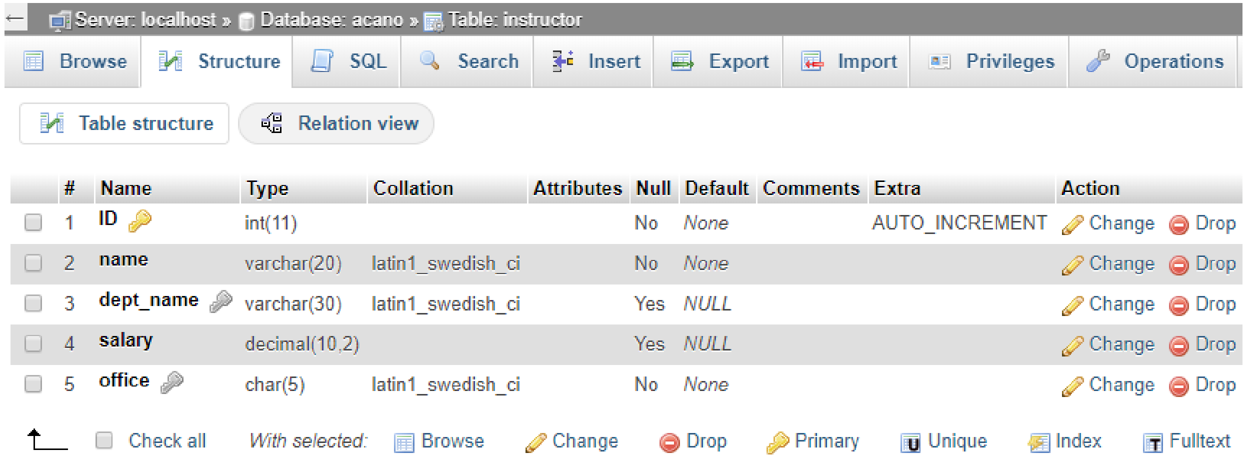
CREATE TABLE *instructor* (
    ID   **int AUTO_INCREMENT**,
    name   **varchar**(20) **NOT NULL**,
    dept_name   **varchar**(30),
    salary   **decimal**(10,2) **CHECK** (salary > 0),
    office   **char**(5) **NOT NULL UNIQUE**,
    PRIMARY KEY (ID),
    FOREIGN KEY (dept_name) **REFERENCES** departments (department_name)
);

- ALTER TABLE (MySQL Reference Manual)

ALTER TABLE instructor **ADD** foocolumn VARCHAR(25);

