UNIT IV

Introduction to SQL

Background:

SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.

SQL is not a database system, but it is a query language.

SQL is a short-form of the structured query language, and it is pronounced as S-Q-L or sometimes as See-Quell.

Why SQL?

Nowadays, SQL is widely used in data science and analytics. Following are the reasons which explain why it is widely used:

- The basic use of SQL for data professionals and SQL users is to insert, update, and delete the data from the relational database.
- SQL allows the data professionals and users to retrieve the data from the relational database management systems.
- It also helps them to describe the structured data.
- It allows SQL users to create, drop, and manipulate the database and its tables.
- It also helps in creating the view, stored procedure, and functions in the relational database.
- It allows you to define the data and modify that stored data in the relational database.
- It also allows SQL users to set the permissions or constraints on table columns, views, and stored procedures.

History of SQL

"A Relational Model of Data for Large Shared Data Banks" was a paper which was published by the great computer scientist "E.F. Codd" in 1970.

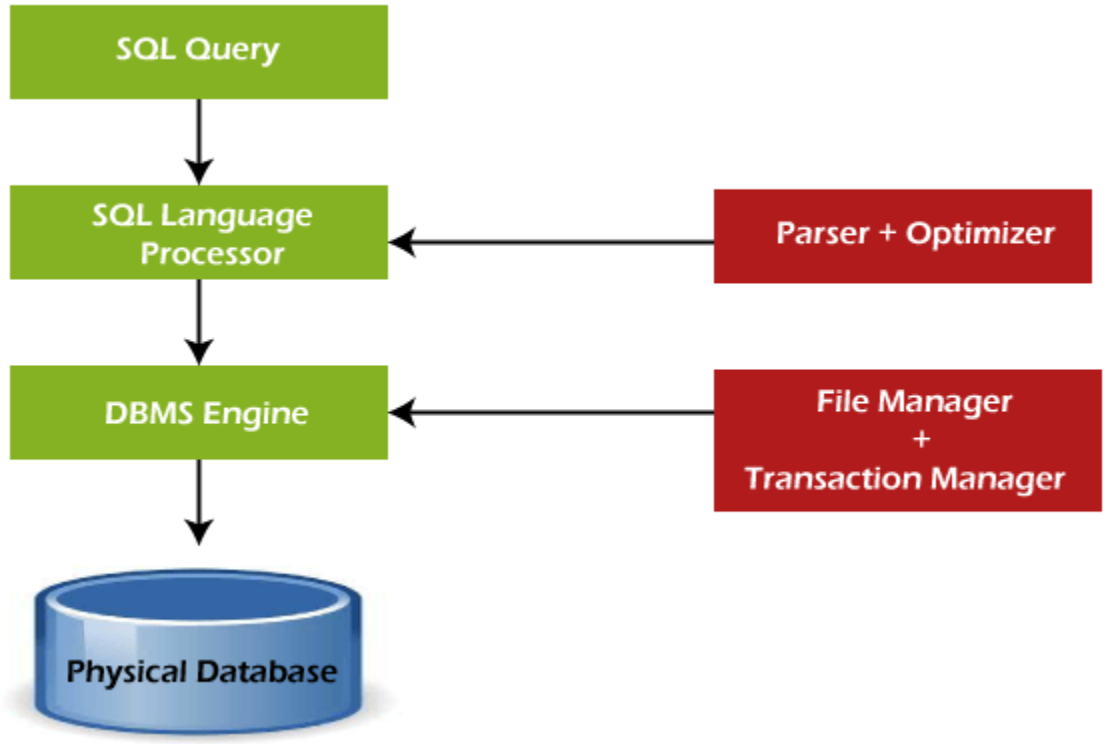
Process of SQL

When we are executing the command of SQL on any Relational database management system, then the system automatically finds the best routine to carry out our request, and the SQL engine determines how to interpret that particular command.

Structured Query Language contains the following four components in its process:

- Query Dispatcher
- Optimization Engines
- Classic Query Engine
- SQL Query Engine, etc.

A classic query engine allows data professionals and users to maintain non-SQL queries. The architecture of SQL is shown in the following diagram:



Types of SQL Commands (DDL, DML, DCL, DQL, TCL)

Commands In SQL:

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language(DCL)
- Transaction Control Language(TCL)
- Data Query Language (DQL)

Data Definition Language (DDL):

Data Definition Language helps you to define the database structure or schema. Let's learn about DDL commands with syntax.

The following commands are used in DDL:

- **CREATE:** Used in the creation of a database.
- **DROP:** Used in the deletion of objects from the database.
- **ALTER**: Used in the alteration of the structure of the database.
- **TRUNCATE:** Used for the deletion of all the records from the table.
- **RENAME**: Used in the renaming of an existing object in the database.

CREATE

CREATE statements is used to define the database structure schema:

Syntax:

```
CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[size]);
```

For example:

```
Create database university;
Create table students;
Create view for_students;
```

SQL CREATE TABLE

SQL CREATE TABLE statement is used to create table in a database.

If you want to create a table, you should name the table and define its column and each column's data type.

Let's see the simple syntax to create the table.

```
create table "tablename"
("column1" "data type",
"column2" "data type",
"column3" "data type",
...
"columnN" "data type");
```

For ex:

```
CREATE TABLE Employee
(
EmployeeID int,
FirstName varchar(20),
LastName varchar(20),
Email varchar(20),
AddressLine varchar(20),
City varchar(20)
);
```

SQL ALTER TABLE

The ALTER TABLE statement in Structured Query Language allows you to add, modify, and delete columns of an existing table.

This statement also allows database users to add and remove various SQL constraints on the existing tables.

