MASTER SQL FOR DATA SCIENCE

Section 1 : Database Basics

- We are using Relational Database.
- Database is a collection of **Tables**.
- Database -> Tables -> Data
- **SQL**: Structured Query Language
- Supplier_ID **is uniquely identified**.
- Supplier_ID, Supplier_Name, City, State are **Attributes**.
- Columns contain only **same** type of information. These types could be **VarChar, INT, DATE...**

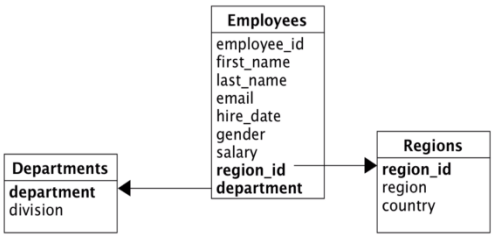
A database is a collection of tables

SUPPLIER_ID	SUPPLIER_NAME	CITY	STATE
100	The Computer Shop	Augusta	Georgia
200	Instant Assembly	Valdez	Alaska
300	Read Time LLC.	Redwood City	California
400	Roundhouse Inc.	New York City	New York
500	Smiths & Berries	Portland	Oregon
600	Hardware Experts	Yuma	Arizona
700	Strong Foods Inc.	Orlando	Florida
800	Cheffmens Inc.	Toledo	Ohio
900	Samwoods Drinks	Portland	Oregon

```
CREATE TABLE table_name (
    column1data_type,
    column2data_type,
    primary key (column_name));
```

```
INSERT INTO table_name values ( data, data, data, ......);
```

- Primary Key prevents:
  - Duplicate information
  - NULL information
- Only 1 Primary Key Per table.




```sql
create table employees (
    employee_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(50),
    hire_date DATE,
    department VARCHAR(17),
    gender VARCHAR(1),
    salary INT,
    region_id INT,
    primary key (employee_id)
);

insert into employees values (1, 'Berrie', 'Manueau', 'bmanueau0@dion.ne.jp', '2006-04-20', 'Sports', 'F', 154864, 4);
insert into employees values (2, 'Aeriell', 'McNee', 'amcnee1@google.es', '2009-01-26', 'Tools', 'F', 56752, 3);
insert into employees values (3, 'Sydney', 'Symonds', 'ssymonds2@hhs.gov', '2010-05-17', 'Clothing', 'F', 95313, 4);
insert into employees values (4, 'Avrom', 'Rowantree', null, '2014-08-02', 'Phones & Tablets', 'M', 119674, 7);
insert into employees values (5, 'Feliks', 'Morffew', 'fmorffew4@a8.net', '2003-01-14', 'Computers', 'M', 55307, 5);
insert into employees values (6, 'Bethena', 'Trow', 'btrow5@technorati.com', '2003-06-08', 'Sports', 'F', 134501, 3);
insert into employees values (7, 'Ardeen', 'Curwood', 'acurwood6@1und1.de', '2006-02-19', 'Clothing', 'F', 28995, 7);
insert into employees values (8, 'Seline', 'Dubber', 'sdubber7@t-online.de', '2012-05-28', 'Phones & Tablets', 'F', 101066, 3);
insert into employees values (9, 'Dayle', 'Trail', 'dtrail8@tamu.edu', '2003-03-01', 'First Aid', 'F', 82753, 1);
insert into employees values (10, 'Redford', 'Roberti', null, '2008-07-21', 'Clothing', 'M', 72225, 7);
insert into employees values (11, 'Nickey', 'Pointon', 'npointona@vistaprint.com', '2006-12-30', 'Jewelry', 'M', 126333, 7);
insert into employees values (12, 'Leonora', 'Casaroli', 'lcasarolib@plala.or.jp', '2013-07-22', 'Beauty', 'F', 99504, 3);
insert into employees values (13, 'Anetta', 'Arnao', null, '2009-05-23', 'Games', 'F', 38162, 1);
insert into employees values (14, 'Jodi', 'Hook', 'jhookd@booking.com', '2003-10-16', 'Tools', 'F', 126588, 2);
insert into employees values (15, 'Alyson', 'Franzonello', null, '2004-01-01', 'Furniture', 'F', 61256, 6);
insert into employees values (16, 'Merell', 'Yakovliv', 'myakovlivf@ucsd.edu', '2008-08-16', 'Movies', 'M', 78141, 7);
insert into employees values (17, 'Annora', 'Bendelow', 'abendelowg@google.com.hk', '2009-06-12', 'Toys', 'F', 75283, 5);
```

Section 2 : SQL Query Basics

```
SELECTcolumn1, column2 .... FROM table_name WHERE condition;
```

- **SELECT** * **FROM** table_name;
- **SELECT** employee_id, first_name, department **FROM** employees;
- **SELECT** * **FROM** employees **WHERE** department = 'Sports';
- ***SQL is not case sensitive, for the key words and table and column name. However the data is case sensitive.***
- ***Keywords (SELECT, FROM, WHERE) generally upper-case.***
- ***Table and column names are lower-case.***
- **SELECT** * **FROM** employees **WHERE** department **like**'%nitu%';
  SELECT * **FROM** employees **WHERE** department **like**'F%nitu%';
- **SELECT** * **FROM** employees **WHERE** salary > 100000;
  SELECT * **FROM** employees **WHERE** salary = 100000;

>, =, <, <=, >= are the options to be used.

SELECT * **FROM** employees **WHERE** 1 = 1; *->this condition is TRUE for all data so all data will be printed.*

SELECT * **FROM** employees **WHERE** 1 < 1; *->this condition is FALSE for all data nothing will be printed.*

- **SELECT** * **FROM** employees **WHERE** department = 'Clothing' **AND** salary > 90000;

  SELECT * **FROM** employees **WHERE** department = 'Clothing' **OR** salary > 90000;

  SELECT * **FROM** employees **WHERE** salary < 40000 **AND (**department = 'Clothing' **OR** department = 'Pharmacy'**);**

IN, NOT IN, IS NULL, BETWEEN

- **SELECT** * **FROM** employees **WHERE NOT** department = 'Sports';

  SELECT * **FROM** employees **WHERE** department **<>** 'Sports';

  SELECT * **FROM** employees **WHERE NOT** department **<>** 'Sports';
- **SELECT** * **FROM** employees **WHERE** NULL **=** NULL; *this will print empty*

  SELECT * **FROM** employees **WHERE** NULL **!=** NULL; *this will print empty*

  SELECT * **FROM** employees **WHERE** email **is** NULL; *this is going to print if email data is null.*

  SELECT * **FROM** employees **WHERE** email **is NOT** NULL;
- **SELECT** * **FROM** employees **WHERE** department = 'Sports' **OR** department = 'Toys' **OR** department = 'Garden'

  SELECT * **FROM** employees **WHERE** department **IN** ('Sports', 'Toys', 'Garden');
- **SELECT** * **FROM** employees **WHERE** salary **BETWEEN** 80000 **and** 100000; *this doesn't include 80000 and 100000 because we say between them.*

EXERCISE

- **SELECT** first_name, email **FROM** employees **WHERE** gender = 'F' **AND** department = 'Tools' **AND** salary >110000;
- **SELECT** first_name, hire_date **FROM** employees **WHERE** salary > 165000 **OR** (department = 'Sports' **AND** gender = 'M');
- **SELECT** first_name, hire_date **FROM** employees **WHERE** hire_date **BETWEEN** '2002-01-01' **AND** '2004-01-01';
- **SELECT** * **FROM** employees **WHERE** (department = 'Automotive' **AND** gender = 'M' **AND** salary **BETWEEN** 40000 AND 100000) **OR** (gender = 'F' **AND** department = 'Toys');
- **-- Hey just review this query for me -- means comment line**
- **SELECT** * **FROM** employees **WHERE** (department = 'Automotive' **AND** gender = 'M' **AND** salary **BETWEEN** 40000 AND 100000) **OR** (gender = 'F' **AND** department = 'Toys');

ORDER BY, LIMIT, DISTINCT AND RENAMING COLUMNS

- **SELECT** * **FROM** employees **ORDER BY** employee_id; *default is ascending (Smallest to Largest)*

  SELECT * **FROM** employees **ORDER BY** employee_id

  SELECT * **FROM** employees **ORDER BY** employee_id **DESC**

  SELECT * **FROM** employees **ORDER BY** department **ASC**

  SELECT * **FROM** employees **ORDER BY** department **DESC**

  SELECT * **FROM** employees **ORDER BY** salary **DESC**
- **SELECT** department **FROM** employees

