ECE30030/ITP30010 Database Systems

# More SQL

*Reading: Chapter 3*

***Charmgil Hong***

charmgil@handong.edu

Spring, 2023

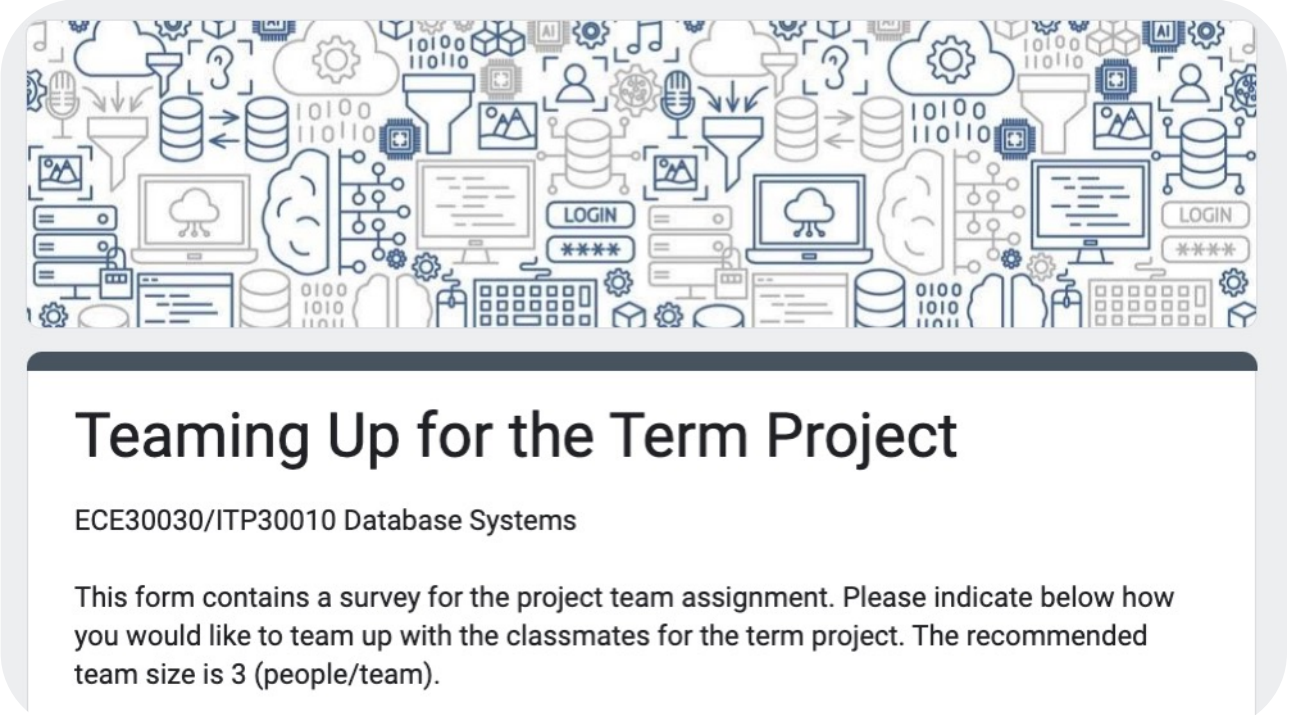
Handong Global University

# Announcement

- Homework assignment #2 is due by this Friday (March 7)
  - If you haven't, please start today

# Announcement

- Forming teams for the term project
  - Response due: Monday, April 10
  - URL: https://forms.gle/kQWG9ML6fqytYm7p7
  - Problem & data release: Week #8



# Agenda

- Structured query language (SQL)
- **SQL data manipulation language (DML)**
  - SELECT, FROM, WHERE
  - NULL values
  - Set operations
  - String operations, ordering
  - **Aggregate functions, aggregation**
- SQL data definition language (DDL)

# 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.';

| `AVG(salary)` |
| --- |
| 77333.333333 |

- Find the total number of instructors who teach a course in the Spring 2018 semester
  - **SELECT COUNT**(**DISTINCT** *ID*)
    **FROM** *teaches*
    **WHERE** *semester* = 'Spring' **AND** *year* = 2018;

| `COUNT(DISTINCT ID)` |
| --- |
| 6 |

- Find the number of tuples in the *teaches* relation
  - **SELECT COUNT** (*)
    **FROM** *teaches*;

| `COUNT(*)` |
| --- |
| 15 |

# 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 | 72000.000000 |
| Comp. Sci. | 77333.333333 |
| Elec. Eng. | 80000.000000 |
| Finance | 85000.000000 |
| History | 61000.000000 |
| Music | 40000.000000 |
| Physics | 91000.000000 |