ALTER TABLE ADD Column statement in SQL

In many situations, you may require to add the columns in the existing table. Instead of creating a whole table or database again you can easily add single and multiple columns using the ADD keyword.

Syntax of ALTER TABLE ADD Column statement in SQL

1. ALTER TABLE table_name ADD column_name column-definition;

2. ALTER TABLE table_name
3. ADD (column_Name1 column-definition,
4. column_Name2 column-definition,
5. .....
6. column_NameN column-definition);

Table: Employee

- Suppose, you want to add two columns, **Emp_ContactNo.** and **Emp_EmailID,** in the above Employee table. For this, you have to type the following query in the SQL:
- **For Ex:**

1. ALTER TABLE Employee ADD (Emp_ContactNo. Number(13), Emp_EmailID varchar(50) ;

This statement will add Emp_ContactNo. and Emp_EmailID columns to the Employee table.

ALTER TABLE MODIFY Column statement in SQL

The MODIFY keyword is used for changing the column definition of the existing table.

Syntax of ALTER TABLE MODIFY Column statement in SQL

1. ALTER TABLE table_name MODIFY column_name column-definition;

This syntax only allows you to modify a single column of the existing table. If you want to modify more than one column of the table in a single SQL statement, then use the following syntax:

1. ALTER TABLE table_name
2. MODIFY (column_Name1 column-definition,
3. column_Name2 column-definition,
4. .....

5. column_NameN column-definition);

For ex:

Table: Employee

- Suppose, you want to modify the datatypes of two columns **Emp_ContactNo.** and **Emp_EmailID** of the above Employee table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Employee ADD (Emp_ContactNo. Int, Emp_EmailID varchar(80) ;

ALTER TABLE DROP Column statement in SQL

In many situations, you may require to delete the columns from the existing table.

Instead of deleting the whole table or database you can use DROP keyword for deleting the columns.

Syntax :

1. ALTER TABLE table_name DROP Column column_name ;

For Ex:

Table: Employee

- Suppose, you want to delete the **Emp_Salary and Emp_City** column from the above Employee table. For this, you have to type the following two different queries in the SQL:

1. ALTER TABLE Cars DROP COLUMN Emp_Salary ;
2. ALTER TABLE Cars DROP COLUMN Emp_City ;

ALTER TABLE RENAME Column statement in SQL

The RENAME keyword is used for changing the name of columns or fields of the existing table.

Syntax:

1. ALTER TABLE table_name RENAME COLUMN old_name to new_name;

Table: Employee

- Suppose, you want to change the name of **the Emp_City** column of the above Employee table. For this, you have to type the following query in the SQL:

1. ALTER TABLE Employee RENAME COLUMN Emp_City to Emp_Address;

SQL TRUNCATE TABLE

A truncate SQL statement is used to remove all rows (complete data) from a table. It is similar to the DELETE statement with no WHERE clause.

TRUNCATE TABLE Vs DELETE TABLE

Truncate table is faster and uses lesser resources than DELETE TABLE command.

TRUNCATE TABLE Vs DROP TABLE

Drop table command can also be used to delete complete table but it deletes table structure too. TRUNCATE TABLE doesn't delete the structure of the table.

1. **TRUNCATE TABLE** table_name;

For example, you can write following command to truncate the data of employee table

SQL DROP TABLE

A SQL DROP TABLE statement is used to delete a table definition and all data from a table.

This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.

Syntax:

1. **DROP TABLE** "table_name";

For ex:

This shows that STUDENTS table is available in the database, so we can drop it as follows:

1. SQL>**DROP TABLE** STUDENTS;

SQL RENAME TABLE

In some situations, database administrators and users want to change the name of the table in the SQL database because they want to give a more relevant name to the table.

Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

The RENAME TABLE and ALTER TABLE syntax help in changing the name of the table.

Syntax of RENAME statement in SQL

1. RENAME old_table _name To new_table_name ;

For ex:

Table: Employee

- Suppose, you want to change the name of the above table into the **"Coding_Employees".** For this, you have to type the following RENAME statement in SQL:

1. RENAME Employee To Coding_Employees ;

What is Data Manipulation Language?

Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data.
It is responsible for performing all types of data modification in a database.
There are three basic constructs which allow database program and user to enter data and information are:

Here are some important DML commands in SQL:

- INSERT
- UPDATE
- DELETE

INSERT:

This is a statement is a SQL query. This command is used to insert data into the row of a table.

SQL INSERT statement is a SQL query. It is used to insert a single or a multiple records in a table.

There are two ways to insert data in a table:

1. By SQL insert into statement
    1. By specifying column names
    2. Without specifying column names
2. By SQL insert into select statement

1) Inserting data directly into a table

You can insert a row in the table by using SQL INSERT INTO command.

There are two ways to insert values in a table.

In the first method there is no need to specify the column name where the data will be inserted, you need only their values.

```
INSERT INTO table_name
VALUES (value1, value2, value3....);
```

The second method specifies both the column name and values which you want to insert.

```
INSERT INTO table_name (column1, column2, column3....)
VALUES (value1, value2, value3.....);
```

Let's take an example of table which has five records within it.

```
INSERT INTO STUDENTS (ROLL_NO, NAME, AGE, CITY)
```

2. **VALUES** (1, ABHIRAM, 22, ALLAHABAD);
3. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
4. **VALUES** (2, ALKA, 20, GHAZIABAD);
5. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
6. **VALUES** (3, DISHA, 21, VARANASI);
7. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
8. **VALUES** (4, ESHA, 21, DELHI);
9. **INSERT INTO** STUDENTS (ROLL_NO, **NAME**, AGE, CITY)
10. **VALUES** (5, MANMEET, 23, JALANDHAR);

It will show the following table as the final result.