  SELECT DISTINCT department **FROM** employees

  SELECT DISTINCT department **FROM** employees **ORDER BY** department **DESC**

  SELECT DISTINCT department **FROM** employees **ORDER BY** 1
- **SELECT DISTINCT** department **FROM** employees **ORDER BY** 1 **LIMIT** 10

  SELECT DISTINCT department **FROM** employees **ORDER BY** 1 **FETCH FIRST** 10 **ROWS ONLY**; *same as above*

  SELECT DISTINCT department **FROM** employees **ORDER BY** department **DESC LIMIT** 10

  SELECT DISTINCT department **FROM** employees **ORDER BY** department **DESC FETCHFIRST** 10 **ROWS ONLY**; *same as above*
- **SELECT DISTINCT** department **AS** sorted_departments **FROM** employees **ORDER BY** 1 **FETCHFIRST** 3 **ROWS ONLY**; *changing the column name*

  SELECT first_name, last_name, department, salary **AS** yearly_salary **FROM** employees

  SELECT first_name, last_name **AS** "Last Name", department, salary **AS** "Yearly Salary" **FROM** employees; *in here you need to put in double quotes because there is space in the new naming of the columns.*

Assignment-2

Write a query to display the names of those students that are between the ages of 18 and 20.

```
SELECT student_name FROM students WHERE age BETWEEN 18 AND 20;
```

Write a query to display all of those students that contain the letters "ch" in their name or their name ends with the letters "nd".

```
SELECT * FROM students WHERE student_name like '%ch%' OR student_name like '%nd';
```

Write a query to display the name of those students that have the letters "ae" or "ph" in their name and are NOT 19 years old.

```
SELECT student_name FROM students WHERE (student_name like '%ae%' OR student_name like '%ph%')AND age
!= 19;
```

Write a query that lists the names of students sorted by their age from largest to smallest.

```
SELECT student_name FROM students ORDER BY age DESC;
```

Write a query that displays the names and ages of the top 4 oldest students.

```
SELECT student_name, age FROM students ORDER BY age DESC LIMIT 4;
```

ADVANCED:

Write a query that returns students based on the following criteria:

The student must not be older than age 20 if their student_no is either between 3 and 5 or their student_no is 7. Your query should also return students older than age 20 but in that case they must have a student_no that is at least 4.

```
SELECT * FROM students WHERE AGE <= 20 AND ( student_no BETWEEN 3 AND 5 OR student_no = 7 )OR (AGE > 2
0 AND student_no>= 4);
```

Section 3 : Using Functions

UPPER(), LOWER(), LENGTH(), TRIM() + Boolean Expressions & Concatenation

- **SELECT** * **FROM** employees
- **SELECT UPPER**(first_name), **LOWER**(department) **FROM** employees
- **SELECT LENGTH**(first_name), **LOWER**(department) **FROM** employees
- **SELECT** ' HELLO THERE '
- **SELECT TRIM**(' HELLO THERE ') *-- This is handy to clean data.Iit cleans the space at the beginning and start.*
- **SELECT LENGTH**(**TRIM**(' HELLO THERE ')) *-- Now we can have the number of characters, excludes gaps.*
- **SELECT LENGTH**(' HELLO THERE ') *--Now we can have the number of characters, includes the gaps at the beginning and end, and middle.*
- **SELECT** first_name **||** last_name **FROM** employees *-- concatenating the data*
- **SELECT** first_name **||** ' ' **||** last_name **FROM** employees *-- concatenating the data nicely*
- **SELECT** first_name **||** ' ' **||** last_name **AS** full_name **FROM** employees *-- concatenating the data nicely*
- **SELECT** first_name **||** ' ' **||** last_name full_name, department **FROM** employees *-- concatenating the data nicely, no need AS*
- **SELECT** first_name **||** ' ' **||** last_name full_name, (salary > 140000) **FROM** employees *-- Boolean creates new column*
- **SELECT** first_name **||** ' ' **||** last_name full_name, (salary > 140000) **FROM** employees **ORDER BY** salary **DESC** *-- Boolean and order by*
- **SELECT** first_name **||** ' ' **||** last_name full_name, (salary > 140000) is_highly_paid **FROM** employees **ORDER BY** salary **DESC***-- Boolean and order by*
- **SELECT** department, ('Clothing' **IN** (department, first_name)) **FROM** employees *-- check values in that parenthesis group and returns TRUE or FALSE*
- **SELECT** ('Clothing' **IN** ('clothing', 'furniture', 'phones')) *-- returns FALSE, because the first letter is upper case.*
- **SELECT** ('Clothing' **IN** ('Clothing', 'furniture', 'phones')) *-- returns TRUE.*
- **SELECT** department, (department **LIKE** '%oth%') **FROM** employees *-- Returns second column with True or False values if department contains 'oth'*

SUBSTRING(), REPLACE(), POSITION() and COALSECE()

- SELECT 'This is test data' test_data;
- SELECT SUBSTRING('This is test data' FROM 1 FOR 4) test_data_extracted; *-- Column name test_data_extracted, result: 'This' is the extracted data. From is the position we want to start and FOR specifies how many characters do you want to extracted.*
  SELECT SUBSTRING('This is test data' FROM 9 FOR 4) test_data_extracted;*-- Column name test_data_extracted, result: 'test' is the extracted data.*
  SELECT SUBSTRING('This is test data' FROM 4) test_data_extracted; *-- Column name test_data_extracted, result: 's is test data' is the extracted data.*
- SELECT department, REPLACE(department, 'Clothing', 'Attire') modified_data FROM departments; *-- it will replace the clothing data to attire data in department column and give a new name as modified_data. It won't change the database.*
  SELECT department, REPLACE(department, 'Clothing', 'Attire') modified_data, department || ' department' AS "Complete Department Name" FROM departments;
- SELECT email, SUBSTRING(email, POSITION('@' IN email) + 1) formated_text FROM employees *-- we will extract the @ sign and retrieve the domain names.*
- SELECT COALESCE(email, 'NONE') AS email FROM employees *-- Coalesce function can change the NULL data whenever it sees it*

MIN(), MAX(), AV(), SUM(), COUNT()

These grouping functions return only a single row.

- SELECT MAX(salary) FROM employees;
- SELECT MIN(salary) FROM employees;
- SELECT AVG(salary) FROM employees;
- SELECT ROUND (AVG(salary)) FROM employees;
- SELECT COUNT(employee_id) FROM employees *-- Use primary key data since there is no NULL value in there.*
- SELECT COUNT(email) FROM employees*-- It will print how many employees have email.*
- SELECT COUNT(*) FROM employees *-- As long as there is a data, then it will count it. Output is 1000 for the employees table.*
- SELECT SUM(salary) FROM employees *-- you extract yearly budget that you pay to your employees.*
- SELECT SUM(salary) FROM employees WHERE department = 'Clothing'
- SELECT SUM(salary) FROM employees WHERE department = 'Toys'

Assignment-3

1. Write a query against the professors table that can output the following in the result:
   "Chong works in the Science department"

SELECT last_name || ' ' || 'works in the '|| department ||' department' FROM professors

2. Write a SQL query against the professors table that would return the following result:

   "It is false that professor Chong is highly paid"
   "It is true that professor Brown is highly paid"
   "It is false that professor Jones is highly paid"
   "It is true that professor Wilson is highly paid"
   "It is false that professor Miller is highly paid"
   "It is true that professor Williams is highly paid"

NOTE: A professor is highly paid if they make greater than 95000.

SELECT 'It is ' || (salary > 95000) ||' that professor ' || last_name || ' is highly paid' FROM professors

3. Write a query that returns all of the records and columns from the professors table but shortens the department names to only the first three characters in upper case.

SELECT last_name, UPPER(SUBSTRING(department, 1, 3)) as department, salary, hire_date FROM professors

4. Write a query that returns the highest and lowest salary from the professors table excluding the professor named 'Wilson'.