# Aggregation

- 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*;

has no meaning, doesn't represent anything. Since each dept_name may have multiple IDs

| dept_name | ID | `AVG(salary)` |
|---|---|---|
| Biology | 76766 | 72000.000000 |
| Comp. Sci. | 10101 | 77333.333333 |
| Elec. Eng. | 98345 | 80000.000000 |
| Finance | 12121 | 85000.000000 |
| History | 32343 | 61000.000000 |
| Music | 15151 | 40000.000000 |
| Physics | 22222 | 91000.000000 |

# Aggregate Functions – Having Clause

- Find the names and average salaries of all departments whose average salary is greater than 65000
  - **SELECT** *dept_name*, **AVG**(*salary*) **AS** *avg_salary*
    **FROM** *instructor*
    **GROUP BY** *dept_name*
    **HAVING AVG**(*salary*) > 65000;

| dept_name | avg_salary |
|---|---|
| Biology | 72000.000000 |
| Comp. Sci. | 77333.333333 |
| Elec. Eng. | 80000.000000 |
| Finance | 85000.000000 |
| Physics | 91000.000000 |

Where vs Having.

: where : filtering out some records.
where clause should be sentenced befor GROUP BY.

HAVING : To filter out after GROUP BY

# Aggregate Functions – Having Clause

- Note: predicates in the **HAVING** clause are applied after the formation of groups whereas predicates in the **WHERE** clause are applied before forming groups

**SELECT** *dept_name*, **AVG**(*salary*) **AS** *avg_salary*
**FROM** *instructor*
**GROUP BY** *dept_name*
**HAVING AVG**(*salary*) > 65000;

| dept_name | avg_salary |
|---|---|
| Biology | 72000.000000 |
| Comp. Sci. | 77333.333333 |
| Elec. Eng. | 80000.000000 |
| Finance | 85000.000000 |
| Physics | 91000.000000 |

**SELECT** *dept_name*, **AVG**(*salary*) **AS** *avg_salary*
**FROM** *instructor*
**WHERE** *salary* > 65000
**GROUP BY** *dept_name*;

| dept_name | avg_salary |
|---|---|
| Biology | 72000.000000 |
| Comp. Sci. | 83500.000000 |
| Elec. Eng. | 80000.000000 |
| Finance | 85000.000000 |
| Physics | 91000.000000 |

# SQL Order of Execution

| Order | Clause | Function |
|---|---|---|
| 1 | FROM | Choose and join tables to get base data |
| 2 | WHERE | Filters the base data |
| 3 | GROUP BY | Aggregates the base data |
| 4 | HAVING | Filters the aggregated data |
| 5 | SELECT | Returns the final data |
| 6 | ORDER BY | Sorts the final data |
| 7 | LIMIT | Limits the returned data to a row count |

# Agenda

- Nested subqueries
- Set membership (SOME, ALL, EXISTS)

# Running Examples

- Relations (tables): *instructor*, *teaches*

*Instructor* relation

| ID | name | dept_name | salary |
|---|---|---|---|
| 10101 | Srinivasan | Comp. Sci. | 65000.00 |
| 12121 | Wu | Finance | 90000.00 |
| 15151 | Mozart | Music | 40000.00 |
| 22222 | Einstein | Physics | 95000.00 |
| 32343 | El Said | History | 60000.00 |
| 33456 | Gold | Physics | 87000.00 |
| 45565 | Katz | Comp. Sci. | 75000.00 |
| 58583 | Califieri | History | 62000.00 |
| 76543 | Singh | Finance | 80000.00 |
| 76766 | Crick | Biology | 72000.00 |
| 83821 | Brandt | Comp. Sci. | 92000.00 |
| 98345 | Kim | Elec. Eng. | 80000.00 |

*teaches* relation

| ID | course_id | sec_id | semester | year |
|---|---|---|---|---|
| 76766 | BIO-101 | 1 | Summer | 2017 |
| 76766 | BIO-301 | 1 | Summer | 2018 |
| 10101 | CS-101 | 1 | Fall | 2017 |
| 45565 | CS-101 | 1 | Spring | 2018 |
| 83821 | CS-190 | 1 | Spring | 2017 |
| 83821 | CS-190 | 2 | Spring | 2017 |
| 10101 | CS-315 | 1 | Spring | 2018 |
| 45565 | CS-319 | 1 | Spring | 2018 |
| 83821 | CS-319 | 2 | Spring | 2018 |
| 10101 | CS-347 | 1 | Fall | 2017 |
| 98345 | EE-181 | 1 | Spring | 2017 |
| 12121 | FIN-201 | 1 | Spring | 2018 |
| 32343 | HIS-351 | 1 | Spring | 2018 |
| 15151 | MU-199 | 1 | Spring | 2018 |
| 22222 | PHY-101 | 1 | Fall | 2017 |