ROLL_NO	NAME	AGE	CITY
1	ABHIRAM	22	ALLAHABAD
2	ALKA	20	GHAZIABAD
3	DISHA	21	VARANASI
4	ESHA	21	DELHI
5	MANMEET	23	JALANDHAR

UPDATE:

SQL UPDATE statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.

The UPDATE statement can be written in following form:

1. **UPDATE** table_name **SET** [column_name1= value1,... column_nameN = valueN] [**WHERE** condition]

Let's see the Syntax:

1. **UPDATE** table_name
2. **SET** column_name = expression
3. **WHERE** conditions

Let's take an example: here we are going to update an entry in the source table.

SQL statement:

1. **UPDATE** students
2. **SET** User_Name = 'beinghuman'
3. **WHERE** Student_Id = '3'

4. **Source Table:**

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	jonny

5. See the result after updating value:

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	**beinghuman**

SQL DELETE TABLE

The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

1. **DELETE FROM** table_name [**WHERE** condition];

But if you do not specify the WHERE condition it will remove all the rows from the table.

1. **DELETE FROM** table_name;

SQL DELETE ROW

Let us take an example of student.

Original table:

ID	STUDENT _NAME	ADDRESS
001	AJEET MAURYA	GHAZIABAD
002	RAJA KHAN	LUCKNOW
003	RAVI MALIK	DELHI

If you want to delete a student with id 003 from the student_name table, then the SQL DELETE query should be like this:

1. **DELETE FROM** student_name
2. **WHERE** id = 003;

Resulting table after SQL DELETE query:

ID	STUDENT_NAME	ADDRESS
001	AJEET MAURYA	GHAZIABAD
002	RAJA KHAN	LUCKNOW

TCL Full Form

TCL stands for **Transaction Control Languages**. These commands are used for maintaining consistency of the database and for the management of transactions made by the DML commands.
A **Transaction** is a set of SQL statements that are executed on the data stored in DBMS. Whenever any transaction is made these transactions are temporarily happen in database.So to make the changes permanent, we use **TCL** commands.
The TCL commands are:

1. COMMIT
2. ROLLBACK

3. **1. COMMIT :**
   This command is used to save the data permanently.
   Whenever we perform any of the DDL command like -INSERT, DELETE or UPDATE, these can be rollback if the data is not stored permanently. So in order to be at the safer side COMMIT command is used.

Syntax:

```
commit;
```

4. **2. ROLLBACK :**
   This command is used to get the data or restore the data to the last savepoint or last committed state. If due to some reasons the data inserted, deleted or updated is not correct, you can rollback the data to a particular savepoint or if savepoint is not done, then to the last committed state.

Syntax:

```
rollback;
```

3. **SAVEPOINT :**
   This command is used to save the data at a particular point temporarily, so that whenever needed can be rollback to that particular point.

Syntax:

```
Savepoint A;
```

Consider the following Table Student:

Name	Marks
John	79
Jolly	65
Shuzan	70

```
UPDATE STUDENT
SET NAME = ‘Sherlock’
WHERE NAME = ‘Jolly’;


COMMIT;
ROLLBACK;
```

By using this command you can update the record and save it permanently by using **COMMIT** command.

Now after COMMIT :

Name	Marks
John	79
Sherlock	65
Shuzan	70

If commit was not performed then the changes made by the update command can be rollback.

Now if no COMMIT is performed.

```
UPDATE STUDENT
SET NAME = ‘Sherlock’
WHERE STUDENT_NAME = ‘Jolly’;
```

After update command the table will be:

Name	Marks
John	79
Sherlock	65
Shuzan	70

Now if ROLLBACK is performed on the above table:

```
rollback;
```

After Rollback:

Name	Marks
John	79
Jolly	65
Shuzan	70

What is DCL?

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give “rights & permissions.” Other permission controls parameters of the database system.

Examples of DCL commands:

Commands that come under DCL:

- Grant
- Revoke

Grant:

This command is use to give user access privileges to a database.

Syntax:

```
GRANT SELECT, UPDATE ON MY_TABLE TO SOME_USER, ANOTHER_USER;
```

For example:

```
GRANT SELECT ON Users TO'Tom'@'localhost;
```

Revoke:

It is useful to back permissions from the user.

Syntax:

```
REVOKE privilege_nameON object_nameFROM {user_name |PUBLIC
|role_name}
```

For example:

```
REVOKE SELECT, UPDATE ON student FROM BCA, MCA;
```

What is DQL?

Data Query Language (DQL) is used to fetch the data from the database. It uses only one command:

SELECT

SQL SELECT Statement

The SELECT statement is the most commonly used command in Structured Query Language. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

By using this command, we can also access the particular record from the particular column of the table. The table which stores the record returned by the SELECT statement is called a result-set table.

If you want to access all rows from all fields of the table, use the following SQL SELECT syntax with * asterisk sign:

Syntax of SELECT Statement in SQL

1. **SELECT** Column_Name_1, Column_Name_2, ..., Column_Name_N **FROM** Table_Name;

If you want to access all rows from all fields of the table, use the following SQL SELECT syntax with * asterisk sign:

1. **SELECT** * **FROM** table_name;

SQL supports following basic data type:-

SQL Data Type is an attribute that specifies the type of data of any object. Each column, variable and expression has a related data type in SQL. You can use these data types while creating your tables.

Numeric Data Types

DATA TYPE	FROM	TO
Float	-1.79E + 308	1.79E + 308
Real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
Datetime	Jan 1, 1753	Dec 31, 9999
Smalldatetime	Jan 1, 1900	Jun 6, 2079
Date	Stores a date like June 30, 1991	
Time	Stores a time of day like 12:30 P.M.	

