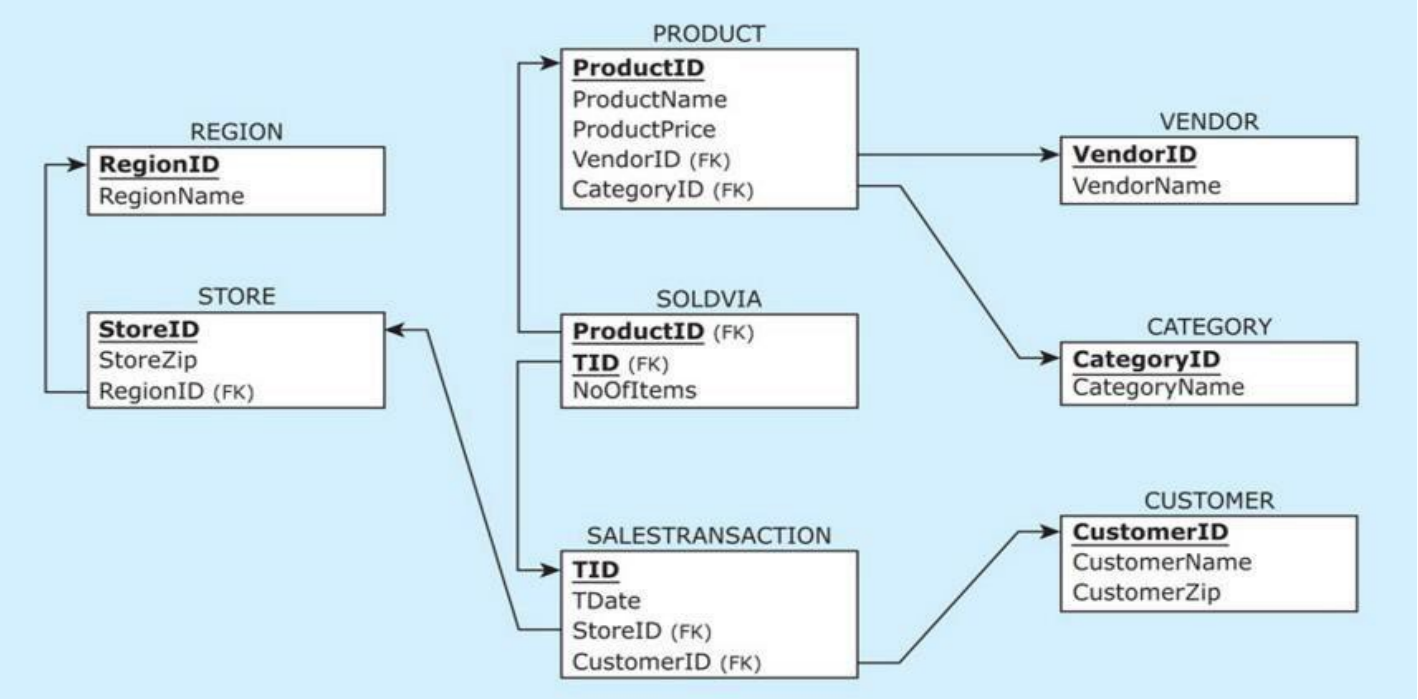
Lab 4 SQL DDL

The aim of this lab is to get hands-on experience on the followings:

1. Using scripts to create, populate and drop tables.
2. Writing SQL queries to retrieve data from a database

For this lab, we will require a database of a ZAGI retail company sales department.
The relational schema is as follows:



Steps to follows:

1. Download from D2L the *ZAGI DB Insert Script* file. It includes SQL script files needed to insert data into tables.

2. Create a Database by the name of zagiDB. Create Table from the Relational Schema above in the database. Assume, all the columns of each Entity are mandatory. Also decide on the datatype based on the data in the INSERT script file. **[10 Marks]**

3. After creating table, Insert data into the tables. **[10 Marks]**

4. Respect SQL coding style:
    1) all SQL keywords in CAPITAL
    2) name of tables and columns in PascalCase or Pascal_Case
    3) write each SELECT, FROM, WHERE, GROUP BY, ORDER BY… clause in a separate line

Submission

Write all the queries along with their sample output screenshot (like in slides) in a word file. This includes the Create Table queries.

For inserting data, no need to submit anything. If any of the queries (20 queries) below has a screenshot, it means insert data was successful and you will get full marks for Insert data question.

Append your name and ID to the solution word file before submitting in D2L.

Deadline: 16 Oct 2022 at 11.59PM

Marking Guideline

- Each query is of 4 marks including 1 mark for the screenshot of the answer. [20 × 4 = 80]

Queries

Query 1. Retrieve the entire contents of the Product table (all columns and all rows of the table).

Query 2. Display the VendorID and VendorName for all vendors.

Query 3. Display the CustomerName and CustomerZip for all customers.

Query 4. Retrieve the entire contents of the table PRODUCT. The columns must be displayed in the following order: ProductName, ProductID, VendorID, CategoryID, ProductPrice.