# Running Examples

- Relations (tables): *course*, *takes*

*course* relation

| course_id | title | dept_name | credits |
|---|---|---|---|
| BIO-101 | Intro. to Biology | Biology | 4 |
| BIO-301 | Genetics | Biology | 4 |
| BIO-399 | Computational Biology | Biology | 3 |
| CS-101 | Intro. to Computer Science | Comp. Sci. | 4 |
| CS-190 | Game Design | Comp. Sci. | 4 |
| CS-315 | Robotics | Comp. Sci. | 3 |
| CS-319 | Image Processing | Comp. Sci. | 3 |
| CS-347 | Database System Concepts | Comp. Sci. | 3 |
| EE-181 | Intro. to Digital Systems | Elec. Eng. | 3 |
| FIN-201 | Investment Banking | Finance | 3 |
| HIS-351 | World History | History | 3 |
| MU-199 | Music Video Production | Music | 3 |
| PHY-101 | Physical Principles | Physics | 4 |

*takes* relation

| ID | course_id | sec_id | semester | year | grade |
|---|---|---|---|---|---|
| 00128 | CS-101 | 1 | Fall | 2017 | A |
| 00128 | CS-347 | 1 | Fall | 2017 | A- |
| 12345 | CS-101 | 1 | Fall | 2017 | C |
| 12345 | CS-190 | 2 | Spring | 2017 | A |
| 12345 | CS-315 | 1 | Spring | 2018 | A |
| 12345 | CS-347 | 1 | Fall | 2017 | A |
| 19991 | HIS-351 | 1 | Spring | 2018 | B |
| 23121 | FIN-201 | 1 | Spring | 2018 | C+ |
| 44553 | PHY-101 | 1 | Fall | 2017 | B- |
| 45678 | CS-101 | 1 | Fall | 2017 | F |
| 45678 | CS-101 | 1 | Spring | 2018 | B+ |
| 45678 | CS-319 | 1 | Spring | 2018 | B |
| 54321 | CS-101 | 1 | Fall | 2017 | A- |
| 54321 | CS-190 | 2 | Spring | 2017 | B+ |
| 55739 | MU-199 | 1 | Spring | 2018 | A- |
| 76543 | CS-101 | 1 | Fall | 2017 | A |
| 76543 | CS-319 | 2 | Spring | 2018 | A |
| 76653 | EE-181 | 1 | Spring | 2017 | C |
| 98765 | CS-101 | 1 | Fall | 2017 | C- |
| 98765 | CS-315 | 1 | Spring | 2018 | B |
| 98988 | BIO-101 | 1 | Summer | 2017 | A |
| 98988 | BIO-301 | 1 | Summer | 2018 | *null* |

# Running Examples

- Relations (tables): *student*

*student* relation

| ID | name | dept_name | tot_cred |
|---|---|---|---|
| 00128 | Zhang | Comp. Sci. | 102 |
| 12345 | Shankar | Comp. Sci. | 32 |
| 19991 | Brandt | History | 80 |
| 23121 | Chavez | Finance | 110 |
| 44553 | Peltier | Physics | 56 |
| 45678 | Levy | Physics | 46 |
| 54321 | Williams | Comp. Sci. | 54 |
| 55739 | Sanchez | Music | 38 |
| 70557 | Snow | Physics | 0 |
| 76543 | Brown | Comp. Sci. | 58 |
| 76653 | Aoi | Elec. Eng. | 60 |
| 98765 | Bourikas | Elec. Eng. | 98 |
| 98988 | Tanaka | Biology | 120 |

# 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** $A_1, A_2, ..., A_n$
  **FROM** $r_1, r_2, ..., r_m$
  **WHERE** $P$

  as follows:
  - **FROM clause:** $r_i$ can be replaced by any valid subquery
  - **WHERE clause:** $P$ can be replaced with an expression of the form:
    $B$ <operation> (subquery)
    $B$ is an attribute and <operation> to be defined later
  - **SELECT clause:**
    $A_i$ can be replaced be a subquery that generates a single value (scalar subquery)

