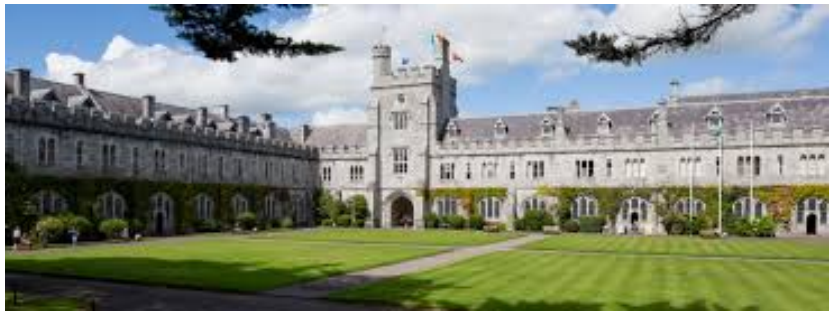
UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh



# Information Storage and Management I

Dr. Alejandro Arbelaez

- Labs commence in week 2 (next week - Thursdays) - **G.24** & G.21
- Canvas is now working ☺

# The MySQL Server

- A MySQL server can be given SQL commands, executes them, and results the results to the connected application:
- Starting the server
- Stopping the server

# Command Line Interface

- Starting the MySQL client
- Commands for looking around in the database
  - SHOW DATABASES
  - USE database
  - SHOW TABLES
  - DESCRIBE table

# Tables

- A typical database table definition has:
- A name
- A list of columns and their data types
- A list of constraints
  - Primary key to ensure uniqueness (if needed)
  - Foreign keys to facilitate relationships with other tables
  - Indices to facilitate fast look ups

# Primary Key

- A **primary key** is a field in a table which uniquely identifies each row/record in a database table.
- **Primary keys** must contain unique values.
- A **primary key** column cannot have NULL values.
- A table can have only one **primary key**, which may consist of single or multiple fields.

| employee_id | course_id | taken_date |
|---|---|---|
| 100 | 3 | 1987-06-17 |
| 101 | 3 | 1989-09-21 |
| 102 | 3 | 1993-01-13 |
| 103 | 3 | 1990-01-03 |
| 104 | 3 | 1991-05-21 |
| 105 | 3 | 1997-06-25 |
| 106 | 3 | 1998-02-05 |

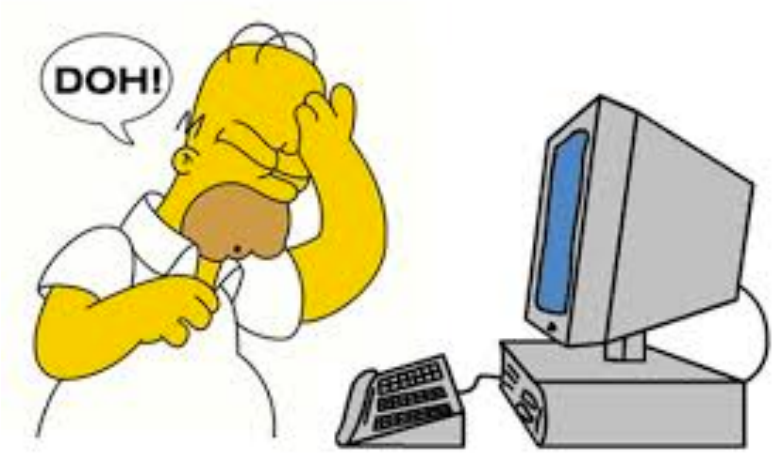
# Primary Key

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2)
);
```

# Primary Key

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2)
);
```

We are missing primary key(s)




```
ALTER TABLE Customers
ADD PRIMARY KEY (ID);
```

# Primary Key

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2),
    PRIMARY KEY (ID)
);
```

# Primary Key (with Multiple Columns)