- **Character**
    1. char(size) //Max size limit is 2000 bytes
    2. varchar(size) //Max size limit is 2000 bytes

3. varchar2(size) //Max size lime is 4000 bytes
4. long(size) //Max size limit is 2GB data

- **Number**
  1. number(p, s) // where **p** is precision & **s** is scale
     - //Max size is 38 digit

Decimal types

The DECIMAL data type is used to store exact numeric values in the database e.g., money values.

The following defines a column with the DECIMAL data type:

```
column_name DECIMAL (p,s)
```

In this syntax:

- p is the precision that represents the number of significant digits.
- s is the scale which represents the number of digits after the decimal point.

The maximum values of p and s depend on the implementation of each database system.

Date and Time types

The date and time data types are used to store information related to dates and times. SQL supports the following date and time data types:

- DATE
- TIME
- TIMESTAMP

DATE data type

The DATE data type represents date values that include three parts: year, month, and day. Typically, the range of the DATE data type is from 0001-01-01 to 9999-12-31.

The date value generally is specified in the form:

```
'YYYY-DD-MM'
Code language: SQL (Structured Query Language) (sql)
```

For example, the following DATE value is December 31, 2020:

```
'2020-12-31'
```

Code language: SQL (Structured Query Language) (sql)

TIME data type

The TIME data type store values representing a time of day in hours, minutes, and seconds.

The TIME values should be specified in the following form:

```
'HH:MM:SS'
```

Code language: SQL (Structured Query Language) (sql)

An optional fractional value can be used to store nanoseconds such as:

```
'10:59:30.9999'
```

Code language: SQL (Structured Query Language) (sql)

TIMESTAMP data type

The TIMESTAMP data type represents timestamp values which include both DATE and TIME values.

The TIMESTAMP values are specified in the following form:

```
TIMESTAMP 'YYYY-MM-DD HH:MM:SS'
```

Code language: SQL (Structured Query Language) (sql)

Notice that there is a space separator between the date and time parts.

SQL | Distinct Clause

The distinct keyword is used in conjunction with select keyword.
It is helpful when there is a need of avoiding duplicate values present in any specific columns/table.
When we use distinct keyword only the **unique values** are fetched.

Syntax :

```
SELECT DISTINCT column1, column2
FROM table_name ;
```

column1, column2 : Names of the fields of the table.
table_name : Table from where we want to fetch the records.

This query will return all the unique combinations of rows in the table with fields column1, column2.

NOTE: If distinct keyword is used with multiple columns, the distinct combination is displayed in the result set.

Table – Student

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	DELHI	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18

Queries:

- To fetch unique names from the NAME field –

```
SELECT DISTINCT NAME
FROM Student;
```

Output :

NAME
Ram
RAMESH
SUJIT
SURESH

- To fetch a unique combination of rows from the whole table –

```
SELECT DISTINCT *
FROM Student;
```

Output :

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18

Note : Without the keyword distinct in both the above examples 6 records would have been fetched instead of 4, since in the original table there are 6 records with the duplicate values.

SQL | WHERE Clause:

WHERE keyword is used for fetching **filtered data** in a result set.

- It is used to fetch data according to a particular criteria.
- WHERE keyword can also be used to filter data by matching patterns.

Basic Syntax:

SELECT column1,column2 FROM table_name WHERE column_name operator value;

column1 , column2: fields int the table
table_name: name of table
column_name: name of field used for filtering the data
operator: operation to be considered for filtering
value: exact value or pattern to get related data in result

List of operators that can be used with where clause:

operator	description
>	Greater Than
>=	Greater than or Equal to
<	Less Than
<=	Less than or Equal to
=	Equal to
<>	Not Equal to
BETWEEN	In an inclusive Range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

Queries

- To fetch record of students with age equal to 20

- `SELECT * FROM Student WHERE Age=20;`

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20

- To fetch Name and Address of students with ROLL_NO greater than 3
- `SELECT ROLL_NO,NAME,ADDRESS FROM Student WHERE ROLL_NO > 3;`

Output:

ROLL_NO	NAME	ADDRESS
4	SURESH	Delhi

BETWEEN operator

It is used to fetch filtered data in a given range inclusive of two values.

Basic Syntax:

SELECT column1,column2 FROM table_name WHERE column_name BETWEEN value1 AND value2;

BETWEEN: operator name

value1 AND value2: exact value from value1 to value2 to get related data in result set.

Queries:

- To fetch records of students where ROLL_NO is between 1 and 3 (inclusive)
- `SELECT * FROM Student WHERE ROLL_NO BETWEEN 1 AND 3;`

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20
3	SUJIT	ROHTAK	XXXXXXXXXX	20
2	RAMESH	GURGAON	XXXXXXXXXX	18

- To fetch NAME,ADDRESS of students where Age is between 20 and 30 (inclusive)
- `SELECT NAME,ADDRESS FROM Student WHERE Age BETWEEN 20 AND 30;`

  Output:

NAME	ADDRESS
SUJIT	Rohtak
SUJIT	Rohtak

LIKE operator:

It is used to fetch filtered data by searching for a particular pattern in where clause.

Basic Syntax:

SELECT column1,column2 FROM table_name WHERE column_name LIKE pattern;

`LIKE:` operator name
pattern: exact value extracted from the pattern to get related data in result set.
Note: The character(s) in pattern are case sensitive.

Queries

- To fetch records of students where NAME starts with letter S.
  `SELECT * FROM Student WHERE NAME LIKE 'S%';`

The '%'(wildcard) signifies the later characters here which can be of any length and value. More about wildcards will be discussed in the later set.