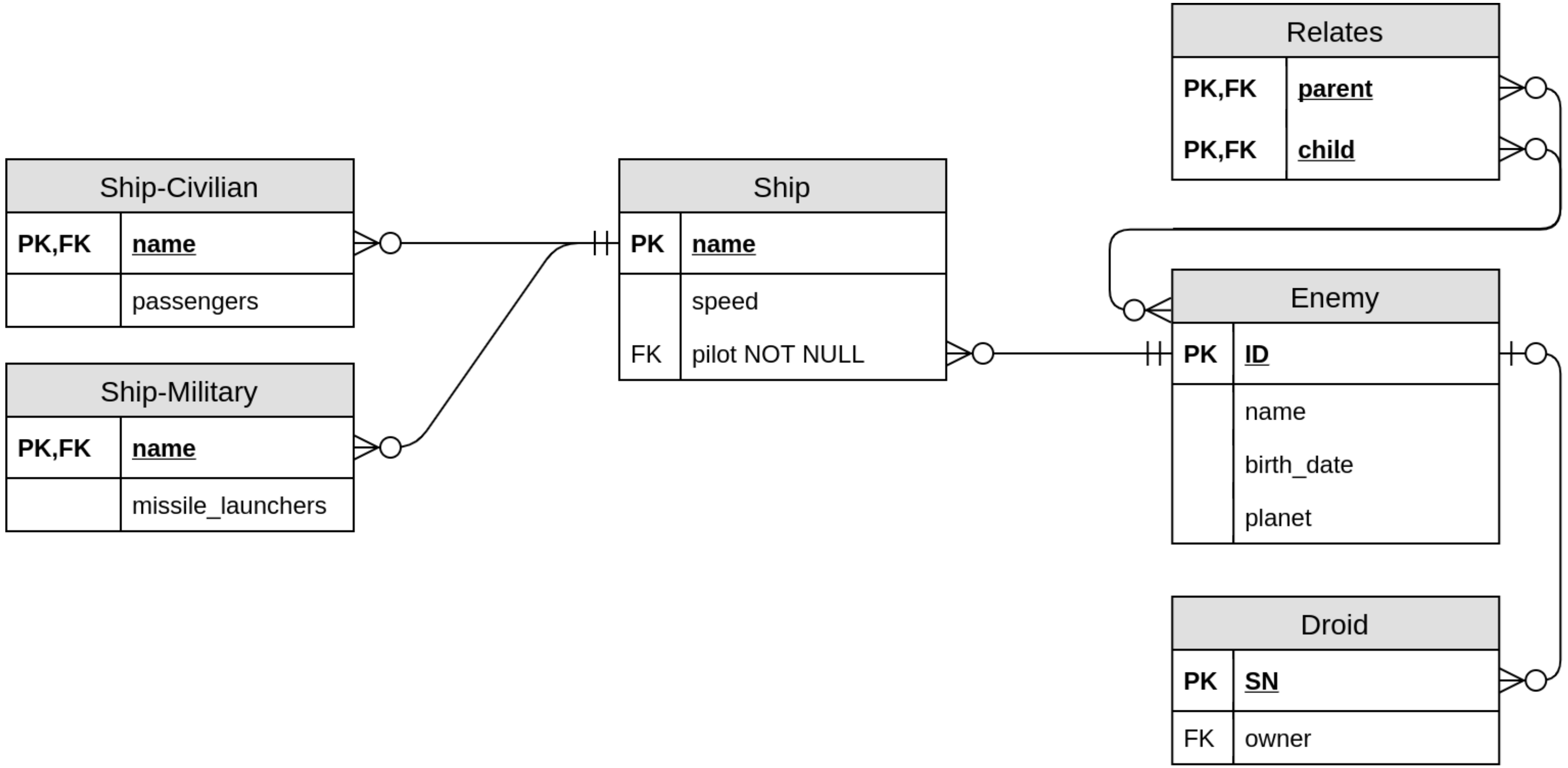
ALTER TABLE instructor **DROP COLUMN** foocolumn;

phpMyAdmin GUI: Structure tab



	#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
☐	1	ID	int(11)			No	*None*		AUTO_INCREMENT	Change Drop
☐	2	name	varchar(20)	latin1_swedish_ci		No	*None*			Change Drop
☐	3	dept_name	varchar(30)	latin1_swedish_ci		Yes	*NULL*			Change Drop
☐	4	salary	decimal(10,2)			Yes	*NULL*			Change Drop
☐	5	office	char(5)	latin1_swedish_ci		No	*None*			Change Drop

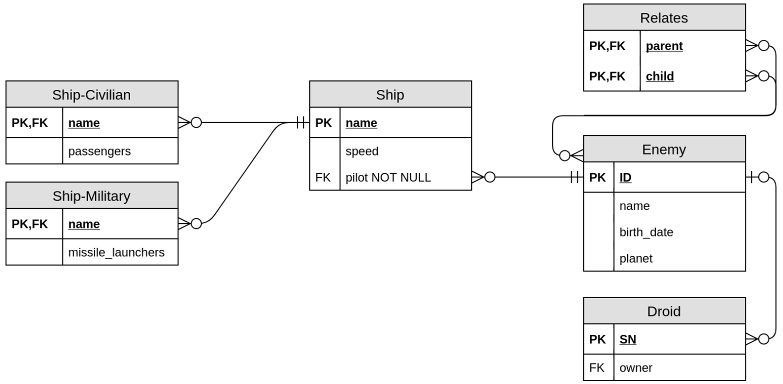
- Remember the Star Wars relational diagram?