SELECT MAX(salary) as higest_salary, MIN(salary) as lowest_salary FROM professors WHERE last_name != 'Wilson'

5. Write a query that will display the hire date of the professor that has been teaching the longest.

SELECT MIN(hire_date) FROM professors

Section 4: Grouping Data and Computing Aggregates

GROUP BY & HAVING

- CREATE TABLE cars(make varchar(10));
  INSERT INTO cars VALUES('HONDA');
  INSERT INTO cars VALUES('HONDA');
  INSERT INTO cars VALUES('HONDA');
  INSERT INTO cars VALUES('TOYOTA');
  INSERT INTO cars VALUES('TOYOTA');
  INSERT INTO cars VALUES('NISSAN');
  INSERT INTO cars VALUES(NULL);
  INSERT INTO cars VALUES(NULL);
  INSERT INTO cars VALUES(NULL);
  INSERT INTO cars VALUES(NULL);
  SELECT * FROM cars;
  SELECT COUNT(*), make FROM cars GROUP BY make; *-- how many of each group we have*
  SELECT make FROM cars GROUP BY make;
  SELECT make, COUNT(*) FROM cars GROUP BY make;
- SELECT SUM(salary) FROM employees
  SELECT COUNT(DISTINCT department) FROM employees
- SELECT department, SUM(salary) FROM employees WHERE 1=1 GROUP BY department
- SELECT department, SUM(salary) FROM employees WHERE region_id IN (4,5,6,7) GROUP BY department
- SELECT department, COUNT(*) FROM employees GROUP BY department
- SELECT department, COUNT(employee_id) FROM employees GROUP BY department
- SELECT department, COUNT(employee_id) total_number_employees, ROUND(AVG(salary)) avg_sal, MIN(salary) min_sal, MAX(salary) max_sal FROM employees GROUP BY department ORDER BY total_number_employees DESC
- SELECT department, COUNT(employee_id) total_number_employees, ROUND(AVG(salary)) avg_sal, MIN(salary) min_sal, MAX(salary) max_sal FROM employees WHERE salary > 70000 GROUP BY department ORDER BY total_number_employees DESC
- SELECT department, gender, COUNT(*) FROM employees GROUP BY department *-- This will throw error. You have to add gender column to the GROUP BY clause.*
- SELECT department, gender, COUNT(*) FROM employees GROUP BY department, gender ORDER BY department *-- Now, you grouped by department and gender. This will give you each department's gender employee numbers.*
- SELECT department, salary, COUNT(*) FROM employees GROUP BY department ORDER BY department *– Again, this will throw error. You have to add salary column to the GROUP BY clause.*
- SELECT department, COUNT(*) FROM employees GROUP BY department HAVING count(*)>35 ORDER BY department *-- if you want to filter aggregated data, use HAVING. HAVING must be after GROUP BY.*

EXERCISE - GROUP BY & HAVING

- SELECT first_name, COUNT(*) FROM employees GROUP BY first_name
- SELECT first_name, COUNT(*) FROM employees GROUP BY first_name HAVING count(*) > 2
- SELECT department FROM employees GROUP BY department
- SELECT SUBSTRING(email, POSITION('@' IN email) + 1) email_domain, COUNT(*) FROM employees WHERE email IS NOT NULL GROUP BY SUBSTRING(email, POSITION('@' IN email) + 1) ORDER BY COUNT(*) DESC *– Good practice example*
- SELECT gender, region_id, MIN(salary) min_salary, MAX(salary) max_salary, ROUND(AVG(salary)) avg_salary FROM employees GROUP BY gender, region_id ORDER BY gender DESC, region_id ASC

Assignment-4

1. Write a query that displays only the state with the largest amount of fruit supply.
   SELECT state FROM fruit_imports GROUP BY state ORDER BY SUM(supply) desc LIMIT 1

2. Write a query that returns the most expensive cost_per_unit of every season. The query should display 2 columns, the
   SELECT season, MAX(cost_per_unit) highest_cost_per_unit FROM fruit_imports GROUP BY season

3. Write a query that returns the state that has more than 1 import of the same fruit.
   SELECT state FROM fruit_imports GROUP BY state, name HAVING COUNT(name)>1

4. Write a query that returns the seasons that produce either 3 fruits or 4 fruits.
   SELECT season, COUNT(name) FROM fruit_imports GROUP BY season HAVING count(name)=3 OR count(name)=4

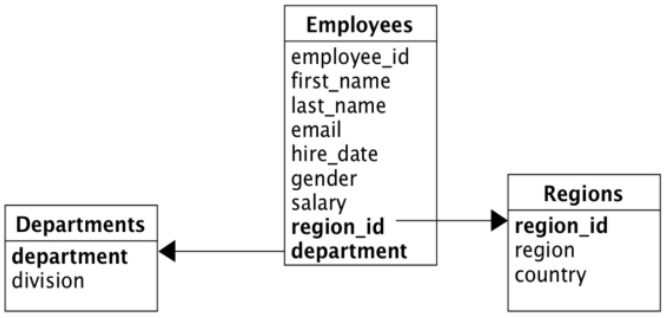
5. Write a query that takes into consideration the **supply** and **cost_per_unit** columns for determining the total cost and returns the most expensive state with the total cost.
   SELECT state, SUM(supply *cost_per_unit) total_cost FROM fruit_imports GROUP BY state ORDER BY total_cost desc LIMIT 1

6. Execute the below SQL script and answer the question that follows:

```
CREATE table fruits (fruit_name varchar(10));
INSERT INTO fruits VALUES ('Orange');
INSERT INTO fruits VALUES ('Apple');
INSERT INTO fruits VALUES (NULL);
INSERT INTO fruits VALUES (NULL);
```

   Write a query that returns the count of 4. You'll need to count on the column fruit_name and not use COUNT(*)
   HINT: You'll need to use an additional function inside of count to make this work.
   SELECT COUNT(COALESCE(fruit_name,'SOMEVALUE')) FROM fruits;

Section 5: Using Subqueries



SUBQUERIES WITH ANY AND ALL

- **SELECT** first_name, last_name, * **FROM** employees
  SELECT department **FROM** employees, departments *-- This will throw error. It is ambiguous that which table you want to pull it from.*
  SELECT departments.department **FROM** employees, departments *--put table_name.column_name, this will bring the department column from departments table.*
  SELECT d.department **FROM** employees e, departments d *-- you can alias the table name*
  SELECT * **FROM** employees **WHERE** department **NOT IN** ('DEPART1', 'DEPART2','DEPART3')
  SELECT * **FROM** employees **WHERE** department **NOT IN** (**SELECT** department **FROM** departments)
  SELECT * **FROM** (**SELECT** * **FROM** employees **WHERE** salary > 150000) *-- This will throw error. You need to name the table*
  SELECT a.first_name, a.salary **FROM** (**SELECT** * **FROM** employees **WHERE** salary > 150000) a
  SELECT a.first_name, a.salary **FROM** (**SELECT** first_name employee_name, salary yearly_salary **FROM** employees **WHERE** salary > 150000) a *-- this will no longer work, because we changed the name on the data source.*
  SELECT a.employee_name, a.yearly_salary **FROM** (**SELECT** first_name employee_name, salary yearly_salary **FROM** employees **WHERE** salary > 150000) a
  SELECT a.employee_name, a.yearly_salary **FROM** (**SELECT** first_name employee_name, salary yearly_salary **FROM** employees **WHERE** salary > 150000) a, (**SELECT** department employee_name **FROM** departments) b
  SELECT * **FROM** employees **WHERE** department **IN** (**SELECT** department **FROM** departments)
  SELECT * **FROM** (**SELECT** department **FROM** departments) a
  SELECT first_name, last_name, salary, (**SELECT** first_name **FROM** employees **LIMIT** 1) **FROM** employees
  SELECT * **FROM** employees
  SELECT * **FROM** employees **WHERE** department **IN** (**SELECT** department **FROM** departments **WHERE** division = 'Electronics')
  SELECT * **FROM** regions
  SELECT * **FROM** employees **WHERE** salary>130000 **AND** region_id **IN** (**SELECT** region_id **FROM** regions **WHERE** country **IN** ('Asia', 'Canada'))

```
SELECT first_name, department, (SELECT MAX(salary) FROM employees), (SELECT MAX(salary) FROM employees) - salary FROM employees WHERE salary>130000 AND region_id IN (SELECT region_id FROM regions WHERE country IN ('Asia', 'Canada'))
SELECT * FROM regions
SELECT * FROM employees WHERE region_id IN (SELECT region_id FROM regions WHERE country= 'United States')
```

