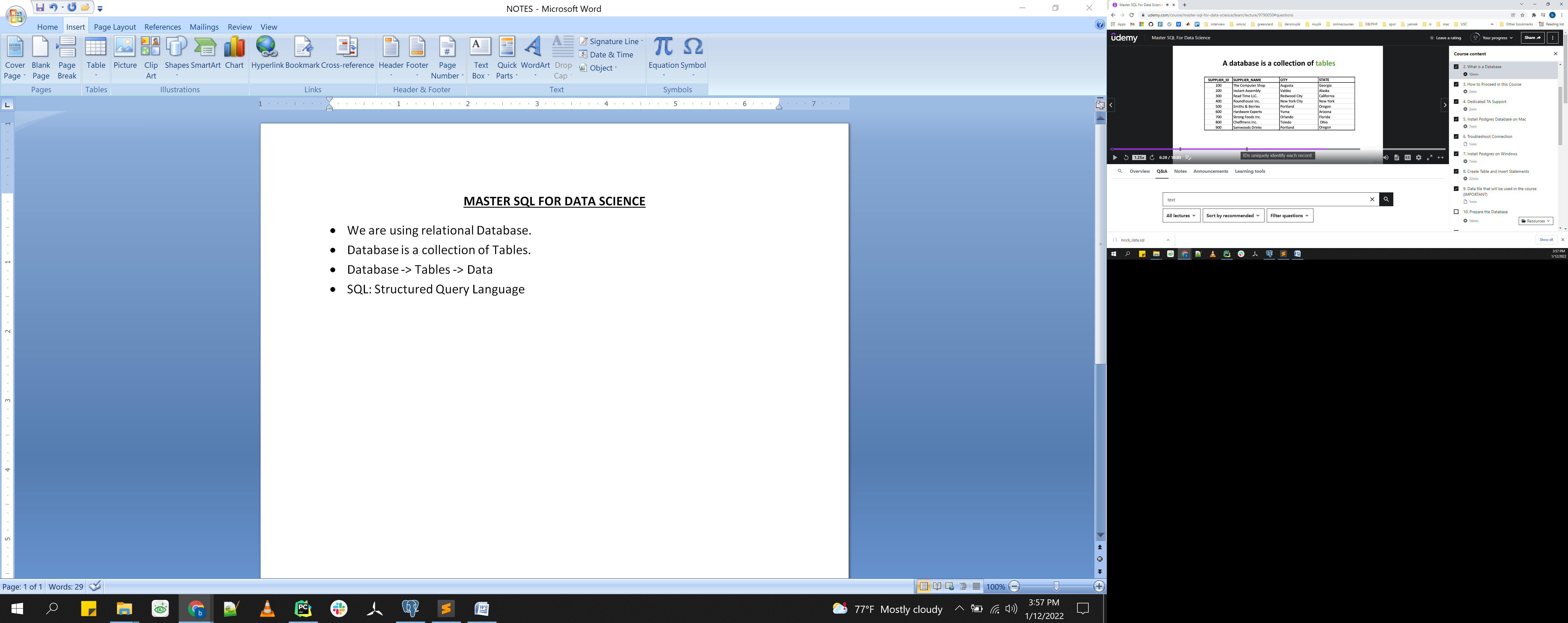
**MASTER SQL FOR DATA SCIENCE**

**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…**

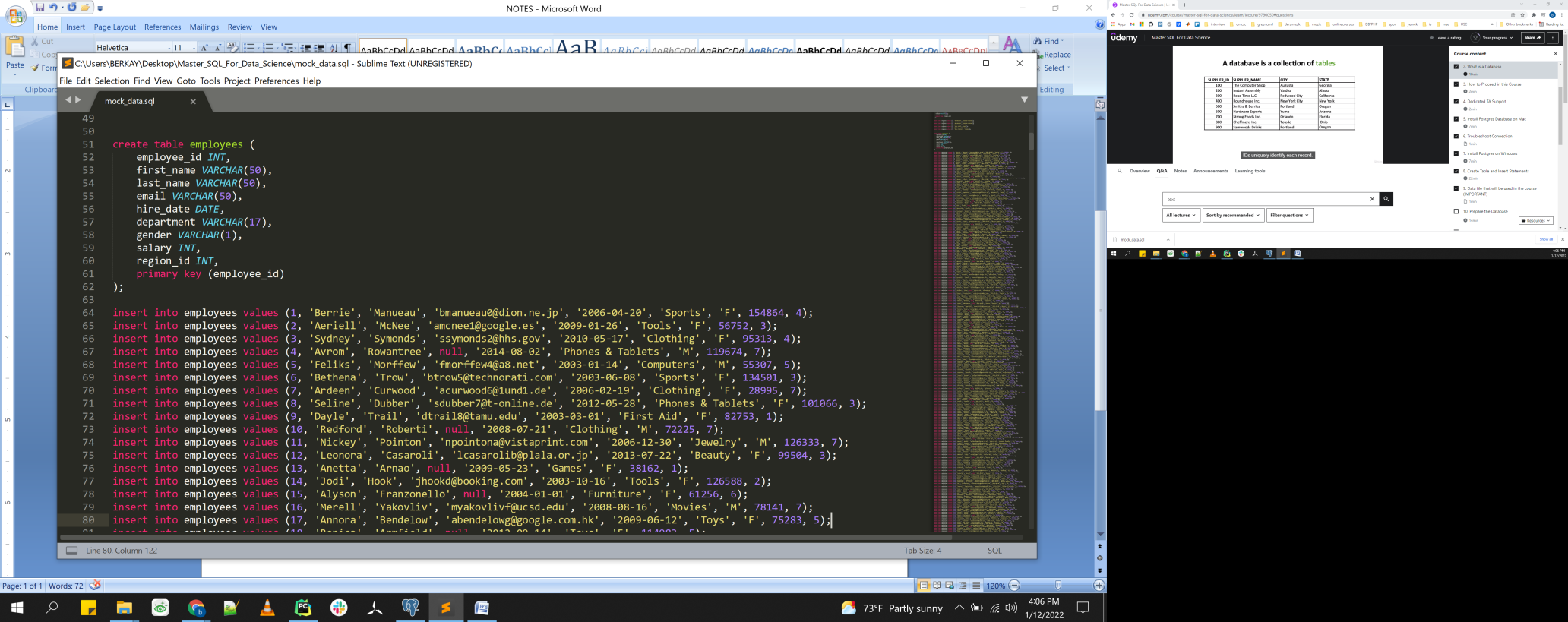
**CREATE TABLE table\_name (**

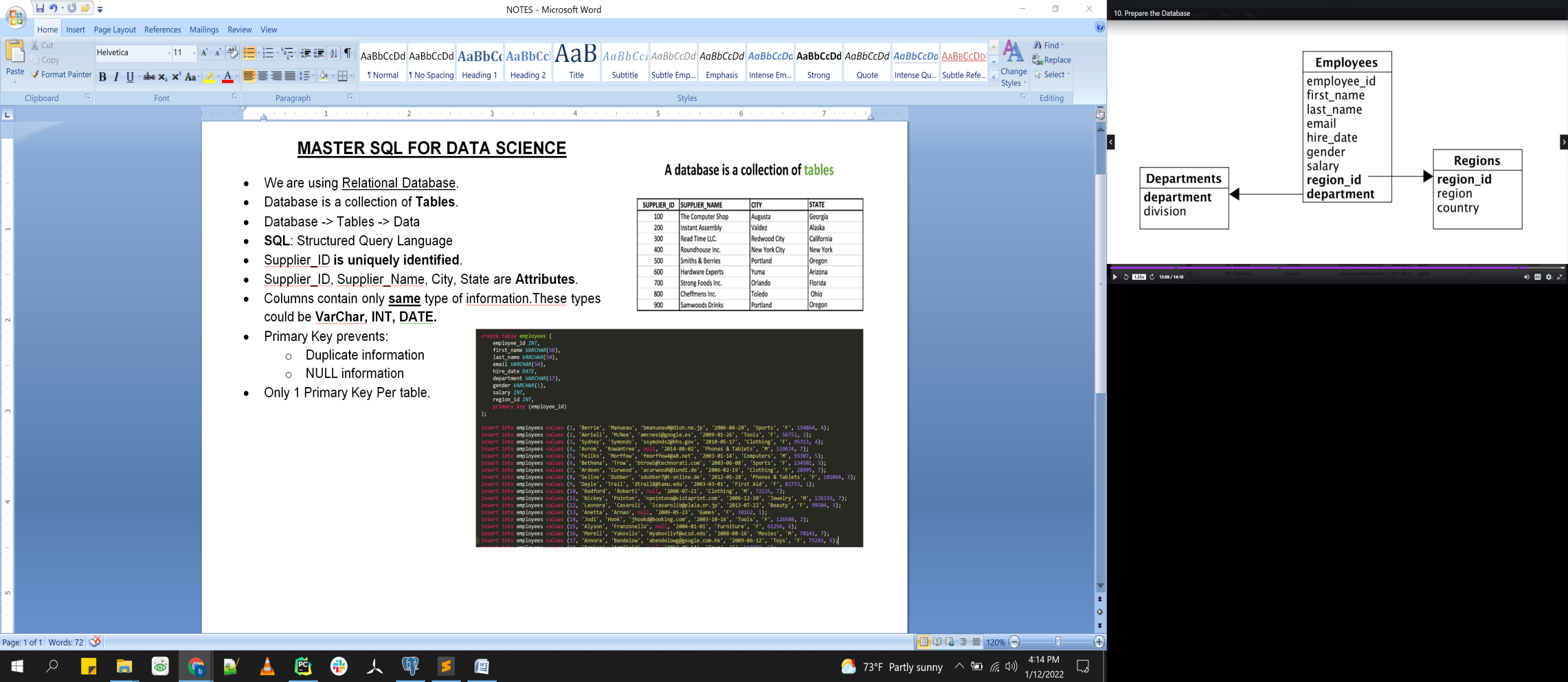
**column1data\_type,**

**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.



**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;**

1. 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';**

1. 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;**