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
3	SUJIT	ROHTAK	XXXXXXXXXX	20
4	SURESH	Delhi	XXXXXXXXXX	18
3	SUJIT	ROHTAK	XXXXXXXXXX	20

- To fetch records of students where NAME contains the patter 'AM'.
- `SELECT * FROM Student WHERE NAME LIKE '%AM%';`

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18
2	RAMESH	GURGAON	XXXXXXXXXX	18

IN operator:

It is used to fetch filtered data same as fetched by '=' operator just the difference is that here we can specify multiple values for which we can get the result set.

Basic Syntax:

SELECT column1,column2 FROM table_name WHERE column_name IN (value1,value2,..);

IN: operator name
value1,value2,..: exact value matching the values given and get related data in result set.

Queries

- To fetch NAME and ADDRESS of students where Age is 18 or 20.
- `SELECT NAME,ADDRESS FROM Student WHERE Age IN (18,20);`

Output:

NAME	ADDRESS
Ram	Delhi
RAMESH	GURGAON
SUJIT	ROHTAK
SURESH	Delhi
SUJIT	ROHTAK
RAMESH	GURGAON

- To fetch records of students where ROLL_NO is 1 or 4.
- `SELECT * FROM Student WHERE ROLL_NO IN (1,4);`

Output:

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	XXXXXXXXXX	18
4	SURESH	Delhi	XXXXXXXXXX	18

You can rename a table or a column temporarily by giving another name known as **Alias**. The use of table aliases is to rename a table in a specific SQL statement. The renaming is a temporary change and the actual table name does not change in the database. The column aliases are used to rename a table's columns for the purpose of a particular SQL query.

Syntax

The basic syntax of a table alias is as follows.

```
SELECT column1, column2....
FROM table_name AS alias_name;
```

The basic syntax of a column alias is as follows.

```
SELECT column_name AS alias_name
FROM table_name;
```

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

Table 2 − ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, the following code block shows the usage of a **table alias**.

```
SQL> SELECT C.ID, C.NAME, C.AGE, O.AMOUNT
   FROM CUSTOMERS AS C, ORDERS AS O
   WHERE  C.ID = O.CUSTOMER_ID;
```

This would produce the following result.

```
+----+----------+-----+--------+
| ID | NAME     | AGE | AMOUNT |
+----+----------+-----+--------+
|  3 | kaushik  |  23 |   3000 |
|  3 | kaushik  |  23 |   1500 |
|  2 | Khilan   |  25 |   1560 |
|  4 | Chaitali |  25 |   2060 |
+----+----------+-----+--------+
```

Following is the usage of a **column alias**.

```
SQL> SELECT  ID AS CUSTOMER_ID, NAME AS CUSTOMER_NAME
   FROM CUSTOMERS;
```

This would produce the following result.

```
+-------------+---------------+
| CUSTOMER_ID | CUSTOMER_NAME |
+-------------+---------------+
|           1 | Ramesh        |
```

```
|           2 | Khilan         |
|           3 | kaushik        |
|           4 | Chaitali       |
|           5 | Hardik         |
|           6 | Komal          |
|           7 | Muffy          |
+-------------+----------------+
```

Single Row Functions Examples

Oracle provides single row functions to manipulate the data values.

The single row functions operate on single rows and return only one result per row.

In general, the functions take one or more inputs as arguments and return a single value as output.

The arguments can be a user-supplied constant, variable, column name and an expression.

The features of single row functions are:

The single row functions are categorized into:

- **Character Functions:** Character functions accept character inputs and can return either character or number values as output.
- **Number Functions:** Number functions accepts numeric inputs and returns only numeric values as output.
- **Date Functions:** Date functions operate on date data type and returns a date value or numeric value.
- **Conversions Functions:** Converts from one data type to another data type.

Character Functions Example

Character Functions

SQL provides a rich set of character functions that allow you to get information about strings and modify the contents of those strings in multiple ways.
Character functions are of the following two types:
1. Case-Manipulative Functions (LOWER, UPPER and INITCAP)

2. Character-Manipulative Functions (CONCAT, LENGTH, SUBSTR, INSTR, LPAD, RPAD, TRIM and REPLACE)

1. LOWER

The Lower function converts the character values into lowercase letters.

```
Input1: SELECT LOWER('COCSIT') FROM DUAL;
Output1: cocsit
```

2. UPPER

The Upper function converts the character values into uppercase letters.

```
Input1: SELECT UPPER('cocsit') FROM DUAL;
Output1: COCSIT
```

3. INITCAP

The Initcap function coverts the first character of each word into uppercase and the remaining characters into lowercase.

```
Input2: SELECT INITCAP('PRACTICE_CODING_FOR_EFFICIENCY') FROM DUAL;
Output2: Practice_Coding_For_Efficiency
```

Character-Manipulative Functions

1. **CONCAT :** This function always appends (concatenates) string2 to the end of string1. If either of the string is NULL, CONCAT function returns the non-NULL argument. If both strings are NULL, CONCAT returns NULL.