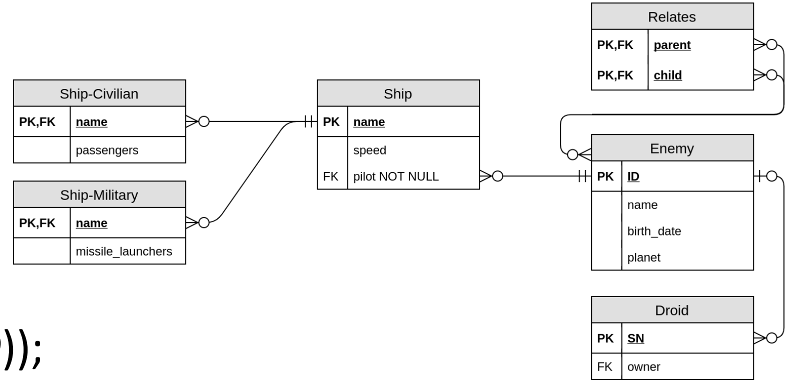
- Star Wars SQL syntax

```
CREATE TABLE enemy (
    ID          char(5),
    name        varchar(255),
    birth_date  date,
    planet      varchar(255),
    PRIMARY KEY (ID));

CREATE TABLE relates (
    parent      char(5),
    child       char(5),
    PRIMARY KEY (parent, child),
    FOREIGN KEY (parent) REFERENCES enemy (ID),
    FOREIGN KEY (child) REFERENCES enemy (ID));
```



- Star Wars SQL syntax



CREATE TABLE *droid* (
    SN **char**(32),
    owner **char**(5),
    PRIMARY KEY (*SN*),
    FOREIGN KEY (*owner*) **REFERENCES** *enemy* (*ID*));

CREATE TABLE *ship* (
    name **varchar**(255),
    speed **float**,
    pilot **char**(5) **NOT NULL**,
    PRIMARY KEY (*name*),
    FOREIGN KEY (*pilot*)
    REFERENCES *enemy* (*ID*));

CREATE TABLE *ship-civilian* (
    name **varchar**(255),
    passengers **int**,
    PRIMARY KEY (*name*),
    FOREIGN KEY (*name*) **REFERENCES** *ship* (*name*));

** Similar to ship-military*

- INSERT (MySQL Reference Manual)

  INSERT INTO relation **VALUES** (V_1, …, V_N);
  Inserts a new row with **all** the values matching the **order** of the columns. DO NOT USE.
  INSERT INTO instructor **VALUES** (123, 'ALBERTO CANO', NULL, 1234.56, 'E4251');

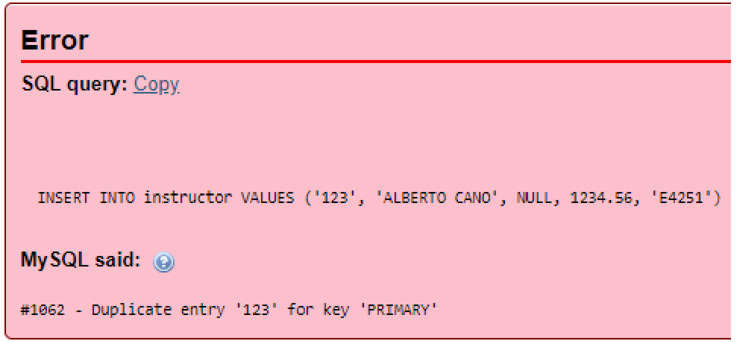
  INSERT INTO relation (C_i, …, C_j) **VALUES** (V_i, …, V_j);
  Inserts a new row with the values matching any order of the columns specified, any other column not listed is assumed to be NULL.

  INSERT INTO instructor (ID, name, salary, office)
  VALUES (123, 'ALBERTO CANO', 1234.56, 'E4251');

  INSERT INTO instructor (name, salary, office)
  VALUES ('ALBERTO CANO', 9876.5, 'W0465'); -- ID AUTO GENERATED

- Understanding MySQL errors
  - If we execute both instructions, the first will be valid inserting the new instructor with ID 123. However, the second will fail because we attempt to insert another row with the same value for the PRIMARY KEY (ID).

INSERT INTO instructor **VALUES** (123, 'ALBERTO CANO', NULL, 1234.56, 'E4251');
INSERT INTO instructor (ID, name, salary, office)
VALUES (123, 'ALBERTO CANO', 1234.56, 'E4251');


Error

SQL query: Copy

```
INSERT INTO instructor VALUES ('123', 'ALBERTO CANO', NULL, 1234.56, 'E4251')
```

MySQL said:

```
#1062 - Duplicate entry '123' for key 'PRIMARY'
```

- UPDATE (MySQL Reference Manual)

UPDATE relation **SET** $C_i = V_i$, $C_j = V_j$, … [WHERE predicate];
Updates the content of a table, modifying the columns specified with the provided values for the rows satisfying the predicate.