1. 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;**

1. 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;**

1. 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 > 20 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**

1. 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**

1. 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**

1. 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'**

1. 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**

1. 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**

1. 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**

1. 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**

1. 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**

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

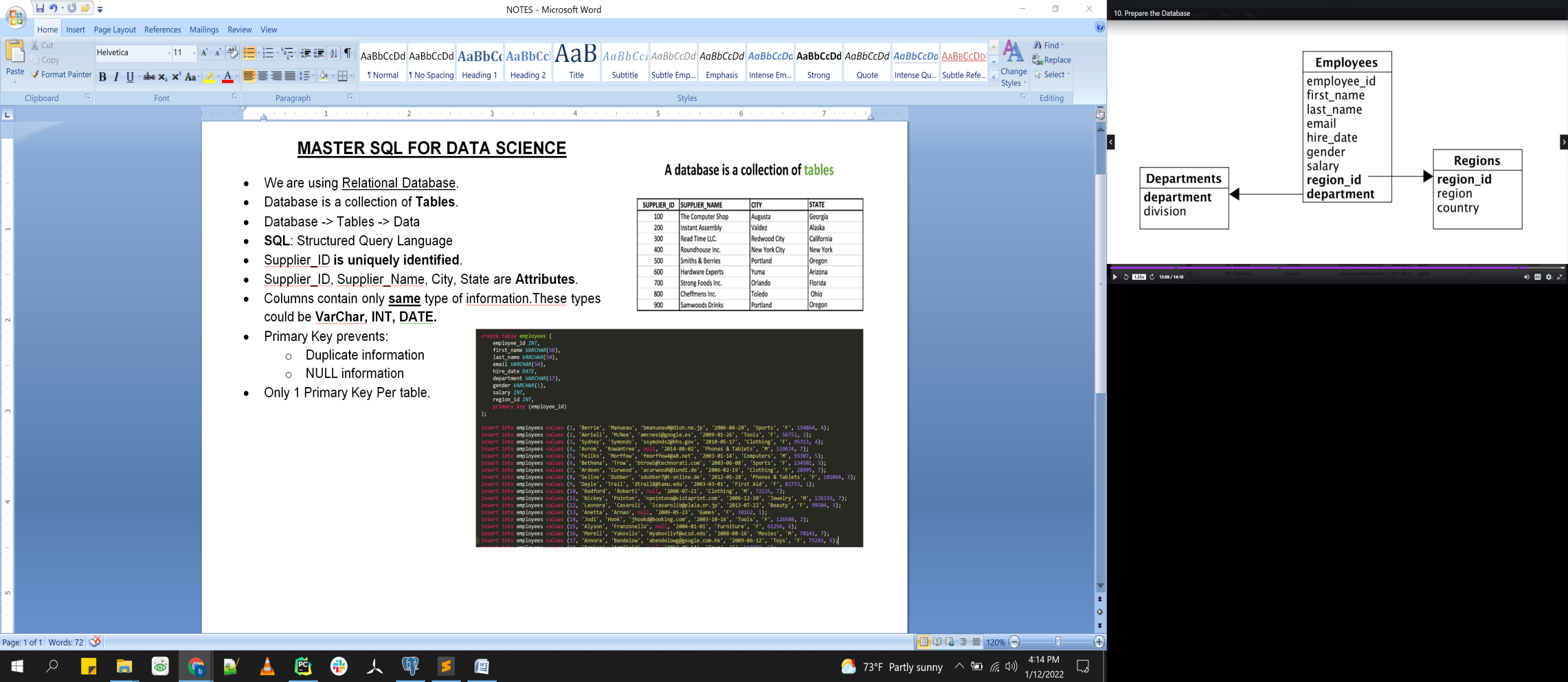
CREATE table fruits (fruit\_name varchar(10));  
INSERT INTO fruits VALUES ('Orange');  
INSERT INTO fruits VALUES ('Apple');  
INSERT INTO fruits VALUES (NULL);  
INSERT INTO fruits VALUES (NULL);