- `SELECT * FROM employees WHERE region_id > (SELECT region_id FROM regions WHERE country= 'United States')` ***-- Error. You can't compare region_id with =, >, <, with multiple values. It should be single value.***

```
SELECT * FROM employees WHERE region_id >ANY (SELECT region_id FROM regions WHERE country= 'United States')
SELECT * FROM employees WHERE region_id >ALL (SELECT region_id FROM regions WHERE country= 'United States')
SELECT * FROM departments
SELECT * FROM employees WHERE (department = ANY (SELECT department FROM departments WHERE division='Kids')) AND hire_date>ALL (SELECT hire_date FROM employees WHERE department='Maintenance')
SELECT salary FROM (SELECT salary, COUNT(*) FROM employees GROUP BY salary ORDER BY COUNT(*) DESC, salary DESC LIMIT 1) a
SELECT salary FROM employees GROUP BY salary HAVING COUNT(*) >= ALL (SELECT COUNT(*) FROM employees GROUP BY salary) ORDER BY salary DESCLIMIT 1
```

-
```
CREATE  table dupes (id integer, name varchar(10));
INSERT INTO dupes VALUES(1, 'FRANK');
INSERT INTO dupes VALUES(2, 'FRANK');
INSERT INTO dupes VALUES(3, 'ROBERT');
INSERT INTO dupes VALUES(4, 'ROBERT');
INSERT INTO dupes VALUES(5, 'SAM');
INSERT INTO dupes VALUES(6, 'FRANK');
INSERT INTO dupes VALUES(7, 'PETER');
SELECT * FROM dupes
SELECT * FROM dupes WHERE id IN (SELECT min(id) FROM dupes GROUP BY name)
DELETE FROM dupes WHERE id NOT IN (SELECT min(id) FROM dupes GROUP BY name)
DROP table dupes
```

-
```
SELECT * FROM employees
SELECT ROUND(AVG(salary)) FROM employees WHERE salary NOT IN ((SELECT MIN(salary) FROM employees), (SELECT MAX(salary) FROM employees))
```

Assignment-5

1. Is the students table directly related to the courses table? Why or why not?

   The students table is not directly related to the courses table. The students table just contains student details. The courses table just contains courses information. The table that relates both the students table and courses table is the student_enrollment table. What student is enrolled in what course is captured in the student_enrollment table.

2. Using subqueries only, write a SQL statement that returns the names of those students that are taking the courses **Physics** and **US History.**
   NOTE: Do not jump ahead and use joins. I want you to solve this problem using only what you've learned in this section.

```
SELECT student_name
FROM students WHERE student_no
            IN (SELECT student_no
              FROM student_enrollment
              WHERE course_no
              IN ( SELECT course_no
                FROM courses
                WHERE course_title
                  IN ('Physics', 'US History')));
```

3. Using subqueries only, write a query that returns the name of the student that is taking the highest number of courses.

   NOTE: Do not jump ahead and use joins. I want you to solve this problem using only what you've learned in this section.

```
SELECT student_name FROM students WHERE student_no IN
          (SELECT student_no FROM ( SELECT student_no, COUNT(course_no) course_cnt
          FROM STUDENT_ENROLLMENT GROUP BY student_no  ORDER BY course_cnt desc  LIMIT 1) a )
```

4. Answer **TRUE** or **FALSE** for the following statement:

Subqueries can be used in the FROM clause and the WHERE clause but cannot be used in the SELECT Clause.

FALSE. Subqueries can be used in the FROM, WHERE, SELECT and even the HAVING clause.

5. Write a query to find the student that is the oldest. You are not allowed to use LIMIT or the ORDER BY clause to solve this problem.

```
SELECT *
FROM students
WHERE age = (SELECT MAX(age) FROM students)
```

Section 6: Using the CASE Clause In Interesting Ways

CASE CLAUSE –

```
CASE...
    WHEN...THEN...
    WHEN...THEN...
    ELSE...
END
```

- ```
  SELECT FIRST_NAME, SALARY,
      CASE    WHEN SALARY < 100000 THEN 'UNDER PAID'
              WHEN SALARY > 100000  AND SALARY < 1600000 THEN 'PAID WELL'
              WHEN SALARY > 160000 THEN 'EXECUTIVE'
              ELSE 'UNPAID'
              END AS CATEGORY
  FROM EMPLOYEES ORDER BY SALARY DESC
  ```

- ```
  SELECT A.CATEGORY,  COUNT(*)
  FROM
      (SELECT FIRST_NAME, SALARY,
              CASE
              WHEN SALARY < 100000 THEN 'UNDER PAID'
              WHEN SALARY > 100000  AND SALARY < 160000 THEN 'PAID WELL'
              WHEN SALARY > 160000 THEN 'EXECUTIVE'
              ELSE 'UNPAID'
              END AS CATEGORY
      FROM EMPLOYEES
      ORDER BY SALARY DESC) A
  GROUP BY A.CATEGORY
  ```

- ```
  SELECT SUM(CASE WHEN SALARY < 100000 THEN 1 ELSE 0 END) AS UNDER_PAID,
      SUM(CASE WHEN SALARY > 100000 AND SALARY < 150000 THEN 1 ELSE 0 END) AS PAID_WELL,
      SUM(CASE WHEN SALARY > 150000 THEN 1 ELSE 0 END) AS EXECUTIVE
  FROM EMPLOYEES
  ```

- ```
  SELECT department, COUNT(*) FROM employees
  WHERE department IN ('Sports', 'Tools', 'Clothing', 'Computers') GROUP BY department
  ```

- ```
  SELECT SUM(CASE WHEN department = 'Sports' THEN 1 ELSE 0 END) as Sports_Employees,
          SUM(CASE WHEN department = 'Tools' THEN 1 ELSE 0 END) as Tools_Employees,
          SUM(CASE WHEN department = 'Clothing' THEN 1 ELSE 0 END) as Clothing_Employees,
          SUM(CASE WHEN department = 'Computers' THEN 1 ELSE 0 END) as Computers_Employees
  FROM employees
  ```
  ```

- ```
  SELECT first_name,
  CASE WHEN region_id = 1 THEN (SELECT country FROM regions WHERE region_id=1) END region_1,
  CASE WHEN region_id = 2 THEN (SELECT country FROM regions WHERE region_id=2) END region_2,
  CASE WHEN region_id = 3 THEN (SELECT country FROM regions WHERE region_id=3) END region_3,
  CASE WHEN region_id = 4 THEN (SELECT country FROM regions WHERE region_id=4) END region_4,
  CASE WHEN region_id = 5 THEN (SELECT country FROM regions WHERE region_id=5) END region_5,
  CASE WHEN region_id = 6 THEN (SELECT country FROM regions WHERE region_id=6) END region_6,
  CASE WHEN region_id = 7 THEN (SELECT country FROM regions WHERE region_id=7) END region_7
  FROM employees
  ```