# Subqueries in the FROM Clause

- Find the average instructors' salaries of those departments where the average salary is greater than $42,000
  - **SELECT** *D.dept_name*, *D.avg_salary* → attribut name in subquery.
    **FROM** ( **SELECT** *dept_name*, **AVG**(*salary*) **AS** *avg_salary*
    **FROM** *instructor*
    **GROUP BY** *dept_name*) **AS** *D*
    **WHERE** *D.avg_salary* > 42000;

| dept_name | avg_salary |
|---|---|
| Biology | 72000.000000 |
| Comp. Sci. | 77333.333333 |
| Elec. Eng. | 80000.000000 |
| Finance | 85000.000000 |
| History | 61000.000000 |
| Physics | 91000.000000 |

# WITH Clause

- The **WITH** clause provides a way of defining a temporary relation
  - The relation is available only to the query in which the **WITH** clause occurs

- Find all departments with the maximum budget
  - **WITH** *max_budget* (*value*) **AS**
    (**SELECT MAX**(*budget*)
    **FROM** *department*)
    **SELECT** *department.dept_name*
    **FROM** *department, max_budget*
    **WHERE** *department.budget* = *max_budget.value*;

← temporary relation until this

| dept_name |
| --- |
| Finance |

# Scalar Subquery

- Scalar subquery is used where a single value is expected
  - Runtime error occurs if a subquery returns more than one result tuple
- 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*;

| dept_name | num_instructors |
|---|---|
| Biology | 1 |
| Comp. Sci. | 3 |
| Elec. Eng. | 1 |
| Finance | 2 |
| History | 2 |
| Music | 1 |
| Physics | 2 |

# Agenda

- Nested subqueries
- **Set membership (SOME, ALL, EXISTS)**

# Set Membership

- Find courses offered in Fall 2017 and in Spring 2018
  - **SELECT DISTINCT** *course_id*
    **FROM** *teaches*
    **WHERE** *semester* = 'Fall' **AND** *year*= 2017 **AND**
    *course_id* **IN** (**SELECT** *course_id*
    **FROM** *teaches*
    **WHERE** *semester* = 'Spring' **AND** *year*= 2018);

| course_id |
| --- |
| CS-101 |

# Set Membership

- Find courses offered in Fall 2017 but not in Spring 2018
  - **SELECT DISTINCT** *course_id*
    **FROM** *teaches*
    **WHERE** *semester* = 'Fall' **AND** *year*= 2017 **AND**
    *course_id* ***NOT IN*** (**SELECT** *course_id*
    **FROM** *teaches*
    **WHERE** *semester* = 'Spring' **AND** *year*= 2018);

| course_id |
| --- |
| CS-347 |
| PHY-101 |

# Set Membership

- Name all instructors whose name is neither “Mozart” nor Einstein”
  - **SELECT DISTINCT** *name*
    **FROM** *instructor*
    **WHERE** *name* **NOT IN** ('Mozart', 'Einstein’);

| name |
| --- |
| Srinivasan |
| Wu |
| El Said |
| Gold |
| Katz |
| Califieri |
| Singh |
| Crick |
| Brandt |
| Kim |

| name |
| --- |
| Srinivasan |
| Wu |
| Mozart |
| Einstein |
| El Said |
| Gold |
| Katz |
| Califieri |
| Singh |
| Crick |
| Brandt |
| Kim |

# Set Membership

- Find the total number of unique 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);

| `COUNT(DISTINCT ID)` |
| --- |
| 6 |

- Note: *Above query can be written in a much simpler manner*
  *The formulation above is simply to illustrate SQL features*

# Set Comparison – SOME

- 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';

- Same query using > **SOME** clause
  - **SELECT** *name*
    **FROM** *instructor*
    **WHERE** *salary* > **SOME** (**SELECT** *salary*
    **FROM** *instructor*
    **WHERE** *dept_name* = 'Biology');

| name |
|---|
| Wu |
| Einstein |
| Gold |
| Katz |
| Singh |
| Brandt |
| Kim |

# Interpretation of SOME

- F <comp> **SOME** $r \Leftrightarrow \exists t \in r$ such that (F <comp> $t$)
  Where <comp> can be: $<, \leq, >, =, \neq$

(5 < **SOME** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **SOME** | 0 / 5 |) = false

(5 = **SOME** | 0 / 5 |) = true

(5 ≠ **SOME** | 0 / 5 |) = true (since 0 ≠ 5)