**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.**

1. 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')));**

1. 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 )**

1. **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.**

1. 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**

1. 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**

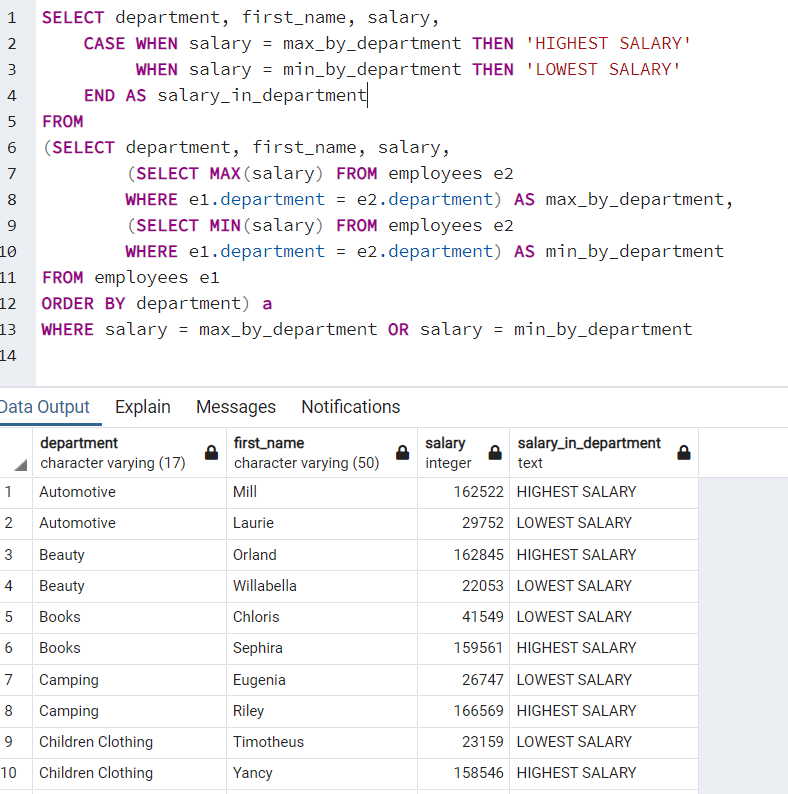
**CORRALETED SUBQUERIES**

* **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.***



**Section 8: Working with Multiple Tables**

|  |  |
| --- | --- |
|  |  |
|  |  |



**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

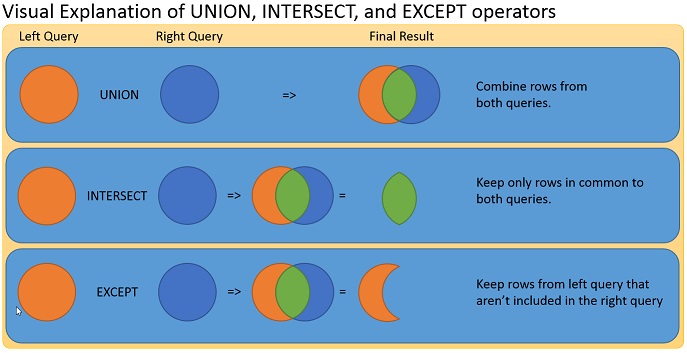
**WHERE** email **IS** **NOT** NULL

**------------------------------------------------------------------------------------------------------**

* **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.**

1. 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;**

1. 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.**

1. 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;**

1. 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!**

1. 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);**

1. 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**

1. 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.

1. 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.

1. 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.

1. 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.