- ```
  SELECT COUNT(a.region_1) + COUNT(a.region_2) + COUNT(a.region_3) as United_States, COUNT(a.region_4) + COUNT(a.region_5)
  as Asia, COUNT(a.region_6)+COUNT(a.region_7) as Canada
  FROM (SELECT first_name,
  CASE WHEN region_id = 1 THEN (SELECT country FROM regions WHERE region_id=1) END region_1,
  CASE WHEN region_id = 2 THEN (SELECT country FROM regions WHERE region_id=2) END region_2,
  CASE WHEN region_id = 3 THEN (SELECT country FROM regions WHERE region_id=3) END region_3,
  CASE WHEN region_id = 4 THEN (SELECT country FROM regions WHERE region_id=4) END region_4,
  CASE WHEN region_id = 5 THEN (SELECT country FROM regions WHERE region_id=5) END region_5,
  CASE WHEN region_id = 6 THEN (SELECT country FROM regions WHERE region_id=6) END region_6,
  CASE WHEN region_id = 7 THEN (SELECT country FROM regions WHERE region_id=7) END region_7
  FROM employees) a
  ```

Assignment-6

1. Write a query that displays 3 columns. The query should display the fruit and it's total supply along with a category of either **LOW**, **ENOUGH** or **FULL**. *Low* category means that the total supply of the fruit is less than 20,000. The *enough* category means that the total supply is between 20,000 and 50,000. If the total supply is greater than 50,000 then that fruit falls in the *full* category.

   ```
   SELECT name, total_supply,
   CASE WHEN total_supply< 20000 THEN 'LOW'
       WHEN total_supply>= 20000 AND total_supply<= 50000 THEN 'ENOUGH'
       WHEN total_supply> 50000 THEN 'FULL'
   END as category
   FROM (
   SELECT name, sum(supply) total_supply
   FROM fruit_imports
   GROUP BY name
      ) a
   ```

2. Taking into consideration the supply column and the cost_per_unit column, you should be able to tabulate the total cost to import fruits by each season. The result will look something like this:

   "Winter" "10072.50"
   "Summer" "19623.00"
   "All Year" "22688.00"
   "Spring" "29930.00"
   "Fall" "29035.00"
   Write a query that would transpose this data so that the seasons become columns and the total cost for each season fills the first row?

   ```
   SELECT SUM(CASE WHEN season = 'Winter' THEN total_cost END) AS Winter_total,
               SUM(CASE WHEN season = 'Summer' THEN total_cost END) AS Summer_total,
               SUM(CASE WHEN season = 'Spring' THEN total_cost END) AS Spring_total,
               SUM(CASE WHEN season = 'Fall' THEN total_cost END) AS Spring_total,
               SUM(CASE WHEN season = 'All Year' THEN total_cost END) AS Spring_total
   FROM (
   SELECT season, SUM(supply * cost_per_unit) total_cost
   FROM fruit_imports
   GROUP BY season  ) a
   ```

Section 7: Advanced Query Techniques using Correlated Subqueries

CORRALETED SUBQUERIES

- **SELECT** first_name, salary **FROM** employees **WHERE** salary > (**SELECT ROUND**(**AVG**(salary)) **FROM** employees *-- sub query example*
- **SELECT** first_name, salary **FROM** employees e1 **WHERE** salary > (**SELECT ROUND**(**AVG**(salary)) **FROM** employees e2 **WHERE** e1.department = e2.department) *-- every single query will trigger the sub query. So, for every data will be checked with their own department.*
- **SELECT** first_name, salary **FROM** employees e1 **WHERE** salary > (**SELECT ROUND**(**AVG**(salary)) **FROM** employees e2 **WHERE** e1.region_id = e2.region_id) *-- every single query will triger the subquery. so, for every data will be checked with their own region.*
- **SELECT**first_name, department, salary, (**SELECT ROUND**(**AVG**(salary)) **FROM** employees e2 **WHERE** e1.department = e2.department) **as** avg_department_salary**FROM** employees e1 *-- corraleted subquery will always run for the outer query and it will use the outer query information.*
- **SELECT** department **FROM** (**SELECT** department, **COUNT**(*) num_emp**FROM** employees **GROUP BY** department) a **WHERE** num_emp> 38 *-- my answer*
- **SELECT** department **FROM** departments d **WHERE** 38 < (**SELECT COUNT**(*) **FROM** employees e **WHERE** e.department= d.department) *-- result 13 departments.*
- **SELECT** department **FROM** employees e1 **WHERE** 38 < (**SELECT COUNT**(*) **FROM** employees e2 **WHERE** e1.department= e2.department) *-- if we use employees table, then it will look for all employees. 603 results. More running time...*
- **SELECT DISTINCT** department **FROM** employees e1 **WHERE** 38 < (**SELECT COUNT**(*) **FROM** employees e2 **WHERE** e1.department= e2.department) *-- Here we use DISTINCT command. result 13 departments. More running time...*
- **SELECT DISTINCT** department **FROM** employees e1 **WHERE** 38 < (**SELECT COUNT**(*) **FROM** employees e2 **WHERE** e1.department= e2.department) **GROUP BY** department *-- Here we use GROUP BY command. result 13 departments. More running time...*
- **SELECT** department, (**SELECT MAX**(salary) **FROM** employees **WHERE** department = d.department) **FROM** departments d **WHERE** 38 < (**SELECT COUNT**(*) **FROM** employees e2 **WHERE** e2.department = d.department)

EXERCISES

- **SELECT** first_name, salary **FROM** employees **WHERE** salary > (**SELECT ROUND**(**AVG**(salary)) **FROM** employees *-- subquery example*

SELECT first_name, salary **FROM** employees e1 **WHERE** salary > (**SELECT ROUND**(**AVG**(salary)) **FROM** employees e2 **WHERE** e1.department = e2.department) *-- every single query will triger the subquery. so, for every data will be checked with their own department.*