Syntax:

```
CONCAT('String1', 'String2')
```

```
Input1: SELECT CONCAT('computer' ,'science') FROM DUAL;
Output1: computerscience

Input2: SELECT CONCAT( NULL ,'Android') FROM DUAL;
Output2: Android

Input3: SELECT CONCAT( NULL ,NULL ) FROM DUAL;
Output3: -
```

2. **LENGTH :** This function returns the length of the input string. If the input string is NULL, then LENGTH function returns NULL and not Zero. Also, if the input string contains extra spaces at the start, or in between or at the end of the string, then the LENGTH function includes the extra spaces too and returns the complete length of the string.
   Syntax:

```
LENGTH(Column|Expression)
Input1: SELECT LENGTH('Learning Is Fun') FROM DUAL;
Output1: 15

Input2: SELECT LENGTH('   Write an Interview  Experience ') FROM
DUAL;
Output2: 34
```

3. **SUBSTR :** This function returns a portion of a string from a given start point to an end point. If a substring length is not given, then SUBSTR returns all the characters till the end of string (from the starting position specified).
   Syntax:

```
SUBSTR('String',start-index,length_of_extracted_string)
Input1: SELECT SUBSTR('Database Management System', 9) FROM DUAL;
Output1: Management System

Input2: SELECT SUBSTR('Database Management System', 9, 7) FROM
DUAL;
Output2: Manage
```

4. LPAD and RPAD : These functions return the strings padded to the left or right (as per the use) ; hence the “L” in “LPAD” and the “R” in “RPAD” ; to a specified length, and with a specified pad string. If the pad string is not specified, then the given string is padded on the left or right (as per the use) with spaces.

Syntax:

```
LPAD(Column|Expression, n, 'String')
Syntax: RPAD(Column|Expression, n, 'String')
LPAD Input1: SELECT LPAD('100',5,'*') FROM DUAL;
LPAD Output1: **100

RPAD Input1: SELECT RPAD('5000',7,'*') FROM DUAL;
RPAD Output1: 5000***

RPAD Input1: SELECT RPAD('earn', 19, 'money') FROM DUAL;
RPAD Output1: earnmoneymoneymoney
```

4. **TRIM :** This function trims the string input from the start or end (or both). If no string or char is specified to be trimmed from the string and there exists some extra space at start or end of the string, then those extra spaces are trimmed off.

Syntax:

TRIM (Leading|Trailing|Both, trim_character FROM trim_source)

Input1: SELECT TRIM('G' FROM 'GEEKS') FROM DUAL;

Output1: EEKS

Input2: SELECT TRIM(' geeksforgeeks ') FROM DUAL;

Output2: geeksforgeeks

5. **REPLACE :**

This function searches for a character string and, if found, replaces it with a given replacement string at all the occurrences of the string.

REPLACE is useful for searching patterns of characters and then changing all instances of that pattern in a single function call.

If a replacement string is not given, then REPLACE function removes all the occurrences of that character string in the input string.
If neither a match string nor a replacement string is specified, then REPLACE returns NULL.

Syntax:

REPLACE(Text, search_string, replacement_string)

Input1:
SELECT REPLACE('DATA MANAGEMENT', 'DATA','DATABASE') FROM DUAL;
Output1: DATABASE MANAGEMENT

Input2:
SELECT REPLACE('abcdeabcccabdddeeabcc', 'abc') FROM DUAL;
Output2: deccabdddeec

Number Functions Example

1. ROUND

The Round function rounds the value to the n decimal values. If n is not specified, there won't be any decimal places. If n is negative, numbers to the left of the decimal point are rounded.

Syntax: round(number,n)

```
SELECT round(123.67,1) FROM DUAL;
SELECT round(123.67) FROM DUAL;
SELECT round(123.67,-1) FROM DUAL;
```

2. TRUNC

The Trunc function truncates the value to the n decimal places. If n is omitted, then n defaults to zero.

```
Syntax: trunc(number,n)

SELECT trunc(123.67,1) FROM DUAL;
SELECT trunc(123.67) FROM DUAL;
```

3. MOD

The Mod function returns the remainder of m divided by n.

```
Syntax: mod(m,n)

SELECT mod(10,5) FROM DUAL;
```

4. ABS(X)

The ABS() function returns the absolute value of X. Consider the following example −

```
SQL> SELECT ABS(2);
+----------------------------------------------------------+
| ABS(2)                                                   |
```

```
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)

SQL> SELECT ABS(-2);
+-----------------------------------------------------------+
| ABS(2)                                                    |
+-----------------------------------------------------------+
| 2                                                         |
+-----------------------------------------------------------+
1 row in set (0.00 sec)
```

Date Functions Example

1. SYSDATE

The Sysdate function returns the current oracle database server date and time.

```
SELECT sysdate FROM DUAL;
```

2. Arithmetic with Dates

You can add or subtract the number of days or hours to the dates. You can also subtract the dates

```
SELECT sysdate+2 "add_days" FROM DUAL;
SELECT sysdate-3 "sub_days" FROM DUAL;
SELECT sysdate+3/24 "add_hours" FROM DUAL;
SELECT sysdate-2/24 "sub_hours" FROM DUAL;
SELECT sysdate-hire_date "sub_dates" FROM EMPLOYEES; -- returns
number of days between the two dates.
```

3. MONTHS_BETWEEN

The Months_Between function returns the number of months between the two given dates.

```
Syntax: months_between(date1,date2)
SELECT months_between(sysdate,hire_date) FROM EMPLOYEES:
SELECT months_between('01-JUL-2000', '23-JAN-2000') FROM DUAL;
```

4. ADD_MONTHS

The Add_Months is used to add or subtract the number of calendar months to the given date.

```
Syntax: add_months(date,n)

SELECT add_months(sysdate,3) FROM DUAL;
SELECT add_months(sysdate,-3) FROM DUAL;
SELECT add_months('01-JUL-2000', 3) FROM DUAL;
```

5. NEXT_DAY

The Next_Day function finds the date of the next specified day of the week. The syntax is

NEXT_DAY(date,'char')