Query 5. For the table PRODUCT, display 3 columns ProductID, ProductPrice, and a column showing ProductPrice increased by 40% (Multiply ProductPrice by 1.40).

Query 6. Display the ProductID, ProductName, and ProductPrice for products with a ProductPrice of $100 or higher.

Query 7. Retrieve the ProductID, ProductName, VendorID, CategoryID, and ProductPrice of products in the FW category whose price is equal to or below $200 (Hint: two conditions in WHERE clause)

Query 8. Display the VendorID of all vendors that we have a product from them. In the result, we must not see duplicate vendorIDs. For example, if we have 5 products from 1 vendor, we must see the vendorID of that vendor only one time not 5 times.

Query 9. Retrieve the average price of all products. (Hint: Use AVG)

Query 10. Show how many products are there for sale. (Hint: COUNT)

Query 11. Count how many distinct vendors are there in the product table. (Hint: the answer is 4)

Query 12. Retrieve the number of products, average product price, lowest product price, and highest product price in the CP product category.

Query 13. Retrieve the product ID, product name, category ID, and product price for each product in the FW product category, sorted by product price in descending order.

Query 14. For each product, retrieve the ProductID, and the total number of product items sold within all transactions. (Hint: transactions can be found in SoldVia table. NoOfItems attribute holds how many items sold in one transaction, but we need to find the total sold of a product in all transactions. See the table below for your reference).

We need the heading of the columns be exactly 'PRODUCTID' and 'Total Sold' as you can see below.

PRODUCTID	Total Sold
4X2	4
1X1	2
4X4	7
6X6	2
1X3	3
1X2	8
3X1	6
5X5	2
2X1	6
4X3	4
More than 10 rows available. Increase rows selector to view more rows.	

Query 15. For each vendor, retrieve the VendorID, number of products supplied by the vendor, and average price of the products supplied by the vendor.

Query 16. Retrieve all attributes of products whose name starts with "Tiny", for example, 'Tiny Tent'

Query 17. Display the ProductID, ProductName, and ProductPrice for products in the category whose CategoryID value is 'CP'. Sort the results by ProductID.

Query 18. Display the transaction id (TID) and the total number of items sold in that transaction (of all products) that the total number of items (of all products) sold in that transaction is greater than five. In other words, we want to get the sample result as follows. We need the heading of the columns be exactly 'TID' and 'Total Items Sold' as you can see below.

TID	Total Items Sold
T333	6
T505	8
T888	7
T606	18
T022	8
T999	10
T303	9
T808	8
T555	7
T707	7
10 rows returned in 0.00 seconds Download	

Query 19. Display all RegionIDs and number of stores in their region

Query 20. Display RegionID and number of Stores in regions that number of stores in their region is 4 or more

QUICK RECAP AND SOME HINTS

1. Tables that do not have foreign keys are populated first. Tables that do have foreign keys cannot have their foreign key columns populated before the values of the primary key columns to which they are referring are populated. Referential integrity constraints require that primary key values that are referred to by a foreign key are entered before the foreign key values can be entered.

2. To retrieve data from database, SELECT statement is used. SELECT statement is structured as follows:

SELECT <columns, expressions>
FROM <tables>
WHERE <row selection condition>
GROUP BY <grouping columns>
HAVING <group selection condition>
ORDER BY <sorting columns, expressions>

WHERE
determines which rows should be retrieved and consequently which rows should not be retrieved. Following logical operators can be used:

= Equal to
< Less than
> Greater than
<= Less than or equal to
>= Greater than or equal to
!= Not equal to
<> Not equal to (alternative notation)

Within one WHERE clause, a multiple comparison expression can be used, connected by the Boolean logical operators AND or OR.

GROUP BY

- Enables summarizations across the groups of related data within tables.
- The guideline for using the GROUP BY clause is that the same column (or columns) that is (are) listed after the GROUP BY clause should also be listed after the SELECT clause.

HAVING

- Enables summarizations across the groups of related data within tables.
- Determines which groups will be displayed in the result of a query and, consequently, which groups will not be displayed in the result of the query
- A query that contains a HAVING clause must also contain a GROUP BY clause

ORDER BY

- To sort the results of the query by one or more columns, ORDER BY clause within the SELECT query is used.
- By default, ORDER BY sorts the data in ascending order. If we want to sort in descending order, we can use the keyword DESC.

3. The meaning of the * symbol after the SELECT keyword is “all columns.”

4. For calculating and summarizing values in queries, SQL provides the following aggregate functions:
   COUNT
   SUM
   AVG
   MIN
   MAX

   The COUNT function counts the number of rows in the result of the query, while the SUM, AVG, MIN, and MAX functions calculate sum, average, minimum, and maximum values in a column, respectively. Functions SUM and AVG operate on numeric values only. Functions MIN and MAX operate not only on numeric values, but on date and character values as well. All aggerate functions, except COUNT ignore NULL values.