(= **SOME**) ≡ **IN**
However, (≠ **SOME**) ≢ **NOT IN**

# Set Comparison – ALL

- Find the names of ALL instructors whose salary is greater than the salary of ALL instructors in the Biology department
  - **SELECT** *name*
    **FROM** *instructor*
    **WHERE** *salary* > **ALL** (**SELECT** *salary*
    **FROM** *instructor*
    **WHERE** *dept name* = 'Biology');

| name |
| --- |
| Wu |
| Einstein |
| Gold |
| Katz |
| Singh |
| Brandt |
| Kim |

# Interpretation of ALL

- F <comp> **ALL** $r \Leftrightarrow \forall t \in r$ (F <comp> $t$)

(5 < **ALL** | 0 / 5 / 6 |) = false

(5 < **ALL** | 6 / 10 |) = true

(5 = **ALL** | 4 / 5 |) = false

(5 ≠ **ALL** | 4 / 6 |) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **ALL**) ≡ **NOT IN**

However, (= **ALL**) ≢ **IN**

# Test for Empty Relations

- The **EXISTS** construct returns the value *true* if the argument subquery is nonempty
  - **EXISTS** $r \Leftrightarrow r \neq \emptyset$
  - **NOT EXISTS** $r \Leftrightarrow r = \emptyset$

# Use of EXISTS

- Yet another way of specifying the query “Find all courses taught in both the Fall 2017 semester and in the Spring 2018 semester”
  - **SELECT** *course_id*
    **FROM** *teaches* **AS** *S*
    **WHERE** *semester* = 'Fall' **AND** *year* = 2017 **AND**
    **EXISTS** (**SELECT** *
    **FROM** *teaches* **AS** *T*
    **WHERE** *semester* = 'Spring' **AND** *year*= 2018
    **AND** *S.course_id* = *T.course_id*);

| course_id |
| --- |
| CS-101 |

# Use of NOT EXISTS

- Find all students who have taken all courses offered in the Music department
  - **SELECT DISTINCT** *S.ID*, *S.name*
    **FROM** *student* **AS** *S*
    **WHERE NOT EXISTS** ( **SELECT** *course_id*
    **FROM** *course*
    **WHERE** *dept_name* = 'Music'
    **AND** *course_id* **NOT IN**
    (**SELECT** *T.course_id*
    **FROM** *takes* **AS** *T*
    **WHERE** *S.ID* = *T.ID*));

| ID | name |
|---|---|
| 55739 | Sanchez |

# Use of NOT EXISTS

- Note: Renaming (**AS**) is optional in certain contexts
  - **SELECT DISTINCT** *ID*, *name*
    **FROM** *student*
    **WHERE NOT EXISTS** ( **SELECT** *course_id*
    **FROM** *course*
    **WHERE** *dept_name* = ‘Music'
    **AND** *course_id* **NOT IN**
    (**SELECT** *course_id*
    **FROM** *takes*
    **WHERE** *student*.*ID* = *takes*.*ID*));

  - Exception: the following query results in an empty relation
    - **SELECT DISTINCT** *name*
      **FROM** *instructor*
      **WHERE** *salary* > *salary* **AND** *dept_name* = 'Biology';

# Use of NOT EXISTS

- Some systems support the **EXCEPT** clause (MySQL does not)
- Find all students who have taken all courses offered in the Music department
  - **SELECT DISTINCT** *S.ID*, *S.name*
    **FROM** *student* **AS** *S*
    **WHERE NOT EXISTS** ( (**SELECT** *course_id*
    **FROM** *course*
    **WHERE** *dept_name* = 'Music')
    **EXCEPT**
    (**SELECT** *T.course_id*
    **FROM** *takes* **AS** *T*
    **WHERE** *S.ID* = *T.ID*));

# Test for Absence of Duplicate Tuples

- The **UNIQUE** construct tests whether a subquery has any duplicate tuples in its result
  - **UNIQUE** evaluates to "true" if a given subquery contains no duplicates
  - MySQL does not support the UNIQUE test (UNIQUE in MySQL is a constraint specifier)

- Find all courses that were offered at most once in 2017
  - **SELECT** *T.course_id*
    **FROM** *course* **AS** *T*
    **WHERE UNIQUE** ( **SELECT** *R.course_id*
    **FROM** *teaches* **AS** *R*
    **WHERE** *T.course_id*= *R.course_id* **AND** *R.year* = 2017);

# EOF

- Coming next:
  - SQL DDL (Data Definition Language)