```
CREATE TABLE Customers (
    ID INT NOT NULL,
    NAME VARCHAR (20) NOT NULL,
    AGE INT NOT NULL,
    ADDRESS CHAR (25) ,
    SALARY DECIMAL (18, 2),
    PRIMARY KEY (ID, NAME)
);
```

# Delete Primary Key

**ALTER TABLE** Customers
**DROP PRIMARY KEY ;**

You can clear the primary key constraints from the table with the syntax given below.

# Database structure for Quizes

| id | description | creation_date |
|---|---|---|
| 1 | Favorite Things Quiz | 10/07/2014 10:22 |

quiz table

| id | text | quiz_id |
|---|---|---|
| 1 | What is your favorite color? | 1 |
| 2 | What is your favorite book? | 1 |
| … | | |

question table

| id | text | point_value | question_id |
|---|---|---|---|
| 1 | Red | 1 | 1 |
| 2 | Green | 10 | 1 |
| … | | | |

answer table

# Creating The Database Schema

```
CREATE TABLE quiz (
    id INT NOT NULL AUTO_INCREMENT,
    description VARCHAR(255),
    create_time DATETIME NOT NULL,
    PRIMARY KEY(id)
)
```

```
CREATE TABLE question (
    id INT NOT NULL AUTO_INCREMENT,
    text VARCHAR(255),
    quiz_id INT NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (quiz_id) REFERENCES
quiz(id) ON DELETE CASCADE
)
```

```
CREATE TABLE answer (
    id INT NOT NULL AUTO_INCREMENT,
    text VARCHAR(255) NOT NULL,
    point_value INT NOT NULL,
    question_id INT NOT NULL,
    PRIMARY KEY(id),
    FOREIGN KEY (question_id) REFERENCES
question(id) ON DELETE CASCADE
)
```

# Filling In the Details

- Data types:
  - Numbers: INT, LONGINT, NUMERIC, FLOAT, DOUBLE
  - Strings: VARCHAR(<<NUM CHARS>>), TEXT, BLOB
  - Other: DATETIME
- NOT NULL vs NULL: whether to allow empty values or not
- PRIMARY KEY and FOREIGN KEY
- CASCADE: Keeping the data clean and robust

# Null Values

- It is possible for tuples to have null values, denoted by null, for some of their attributes
- Null signifies an unknown value or that a value does not exist
- The result of any arithmetic expression involving null is null
  - Example: 5 + null returns null
- The predicate is null can be used to check for null values
  - **SELECT** name **FROM** instructor **WHERE** salary **is null**
- The predicate null is not null success if the value on which it is applied is not null

# Null Values (Cont.)

- SQL treats as unknown the result of any comparison involving a null value (other than predicates is null and  is not null).
  - Example: 5 < null  or  null <> null  or  null = null
- The predicate in a where clause can involve Boolean operations (and, or, not); thus the definitions of the Boolean operations need to be  extended to deal with the value unknown.
  - and : (true and unknown)  = unknown,
    (false and unknown) = false,
    (unknown and unknown) = unknown
  - or:  (unknown or true)  = true,
    (unknown or false)  = unknown
    (unknown or unknown) = unknown
- Result of where clause predicate is treated as false if it evaluates to unknown

# Aggregate Functions

These functions operate on the multiset of values of a column of a relation, and return a value

avg: average value
min: minimum value
max: maximum value
sum: sum of values
count: number of values

# Aggregate Functions Examples

- Find the average salary of instructors in the Computer Science department
  - **SELECT AVG**(salary)  **FROM** instructor **WHERE** dept_name= 'Comp. Sci.'
- Find the total number of instructors who teach a course in the Spring 2010 semester
  - **SELECT COUNT** (distinct ID) **FROM** teaches **WHERE** semester = 'Spring' and year = 2018
- Find the number of tuples in the course relation
  - **SELECT COUNT** (*) **FROM** course;

# Aggregate Functions – Group By

- Find the average salary of instructors in each department

**SELECT** [?]

**FROM** instructor **GROUP BY** dept_name;

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 76766 | Crick | Biology | 72000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 12121 | Wu | Finance | 90000 |
| 76543 | Singh | Finance | 80000 |
| 32343 | El Said | History | 60000 |
| 58583 | Califieri | History | 62000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 22222 | Einstein | Physics | 95000 |

# Aggregate Functions – Group By

- Find the average salary of instructors in each department

**SELECT** dept_name, AVG (salary) AS avg_salary **FROM** instructor **GROUP BY** dept_name;

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 76766 | Crick | Biology | 72000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 12121 | Wu | Finance | 90000 |
| 76543 | Singh | Finance | 80000 |
| 32343 | El Said | History | 60000 |
| 58583 | Califieri | History | 62000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 22222 | Einstein | Physics | 95000 |

# Aggregate Functions – Group By

- Find the average salary of instructors in each department

**SELECT** dept_name, AVG (salary) AS avg_salary **FROM** instructor **GROUP BY** dept_name;

| *ID* | *name* | *dept_name* | *salary* |
|---|---|---|---|
| 76766 | Crick | Biology | 72000 |
| 45565 | Katz | Comp. Sci. | 75000 |
| 10101 | Srinivasan | Comp. Sci. | 65000 |
| 83821 | Brandt | Comp. Sci. | 92000 |
| 98345 | Kim | Elec. Eng. | 80000 |
| 12121 | Wu | Finance | 90000 |
| 76543 | Singh | Finance | 80000 |
| 32343 | El Said | History | 60000 |
| 58583 | Califieri | History | 62000 |
| 15151 | Mozart | Music | 40000 |
| 33456 | Gold | Physics | 87000 |
| 22222 | Einstein | Physics | 95000 |

| *dept_name* | *avg_salary* |
|---|---|
| Biology | 72000 |
| Comp. Sci. | 77333 |
| Elec. Eng. | 80000 |
| Finance | 85000 |
| History | 61000 |
| Music | 40000 |
| Physics | 91000 |

# Aggregation (Cont.)

Attributes in select clause outside of aggregate functions must appear in group by list

**SELECT** dept_name, ID, **AVG** (salary)
**FROM** instructor
**GROUP BY** dept_name;

correct or incorrect

# Aggregation (Cont.)

Attributes in select clause outside of aggregate functions must appear in group by list