UPDATE instructor **SET** salary = 4321.00 **WHERE** ID = 123;
CORRECT. Will update the salary based on the **PRIMARY KEY.**

UPDATE instructor **SET** salary = 4321.00 **WHERE** name = 'ALBERTO CANO';
INCORRECT. Will update the salary for ALL instructors with the given name.

UPDATE instructor **SET** salary = 4321.00;
INCORRECT. Will update everyone's salary!

UPDATE instructor **SET** salary = salary * 1.05;
Will update everyone's salary +5% respectively.

- UPDATE (MySQL Reference Manual)

```
UPDATE instructor
SET salary = CASE
    WHEN salary <= 100000 THEN
        salary * 1.05
   ELSE
        salary * 1.03
END;
```

Updates every salary based on SWITCH CASE

```
UPDATE instructor SET dept_name = 'CS' WHERE ID = 123;
```

```
#1452 - Cannot add or update a child row: a foreign key constraint fails (`acano`.`instructor`, CONSTRAINT `instructor_ibfk_1` FOREIGN KEY (`dept_name`) REFERENCES `departments` (`department_name`))
```

Foreign key violation! there's **no** 'CS' department in the departments table.
Update command canceled.

- DELETE (MySQL Reference Manual)

  DELETE FROM relation [WHERE predicate];
  Removes from a table the rows satisfying the condition of the predicate.

  DELETE FROM instructor;
  Removes every row from the instructor table. Are you sure about that?

  DELETE FROM instructor **WHERE** ID = 123;
  CORRECT. Removes the row where the **PRIMARY KEY** (ID) is 123.

  DELETE FROM instructor **WHERE** name = 'ALBERTO CANO';
  INCORRECT. Removes all rows for instructors with a given name.

  DELETE FROM instructor; [PANIC]
  If you know some Spanish, there's even a musical video to honor those who forget the WHERE clause in a DELETE statement on production systems.

- DELETE (MySQL Reference Manual)

What happens if you try to delete a row for which there's an existing foreign key?

DELETE FROM employees **WHERE** employee_id = 100;

```
#1451 - Cannot delete or update a parent row: a foreign key constraint fails (`acano`.`departments`, CONSTRAINT `dept_mgr_fk` FOREIGN KEY (`manager_id`) REFERENCES `employees` (`employee_id`))
```

You cannot delete a "parent" record if there's a child dependency -> Foreign key constraints -> Referential integrity

- TRUNCATE (MySQL Reference Manual)

  TRUNCATE TABLE relation;
  Removes all rows in a table and resets metadata (counters, indexes, etc)

- DROP TABLE (MySQL Reference Manual)

  DROP TABLE relation;
  Removes the table from the database.

  DROP TABLE instructor;

  DELETE = Removes rows satisfying conditions
  TRUNCATE = Removes all rows in a table and resets metadata (e.g. auto increment)
  DROP TABLE = Deletes the whole table from the database

- Basic date and time data types

DATA TYPES: DATE vs DATETIME vs TIMESTAMP

```
SELECT SYSDATE() FROM DUAL;
SELECT UTC_TIMESTAMP() FROM DUAL;
SELECT CURDATE() FROM DUAL;

CREATE TABLE time_test (
  ex_date           DATE,
  ex_datetime       DATETIME,
  ex_timestamp      TIMESTAMP
);

INSERT INTO time_test (ex_date, ex_datetime) VALUES (CURDATE(), SYSDATE());
UPDATE time_test SET ex_date = '2020-01-01’;
```

I strongly recommend to **ALWAYS** use UTC_TIMESTAMP() and forget timezones

- Exercises

1. Create a table to store the set of countries visited by the employees, the date and the cost of the trip. Use the correct data types and foreign keys whenever necessary.
2. Insert 5 valid rows into the previous table with diverse combinations (including an employee visiting a country on multiple dates).
3. Update the cost of a trip for an employee who had several trips to the same country (based on the data you inserted).
4. Delete a given trip.
5. Delete all trips with a cost > $10k.
6. Try to delete a country for which an employee made a trip to. If you can't, why?
7. Delete all rows. Insert a new row. What's the new trip ID?
8. Truncate the table and insert a new row. What's the new trip ID?
9. Watch this piece of art

CMSC 508 Databases

Introduction to SQL (I)

VCU College of Engineering

Dr. Alberto Cano
Associate Professor
Department of Computer Science

Chapter 3 from Database System Concepts, 7th Ed. by Silberschatz, Korth, Sudarshan, 2019
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003