The char can be a character string or a number representing the day.

```
SELECT next_day(sysdate,'FRIDAY') FROM DUAL;
SELECT next_day(sysdate,5) FROM DUAL;
```

```
SELECT next_day('01-JUL-2000', 'FRIDAY') FROM DUAL;
```

6. LAST_DAY

The Last_Day function returns the last day of the month.

```
SELECT last_day(sysdate) FROM DUAL;
SELECT last_day('01-JUL-2000') FROM DUAL;
```

7. ROUND

The Round function returns the date rounded to the specified format. The Syntax is Round(date [,'fmt'])

```
SELECT round(sysdate,'MONTH') FROM DUAL;
SELECT round(sysdate,'YEAR') FROM DUAL;
SELECT round('30-OCT-85','YEAR') FROM DUAL;
```

8. TRUNC

The Trunc function returns the date truncated to the specified format. The Syntax is Trunc(date [,'fmt'])

```
SELECT trunc(sysdate,'MONTH') FROM DUAL;
SELECT trunc(sysdate,'YEAR') FROM DUAL;
SELECT trunc('01-MAR-85','YEAR') FROM DUAL;
```

Type Conversion Functions

TO_CHAR -converts number or date to VARCHAR2
TO_NUMBER : Convert character to number
TO_DATE: It converts to date

Examples

```
SQL> select to_char (sysdate ,'DD-MON-YYYY,MI') from dual;

TO_CHAR(SYSDATE,'DD-

--------------------

01-SEP-2019,34
```

Multiple Row Functions in Oracle with Examples

1. **SUM() Function**
2. **AVG() Function**
3. **MIN() Function**
4. **MAX() Function**
5. **COUNT() Function**

The Multiple Row Functions in Oracle are used to return either group of values (or) a single value.
These functions are basically operated on a set of rows and return one result or one result per group.
This is a powerful feature provided by oracle because these Multiple Row Functions allow us to generate subtotals, max. min, sums, and averages within the SQL that is retrieving the data.
The Multiple row function in Oracle is also called group functions or it is also called aggregate functions.

Types of Multiple Row Functions:

1. **AVG**: It retrieves the average value of the number of rows in a table by ignoring the null value

2. **COUNT**: It retrieves the number of rows (count all selected rows using *, including duplicates and rows with null values)
3. **MAX**: It retrieves the maximum value of the expression, ignores null values
4. **MIN**: It retrieves the minimum value of the expression, ignores null values
5. **SUM**: It retrieves the sum of values of the number of rows in a table, it ignores null values

We are going to use the following EMP table to understand the Oracle Multiple Row Functions with examples.

For ex:

```
CREATE TABLE EMP
(EMPNO NUMBER(4) NOT NULL,
ENAME VARCHAR2(10),
JOB VARCHAR2(9),
MGR NUMBER(4),
HIREDATE DATE,
SAL NUMBER(7, 2),
COMM NUMBER(7, 2),
DEPTNO NUMBER(2));

INSERT INTO EMP VALUES
(7369, 'SMITH', 'CLERK', 7902,
TO_DATE('17-DEC-1980', 'DD-MON-YYYY'), 800, NULL, 20);
INSERT INTO EMP VALUES
(7499, 'ALLEN', 'SALESMAN', 7698,
TO_DATE('20-FEB-1981', 'DD-MON-YYYY'), 1600, 300, 30);
```

```
SQL> SELECT * FROM EMP;

     EMPNO ENAME      JOB              MGR HIREDATE                 SAL       COMM     DEPTNO
---------- ---------- --------- ---------- ------------------- ---------- ---------- ----------
      7369 SMITH      CLERK           7902 17-DEC-1980 00:00:00        800                    20
      7499 ALLEN      SALESMAN        7698 20-FEB-1981 00:00:00       1600        300         30
      7521 WARD       SALESMAN        7698 22-FEB-1981 00:00:00       1250        500         30
      7566 JONES      MANAGER         7839 02-APR-1981 00:00:00       2975                    20
      7654 MARTIN     SALESMAN        7698 28-SEP-1981 00:00:00       1250       1400         30
      7698 BLAKE      MANAGER         7839 01-MAY-1981 00:00:00       2850                    30
      7782 CLARK      MANAGER         7839 09-JUN-1981 00:00:00       2450                    10
      7788 SCOTT      ANALYST         7566 09-DEC-1982 00:00:00       3000                    20
      7839 KING       PRESIDENT            17-NOV-1981 00:00:00       5000                    10
      7844 TURNER     SALESMAN        7698 08-SEP-1981 00:00:00       1500          0         30
      7876 ADAMS      CLERK           7788 12-JAN-1983 00:00:00       1100                    20

     EMPNO ENAME      JOB              MGR HIREDATE                 SAL       COMM     DEPTNO
---------- ---------- --------- ---------- ------------------- ---------- ---------- ----------
      7900 JAMES      CLERK           7698 03-DEC-1981 00:00:00        950                    30
      7902 FORD       ANALYST         7566 03-DEC-1981 00:00:00       3000                    20
      7934 MILLER     CLERK           7782 23-JAN-1982 00:00:00       1300                    10

14 rows selected.
```

SQL SUM Function:

The SUM Aggregate Function in sql is used to return the total sum of a given numeric column. The SUM function will only work on numeric data types. For columns containing string values, it will always return 0.
Following is the syntax to use the SUM Function in SQL.

Syntax:
SELECT SUM(column) FROM TableName;

Example:
SELECT SUM(SAL) FROM EMP;

Output:

```
SQL> SELECT SUM(SAL) FROM EMP;

  SUM(SAL)
----------
     29025
```

Example 2 :
SELECT SUM(SAL) FROM EMP WHERE JOB='CLERK';

Output:

```
SQL> SELECT SUM(SAL) FROM EMP WHERE JOB='CLERK';

  SUM(SAL)
----------
      4150
```

AVG() Function:

The AVG function in sql is used to return the average value of the given numeric column. The AVG function will only work on numeric columns.
For columns containing string values, it will always return 0. Following is the syntax to use the AVG function in sql.