```
/* erroneous query */
SELECT dept_name, ID, AVG (salary)
FROM instructor
GROUP BY dept_name;
```

# Aggregate Functions – Having Clause

Find the names and average salaries of all departments whose average salary is greater than 42000

```
SELECT dept_name, AVG (salary) AS avg_salary
FROM instructor
GROUP BY dept_name
HAVING AVG (salary) > 42000;
```

**Note:** predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

# Null Values and Aggregates

- Total all salaries

  SELECT SUM (salary ) FROM instructor

  - Above statement ignores null amounts
  - Result is null if there is no non-null amount
- All aggregate operations except count(*) ignore tuples with null values on the aggregated attributes
- What if collection has only null values?
  - count returns 0
  - all other aggregates return null

# Nested Subqueries

- SQL provides a mechanism for the nesting of subqueries. A subquery is a select-from-where expression that is nested within another query.
- The nesting can be done in the following SQL query

**SELECT** A1, A2, ..., An **FROM** r1, r2, ..., rm **WHERE** P

as follows:

From clause: ri can be replaced by any valid subquery

Where clause: P can be replaced with an expression of the form:

B <operation> (subquery)

Where B is an attribute and <operation> to be defined later.

Select clause:

Ai can be replaced be a subquery that generates a single value.

# Set Membership

- Find courses offered in Fall 2017 and in Spring 2018

```
SELECT DISTINCT course_id
FROM section
WHERE semester = 'Fall' AND year= 2017 AND
        course_id IN (SELECTcourse_id
                      FROM section
                      WHERE semester = 'Spring' AND year= 2018);
```

- Find courses offered in Fall 2017 but not in Spring 2018