```sql
SELECT department, first_name, salary,
    CASE WHEN salary = max_by_department THEN 'HIGHEST SALARY'
         WHEN salary = min_by_department THEN 'LOWEST SALARY'
    END AS salary_in_department
FROM
(SELECT department, first_name, salary,
        (SELECT MAX(salary) FROM employees e2
        WHERE e1.department = e2.department) AS max_by_department,
        (SELECT MIN(salary) FROM employees e2
        WHERE e1.department = e2.department) AS min_by_department
FROM employees e1
ORDER BY department) a
WHERE salary = max_by_department OR salary = min_by_department
```

Data Output | Explain | Messages | Notifications

	department character varying (17)	first_name character varying (50)	salary integer	salary_in_department text
1	Automotive	Mill	162522	HIGHEST SALARY
2	Automotive	Laurie	29752	LOWEST SALARY
3	Beauty	Orland	162845	HIGHEST SALARY
4	Beauty	Willabella	22053	LOWEST SALARY
5	Books	Chloris	41549	LOWEST SALARY
6	Books	Sephira	159561	HIGHEST SALARY
7	Camping	Eugenia	26747	LOWEST SALARY
8	Camping	Riley	166569	HIGHEST SALARY
9	Children Clothing	Timotheus	23159	LOWEST SALARY
10	Children Clothing	Yancy	158546	HIGHEST SALARY

Section 8: Working with Multiple Tables

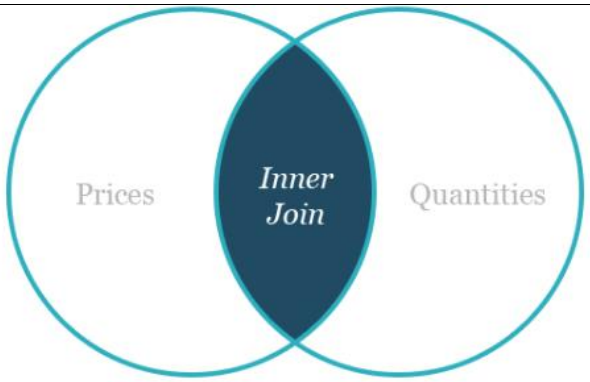


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices INNER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR INNER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66

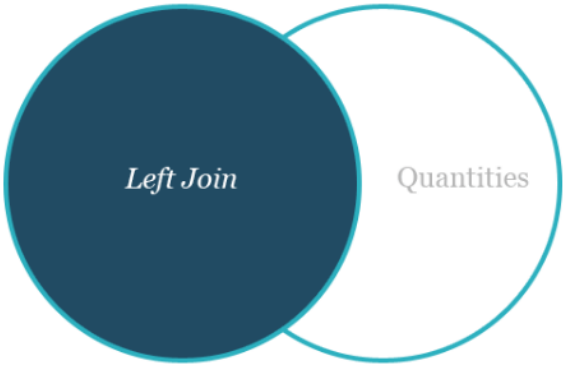


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices LEFT OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR LEFT OUTER JOIN

PRODUCT	PRICE	QUANTITY
Potatoes	$3	45
Avocados	$4	63
Kiwis	$2	19
Onions	$1	20
Melons	$5	66
Oranges	$5	NULL
Tomatoes	$6	NULL

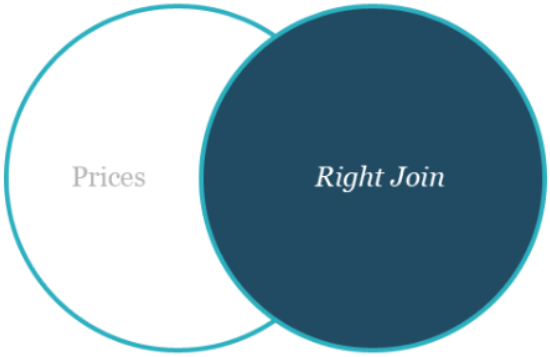


TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices RIGHT OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR RIGHT OUTER JOIN

PRICE	PRODUCT	QUANTITY
$3	Potatoes	45
$4	Avocados	63
$2	Kiwis	19
$1	Onions	20
$5	Melons	66
NULL	Broccoli	27
NULL	Squash	92

TABLE 1: PRICES

PRODUCT	PRICE
Potatoes	$3
Avocados	$4
Kiwis	$2
Onions	$1
Melons	$5
Oranges	$5
Tomatoes	$6

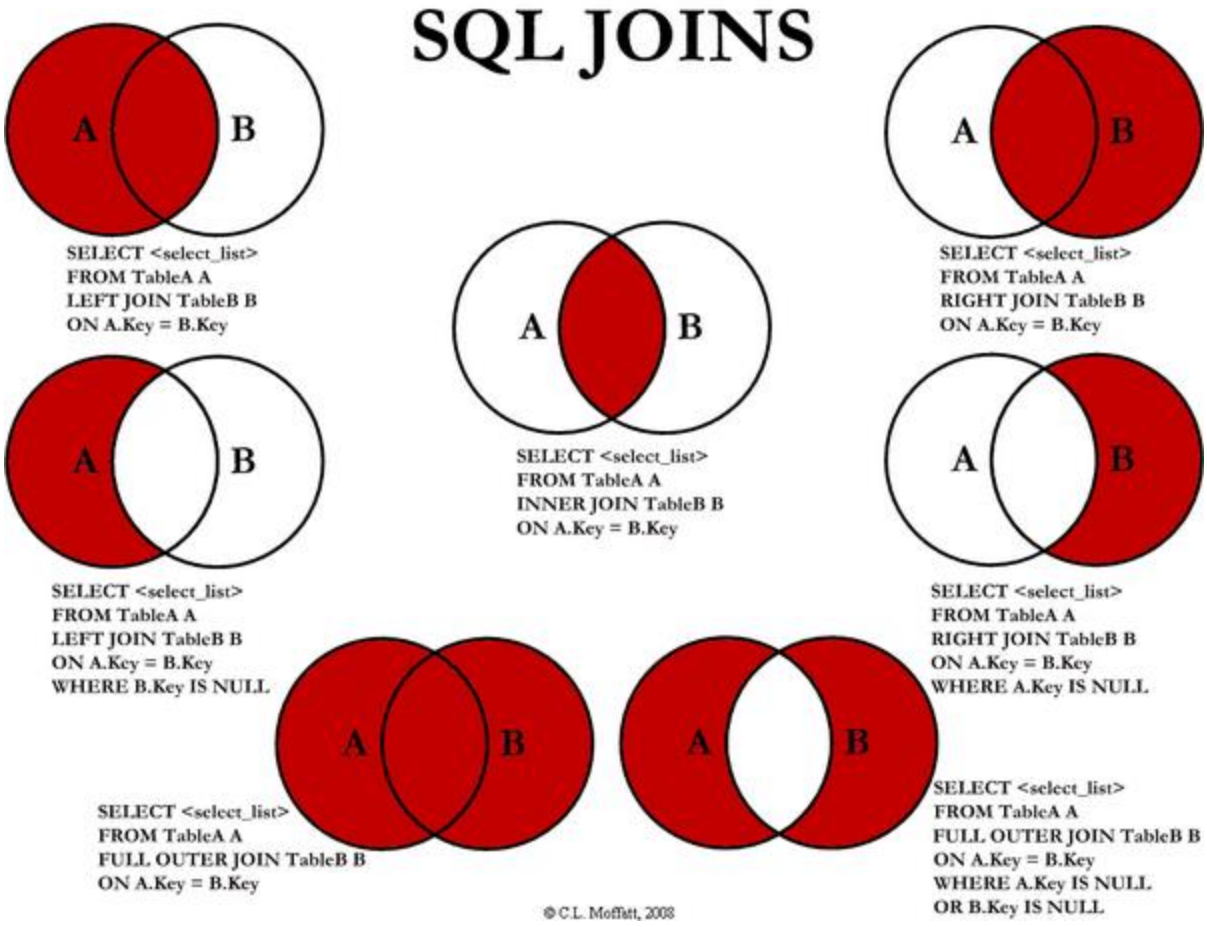
TABLE 2: QUANTITIES

PRODUCT	QUANTITY
Potatoes	45
Avocados	63
Kiwis	19
Onions	20
Melons	66
Broccoli	27
Squash	92

```
SELECT Prices.*, Quantities.Quantity
FROM Prices FULL OUTER JOIN Quantities
ON Prices.Product = Quantities.Product;
```

QUERY RESULT FOR FULL OUTER JOIN

PRICES.PRODUCT	PRICE	QUANTITIES.PRODUCT	QUANTITY
Potatoes	$3	Potatoes	45
Avocados	$4	Avocados	63
Kiwis	$2	Kiwis	19
Onions	$1	Onions	20
Melons	$5	Melons	66
Oranges	$5	NULL	NULL
Tomatoes	$6	NULL	NULL
NULL	NULL	Broccoli	27
NULL	NULL	Squash	92



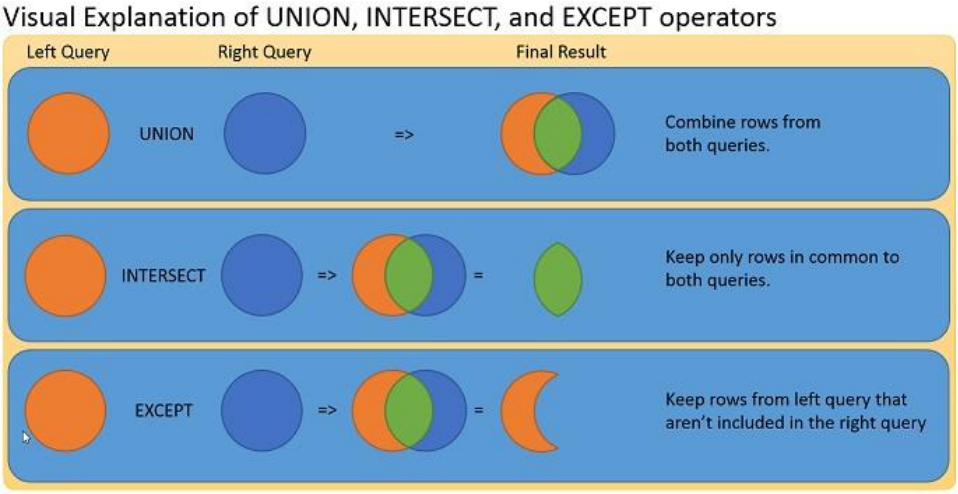
INNER and OUTER JOINS

- **SELECT** first_name, country **FROM** employees, regions **WHERE** employees.region_id = regions.region_id *-- joining tables based on the columns*
- **SELECT** first_name, email, e.department, division, country **FROM** employees e, departments d, regions r **WHERE** e.department = d.department **AND** e.region_id = r.region_id **AND** email **IS NOT NULL** *-- 3 tables joined. shared columns should be identfied which table it is coming from*
- **SELECT** country, **COUNT**(employee_id) **FROM** employees e, regions r **WHERE** e.region_id = r.region_id **GROUP BY** country
- **SELECT** country, **COUNT**(employee_id) **FROM** employees e, (**SELECT** * **FROM** regions) r **WHERE** e.region_id = r.region_id **GROUP BY** country
- **SELECT** first_name, country **FROM** employees **INNER JOIN** regions **ON** employees.region_id = regions.region_id
- **SELECT** first_name, email, division **FROM** employees **INNER JOIN** departments **ON** employees.department = departments.department **WHERE** email **IS NOT NULL**
- **SELECT** first_name, email, division, country **FROM** employees
  INNER JOIN departments **ON** employees.department = departments.department
  INNER JOIN regions **ON** employees.region_id = regions.region_id
  WHERE email **IS NOT** NULL

- **SELECT DISTINCT** department **FROM** employees *-- 27 departments*
- **SELECT DISTINCT** department **FROM** department *-- 24 departments*
- **SELECT DISTINCT** employees.department, departments.department **FROM** employees **INNER JOIN** departments **ON** employees.department = departments.department *-- 23 departments, this is because An inner join searches tables for matching or overlapping data.*
- **SELECT DISTINCT** employees.department, departments.department **FROM** employees **LEFT JOIN** departments **ON** employees.department = departments.department *-- 27 departments, it takes the employees table*

- **SELECT DISTINCT** employees.department, departments.department **FROM** employees **RIGHT JOIN** departments **ON** employees.department = departments.department *-- 24 departments, it takes the department table*
- **SELECT DISTINCT** employees.department, departments.department **FROM** employees **FULL OUTER JOIN** departments **ON** employees.department = departments.department *-- 28 departments. It takes both table information.*
- **SELECT DISTINCT** employees.department, departments.department **FROM** employees **LEFT JOIN** departments **ON** employees.department = departments.department WHERE departments.department **IS NULL** *-- This is basically A-B. So it prints 4 different departments from employees table that doesn't exist in the departments table.*

UNION, UNION ALL and EXCEPT Clauses



- **SELECT** department **FROM** employees **UNION SELECT** department **FROM** departments *-- Union is going to stack both tables, but eliminate duplicates. 27 + 1 = 28 records.*
- **SELECT** department **FROM** employees **UNION ALL SELECT** department **FROM** departments *-- Union is going to stack both tables, but it will not eliminate duplicates. 1000 + 24 = 1024 records.*
- **SELECT DISTINCT** department **FROM** employees **UNION ALL SELECT** department **FROM** departments *-- Union is going to stack both tables, but it will not eliminate duplicates. 27 + 24 = 51 records.*
- **SELECT DISTINCT** department, region_id FROM employees **UNION ALL SELECT** department **FROM** departments *-- Error: Number of columns and data types should match*
- **SELECT DISTINCT** department, first_name **FROM** employees **UNION ALL SELECT** department, division **FROM** departments *-- This will work but doesn't make sense, because it just looks for the # of columns and data types.*
- **SELECT DISTINCT** department **FROM** employees **UNION ALL SELECT** department **FROM** departments **ORDER BY** department *-- ORDER BY applies to all clause*
- **SELECT DISTINCT** department **FROM** employees **UNION ALL SELECT** department **FROM** departments **UNION SELECT** country FROM regions **ORDER BY** department
- **SELECT DISTINCT** department **FROM** employees **EXCEPT SELECT** department **FROM** departments *-- A-B. 4 records will show.*
- **SELECT** department **FROM** departments **EXCEPT SELECT DISTINCT** department **FROM** employees *-- A- B. 1 record.*
- *-- ORACLE database has MINUS instead of EXCEPT.*
- **SELECT** department, **COUNT**(*) **FROM** employees **GROUP BY** department **UNION ALL SELECT** 'TOTAL', **COUNT**(*) **FROM** employees

Cartesian product with CROSS JOIN

- **SELECT** * **FROM** employees, departments *-- 24000 records = 1000 from employees x 24 from department*
- **SELECT** * **FROM** employees e1, employees e2 *-- 1000000 records = 1000 from employees x 1000 from employees*
- **SELECT** * **FROM** employees e1, employees e2, departments *-- 24000000 records = 1000 from employees x 1000 from employees x 24 from department*
- **SELECT** * **FROM** employees e1 **CROSS JOIN** departments b *-- 24000 records = 1000 from employees x 24 from department*

EXERCISE JOIN

- (SELECT first_name, department, hire_date, country FROM employees e INNER JOIN regions r ON e.region_id=r.region_id
  WHERE hire_date = (SELECT MIN(hire_date) FROM employees e2) LIMIT 1)
  UNION
  SELECT first_name, department, hire_date, country FROM employees e INNER JOIN regions r ON e.region_id=r.region_id
  WHERE hire_date = (SELECT MAX(hire_date) FROM employees e2)
  ORDER BY hire_date
- SELECT hire_date, salary, (SELECT SUM(salary) FROM employees e2 WHERE e2.hire_date BETWEEN e1.hire_date - 90 AND e1.hire_date) AS spending_pattern FROM employees e1 ORDER BY hire_date

VIEWS vs INLINE VIEWS

- CREATE VIEW v_employee_information AS
  SELECT first_name, email, e.department, salary, division, region, country FROM employees e, departments d, regions r
  WHERE e.department = d.department
  AND e.region_id = r.region_id – - you can't delete or insert to views

Assignment-7

1. Are the tables **student_enrollment** and **professors** directly related to each other? Why or why not?

They are NOT related directly. The reason is, there is no common column shared amongst them. There cannot be a direct relationship formed between these 2 tables.

2. Write a query that shows the student's name, the courses the student is taking and the professors that teach that course.

```
SELECT student_name, se.course_no, p.last_name
FROM students s
INNER JOIN student_enrollment se
   ON s.student_no = se.student_no
INNER JOIN teach t
   ON se.course_no = t.course_no
INNER JOIN professors p
   ON t.last_name = p.last_name
ORDER BY student_name;
```

3. If you execute the query from the previous answer, you'll notice the **student_name** and the **course_no** is being repeated. Why is this happening?

The combination of *student_name* and *course_no* is being repeated for as many professors that are teaching that particular course. If you ORDER BY the *student_name* column, you'll clearly be able to see that multiple professors are teaching the same subject. For example, course CS110 is being taught by both Brown and Wilson. That is why you'll see the combination of the student Arnold with CS110 twice. Analyze the data and understand what's going on because in the next question you'll need to write a query to be eliminate this redundancy.

4. In question 3 you discovered why there is repeating data. How can we eliminate this redundancy? Let's say we only care to see a single professor teaching a course and we don't care for all the other professors that teach the particular course. Write a query that will accomplish this so that every record is distinct.

HINT: Using the DISTINCT keyword will not help. :-)

```
SELECT student_name, course_no, min(last_name)
FROM (
SELECT student_name, se.course_no, p.last_name
FROM students s
INNER JOIN student_enrollment se
   ON s.student_no = se.student_no
INNER JOIN teach t
   ON se.course_no = t.course_no
INNER JOIN professors p
   ON t.last_name = p.last_name
   ) a
   GROUP BY student_name, course_no
   ORDER BY student_name, course_no;
```

5. Why are correlated subqueries slower that non-correlated subqueries and joins?

A "correlated subquery" (i.e., one in which the where condition depends on values obtained from the rows of the containing/outer query) will execute once for each row. A non-correlated subquery (one in which the where condition is independent of the containing query) will execute once at the beginning. If a subquery needs to run for each row of the outer query, that's going be very slow!

6. In the video lectures, we've been discussing the **employees** table and the **departments** table. Considering those tables, write a query that returns employees whose salary is above average for their given department.

```
SELECT first_name
  FROM employees outer_emp
  WHERE salary > (
   SELECT AVG(salary)
     FROM employees
     WHERE department = outer_emp.department);
```

7. Write a query that returns ALL of the students as well as any courses they may or may not be taking.