```
SELECT DISTINCT course_id
FROM section
WHERE semester = 'Fall' AND year= 2017 AND
        course_id  NOT IN (SELECT course_id
                           FROM section
                           WHERE semester = 'Spring' AND year= 2018);
```

# Set Membership (Cont.)

- Name all instructors whose name is neither “Mozart” nor Einstein”

  **SELECT distinct** name **FROM** instructor

  **WHERE** name **NOT IN** ('Mozart', 'Einstein')

- Find the total number of (distinct) students who have taken course sections taught by the instructor with ID 10101

**SELECT COUNT** (**distinct** *ID*) **FROM** *takes*
**WHERE** (*course_id*, *sec_id*, *semester*, *year*) **IN**
(**SELECT** *course_id*, *sec_id*, *semester*, *year*
**FROM** *teaches*
**WHERE** *teaches*.*ID*= 10101);

# Set Comparison

- Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

**SELECT DISTINCT** *T.name* **from** *instructor* **as** *T*, *instructor* **AS** *S*
**WHERE** *T.salary* > *S.salary* **AND** *S.dept name* = 'Biology';

# Scalar Subquery

- Scalar subquery is one which is used where a single value is expected
- List all departments along with the number of instructors in each department

```
SELECT dept_name,
 ( SELECT count(*)
    FROM instructor
    WHERE department.dept_name = instructor.dept_name)
  AS num_instructors
FROM department;
```

- Runtime error if subquery returns more than one result tuple

# Modification of the Database

- Deletion of tuples from a given relation/table.
- Insertion of new tuples into a given relation/table
- Updating of values in some tuples in a given relation/table

# Deletion

- Delete all instructors

```
DELETE FROM instructor
```

- Delete all instructors from the Finance department

```
DELETE FROM instructor
WHERE  dept_name= 'Finance';
```

# Deletion (Cont.)

- Delete all instructors whose salary is less than the average salary of instructors

```
DELETE FROM instructor
WHERE salary < (SELECT AVG(salary)
                FROM instructor);
```

Problem: as we delete tuples from deposit, the average salary changes

Solution used in SQL:

1. First, compute **avg** (salary) and find all tuples to delete
2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

# Insertion

- Add a new tuple to course

```
INSERT INTO course
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- or equivalently

```
INSERT INTO course (course_id, title, dept_name, credits)
VALUES ('CS-437', 'Database Systems', 'Comp. Sci.', 4);
```

- Add a new tuple to student  with tot_creds set to null

```
INSERT INTO student
VALUES ('3003', 'Green', 'Finance', null);
```

# Insertion (Cont.)

- Make each student in the Music department who has earned more than 144 credit hours an instructor in the Music department with a salary of $18,000.

```
INSERT INTO instructor
SELECT ID, name, dept_name, 18000
FROM  student
WHERE dept_name = 'Music' and total_cred > 144;
```

- The select from where statement is evaluated fully before any of its results are inserted into the relation.
- Otherwise queries like

```
INSERT INTO table1 SELECT * FROM table1
```

would cause problem

# Updates

- Give a 5% salary raise to all instructors

```
UPDATE instructor
SET salary = salary * 1.05
```

- Give a 5% salary raise to those instructors who Eran less than 70000

```
UPDATE instructor
SET salary = salary * 1.05
WHERE salary < 70000;
```

- Give a 5% salary raise to instructors whose salary is less than average

```
UPDATE instructor
SET salary = salary * 1.05
WHERE salary <  (select avg (salary)
                 from instructor);
```

# Updates (Cont.)

- Increase salaries of instructors whose salary is over $100,000 by 3%, and all others by a 5%
  - Write two update statements:

```
UPDATE  instructor
SET salary = salary * 1.03
WHERE salary > 100000;
UPDATE  instructor
SET salary = salary * 1.05
WHERE salary <= 100000;
```

  - The order is important
  - Can be done better using the case statement (next slide)

# Case Statement for Conditional Updates

- Same query as before but with case statement

```
UPDATE instructor
 SET  salary = case
            WHEN salary <= 100000 then salary * 1.05
             ELSE salary * 1.03
             END
```

# Destroying and Altering Relations

**DROP TABLE** Students

- Destroys Students

**ALTER TABLE** Students **ADD COLUMN** firstYear

- Students will be altered by adding anew field
- Every tuple in the current instance is extended with a null value in the new field