```
SELECT s.student_no, student_name, course_no
FROM students s LEFT JOIN student_enrollment se
   ON s.student_no = se.student_no
```

Section 9: Window Functions for Analytics

Window Functions using OVER() Clause

- SELECT first_name, department, (SELECT COUNT(*) FROM employees e1 WHERE e1.department = e2.department) FROM employees e2 ORDER BY department *-- this sub query is going to run for each row. So it slows.*

- *-- we can use the window function*
- SELECT first_name, department, COUNT(*) OVER(PARTITION BY department) FROM employees
- SELECT first_name, department, SUM(salary) OVER(PARTITION BY department) FROM employees
- SELECT first_name, department, COUNT(*) OVER(PARTITION BY department) dept_count, region_id, COUNT(*) OVER(PARTITION BY region_id) region_count FROM employees
- SELECT first_name, department, COUNT(*) OVER(PARTITION BY department) FROM employees WHERE region_id = 3

- *-- More complicated window function*
- SELECT first_name, hire_date, salary, SUM(salary) OVER(ORDER BY hire_date RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) as running_total_of_salaries FROM employees
- SELECT first_name, hire_date, salary, SUM(salary) OVER(ORDER BY hire_date) as running_total_of_salaries FROM employees
- SELECT first_name, hire_date, department, salary, SUM(salary) OVER(PARTITION BY department ORDER BY hire_date) as running_total_of_salaries FROM employees
- SELECT first_name, hire_date, department, salary, SUM(salary) OVER(ORDER BY hire_date ROWS BETWEEN 1 PRECEDING AND CURRENT ROW) as running_total_of_salaries FROM employees
- SELECT first_name, hire_date, department, salary, SUM(salary) OVER(ORDER BY hire_date ROWS BETWEEN 3 PRECEDING AND CURRENT ROW) as running_total_of_salaries FROM employees

RANK, FIRST_VALUE and NTILE Functions

- SELECT first_name, email, department, salary, RANK() OVER(PARTITION BY department ORDER BY salary DESC) FROM employees
- SELECT first_name, email, department, salary, RANK() OVER(PARTITION BY department ORDER BY salary DESC) FROM employees WHERE RANK = 8 *-- ERROR. Because rank doesn't exist.*
- SELECT * FROM (SELECT first_name, email, department, salary, RANK() OVER(PARTITION BY department ORDER BY salary DESC) FROM employees) a WHERE RANK = 8
- SELECT first_name, email, department, salary, NTILE(5) OVER(PARTITION BY department ORDER BY salary DESC) FROM employees *-- how many buckets? so it will divide employees into 5 groups by their departments.*
- SELECT first_name, email, department, salary, first_value(salary) OVER(PARTITION BY department ORDER BY salary DESC) first_value FROM employees -- First value of each department.

- **SELECT** first_name, email, department, salary, **MAX**(salary) **OVER**(**PARTITION BY** department **ORDER BY** salary **DESC**) first_value FROM employees *-- Same as the above query*
- **SELECT** first_name, email, department, salary, **nth_value**(salary, 5) **OVER**(**PARTITION BY** department **ORDER BY** first_name **ASC**) nth_value **FROM** employees *-- 5th value of each department.*

LEAD, LAG Functions

- **SELECT** first_name, last_name, salary, **LEAD**(salary) **OVER**() next_salary **FROM** employees *-- pull the salary from the next row*
- **SELECT** first_name, last_name, salary, **LAG**(salary) **OVER**() next_salary **FROM** employees *-- pull the salary from the previous row*
- **SELECT** department, last_name, salary, **LAG**(salary) **OVER**(**ORDER BY** salary **DESC**) closest_higher_salary **FROM** employees *-- pull the next higher paid salary*
- **SELECT** department, last_name, salary, **LEAD**(salary) **OVER**(**ORDER BY** salary **DESC**) closest_lower_salary **FROM** employees *-- pull the next lower paid salary*
- **SELECT** department, last_name, salary, **LEAD**(salary) **OVER**(**PARTITION BY** department **ORDER BY** salary **DESC**) closest_lower_salary **FROM** employees *-- pull the next lower paid salary*

Working with ROLLUPs and CUBES

- **SELECT** * **FROM** sales **ORDER BY** continent, country, city
-
- **SELECT** continent, **SUM**(units_sold) **FROM** sales **GROUP BY** continent
- **SELECT** country, **SUM**(units_sold) **FROM** sales **GROUP BY** country
- **SELECT** city, **SUM**(units_sold) **FROM** sales **GROUP BY** city
-
- **SELECT** continent, country, city, **SUM**(units_sold) **FROM** sales **GROUP BY GROUPING SETS**(continent, country, city) *-- this combines all of above 3 group by statements.*
-
- **SELECT** continent, country, city, **SUM**(units_sold) **FROM** sales **GROUP BY GROUPING SETS**(continent, country, city, ()) *-- () syntax provides the total, similar to OVER()*
-
- **SELECT** continent, country, city, **SUM**(units_sold) **FROM** sales **GROUP BY ROLLUP**(continent, country, city) *-- it groups by continent+country+city, continent+country, continent*
- **SELECT** continent, country, city, **SUM**(units_sold) **FROM** sales **GROUP BY CUBE**(continent, country, city) *-- it groups by all combinations, total : 2^3 combinations.*

SECTION 10: Assignment

1. Write a query that finds students who do not take CS180.

   You may have thought about the following query at first, but this is not correct:

   ```
   SELECT * FROM students
   WHERE student_no IN (SELECT student_no
                        FROM student_enrollment
                        WHERE course_no != 'CS180')
                       ORDER BY student_name
   ```

   The above query is incorrect because it does not answer the question "Who does not take CS180?". Instead, it answers the question "Who takes a course that is not CS180?" The correct result should include students who take no courses as well as students who take courses but none of them CS180.

   2 CORRECT ANSWERS BELOW:

   Answer A:

   ```
   SELECT * FROM students
   WHERE student_no NOT IN (
      SELECT student_no
      FROM student_enrollment
      WHERE course_no = 'CS180'
      );
   ```

Answer B: Bonus points if you can understand the below solution.

```
SELECT s.student_no, s.student_name, s.age
FROM students s LEFT JOIN student_enrollment se
   ON s.student_no = se.student_no
GROUP BY s.student_no, s.student_name, s.age
HAVING MAX(CASE WHEN se.course_no = 'CS180'
        THEN 1 ELSE 0 END) = 0
```

2. Write a query to find students who take CS110 or CS107 but not both.

   The following query looks promising as a solution but returns the wrong result!

```
SELECT *
FROM students
WHERE student_no IN (SELECT student_no
                FROM student_enrollment
                WHERE course_no != 'CS110'
                AND course_no != 'CS107')
```

2 CORRECT ANSWERS BELOW:

Solution A:

```
SELECT s.*
FROM students s, student_enrollment se
WHERE s.student_no = se.student_no
AND se.course_no IN ('CS110', 'CS107')
AND s.student_no NOT IN ( SELECT a.student_no
                FROM student_enrollment a, student_enrollment b
                WHERE a.student_no = b.student_no
                AND a.course_no = 'CS110'
                AND b.course_no = 'CS107')
```

Solution A uses a self join on the student_enrollment table so that those students are narrowed down that take both CS110 and CS107 in the subquery. The outer query filters for those student_no that are not the ones retrieved from the subquery.

Solution B:

```
SELECT s.student_no, s.student_name, s.age
FROM students s, student_enrollment se
WHERE s.student_no = se.student_no
GROUP BY s.student_no, s.student_name, s.age
HAVING SUM(CASE WHEN se.course_no IN ('CS110', 'CS107')
        THEN 1 ELSE 0 END ) = 1
```

In solution B, a CASE expression is used with the aggregate SUM function to find students who take either CS110 or CS107, but not both.

3. Write a query to find students who take CS220 and no other courses.

   You may have thought about the below query to solve this problem but this will not give you the correct result:

```
SELECT s.*
FROM students s, student_enrollment se
WHERE s.student_no = se.student_no
AND se.course_no = 'CS220'
```

We want to see those students who only take CS220 and no other course. The above query returns students who take CS220 but these students could also be taking other courses and that is why this query doesn't work.

2 CORRECT ANSWERS BELOW:

Solution A:

```
SELECT s.*
FROM students s, student_enrollment se
WHERE s.student_no = se.student_no
AND s.student_no NOT IN ( SELECT student_no
                FROM student_enrollment
                WHERE course_no != 'CS220')
```

In Solution A, the subquery returns all students that take a course other than CS220. The outer query gets all students regardless of what course they take. In essence, the subquery finds all students who take a course that is not CS220. The outer query returns all student who are not

amongst those that take a course other than CS220. At this point, the only available students are those who actually take CS220 or take nothing at all.

Solution B:

```
SELECT s.*
FROM students s, student_enrollment se1,
    (SELECT student_no FROM student_enrollment
     GROUP BY student_no
     HAVING count(*) = 1) se2
WHERE s.student_no = se1.student_no
AND se1.student_no = se2.student_no
AND se1.course_no = 'CS220'
```

Solution B uses subquery to get those students who take only a single course and since it's in the from clause, it's considered a source of data just like a table. This is also called an inline view if you recall. So the student_no from the inline view is joined with the outer query and we filter for only those students that take the course CS220. So this query returns that one student that takes CS220 and no other course.

4. Write a query that finds those students who take at most 2 courses. Your query should exclude students that don't take any courses as well as those that take more than 2 course.

   SOLUTION:

   ```
   SELECT s.student_no, s.student_name, s.age
   FROM students s, student_enrollment se
   WHERE s.student_no = se.student_no
   GROUP BY s.student_no, s.student_name, s.age
   HAVING COUNT(*) <= 2
   ```

   Use the COUNT function to determine which students take no more than 2 courses. Students that don't take any courses are being excluded anyway because of the join.

5. Write a query to find students who are older than at most two other students.

   SOLUTION:

   ```
   SELECT s1.*
   FROM students s1
   WHERE 2 >= (SELECT count(*)
               FROM students s2
               WHERE s2.age < s1.age)
   ```

   Using the aggregate function COUNT and a correlated subquery as shown in the solution above, you can retrieve the students who are older than zero, one or two other students.