Syntax: SELECT AVG(columnname) FROM tablename;

Example:
SELECT AVG(SAL) FROM EMP;
Output:

```
SQL> SELECT AVG(SAL) FROM EMP;

  AVG(SAL)
----------
2073.21429
```

MIN() Function in SQL:

The MIN function in Oracle is used to return the smallest value of the given column. The MIN function works on numeric columns as well as string columns. The following is the syntax to use of MIN function in Oracle.

Syntax: SELECT MIN(columnname) FROM tablename;
Example:

SELECT MIN(HIREDATE) FROM EMP;

Output:

```
SQL> SELECT MIN(HIREDATE) FROM EMP;

MIN(HIREDATE)
--------------------
17-DEC-1980 00:00:00
```

Example:

SELECT MIN(HIREDATE) FROM EMP WHERE JOB='MANAGER';

Output:

```
SQL> SELECT MIN(HIREDATE) FROM EMP WHERE JOB='MANAGER';

MIN(HIREDATE)
--------------------
02-APR-1981 00:00:00
```

Example:

SELECT MIN(SAL) FROM EMP;

Output:

```
SQL> SELECT MIN(SAL) FROM EMP;

  MIN(SAL)
----------
       800
```

MAX() Function in SQL:

The MAX function in sql is used to return the largest value of the given column.
The MAX function works on numeric columns as well as string columns.

Syntax: SELECT MAX(columnname) FROM tablename;
Example:

SELECT MAX(HIREDATE) FROM EMP;
Output:

```
SQL> SELECT MAX(HIREDATE) FROM EMP;

MAX(HIREDATE)
--------------------
12-JAN-1983 00:00:00
```

SQL CREATE VIEW Statement

In SQL, a view is a virtual table based on the result-set of an SQL statement.

A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.

You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

A view is created with the `CREATE VIEW` statement.

CREATE VIEW Syntax

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name;
```

SQL CREATE VIEW Examples

The following SQL creates a view that shows all customers from Brazil:

Example

```
CREATE VIEW [Brazil Customers] AS
SELECT *
FROM Customers;
```

Example

```
SELECT * FROM [Brazil Customers];
```

SQL Dropping a View

A view is deleted with the `DROP VIEW` statement.

SQL DROP VIEW Syntax

```
DROP VIEW view_name;
```

The following SQL drops the "Brazil Customers" view:

Example

```
DROP VIEW [Brazil Customers];
```

Update View

The SQL UPDATE VIEW command can be used to modify the data of a view.

All views are not updatable. So, UPDATE command is not applicable to all views. An updatable view is one which allows performing a UPDATE command on itself without affecting any other table.

Syntax:

UPDATE < view_name > SET<column1>=<value1>,<column2>=<value2>,.....

WHERE <condition>;

SQL Code:

```
UPDATE agentview

SET commission=.13

WHERE working_area=’London’;
```

Basic Structure of SQL Queries

The basic structure of an SQL query consists of **three clauses:**

1. **select**,

2. from
3. where.

The query takes as its input the relations listed in the **from** clause, operates
on them as specified in the **where** and **select** clauses, and then produces a relation
as the result.

We introduce the SQL syntax through examples, and describe the general structure of SQL queries later.

1. Qeries on a Single Relation

Let us consider a simple query using our university example, "Find the names
of all instructors." Instructor names are found in the *instructor* relation, so we put that relation in the from clause.

The instructor's name appears in the name
attribute, so we put that in the select clause.

select *name*
from *instructor*;

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

Figure 3.2 Result of "**select** *name* **from** *instructor*"

The result is a relation consisting of a single attribute with the heading name. If the *instructor* relation is as shown in Figure 2.1, then the relation that results from
the preceding query is shown in Figure 3.2.

Now consider another query, "Find the department names of all instructors,"
which can be written as:

select *dept_name*
from *instructor*;

In those cases where we want to force the elimination of duplicates, we insert
the keyword **distinct** after **select**. We can rewrite the preceding query as:

select distinct *dept_name*
from *instructor*;

dept_name
Comp. Sci.
Finance
Music
Physics
History
Physics
Comp. Sci.
History
Finance
Biology
Comp. Sci.
Elec. Eng.

Figure 3.3 Result of "**select** *dept_name* **from** *instructor*"

if we want duplicates removed. The result of the above query would contain each
department name at most once.

SQL allows us to use the keyword **all** to specify explicitly that duplicates are
not removed:

select all *dept_name*
from *instructor*;

Since duplicate retention is the default, we shall not use **all** in our examples. To

example,
the query:

select *ID*, *name*, *dept_name*, *salary* * **from** *instructor*;

returns a relation that is the same as the instructor relation, except that The **where** clause allows us to select only those rows in the result relation of the **from** clause that satisfy a specified predicate. Consider the query "Find the names of all instructors in the Computer Science department who have salary greater than $70,000." This query can be written in SQL as:

select *name*
from *instructor*

where *dept_name* = 'Comp. Sci.' **and** *salary* > 70000;

Output:

name
Katz
Brandt

Figure 3.4 Result of "Find the names of all instructors in the Computer Science department who have salary greater than $70,000."

2. Queries on Multiple Relations

So far our example queries were on a single relation. Queries often need to access
information from multiple relations. We now study how to write such queries.

In SQL, to answer the above query, we list the relations that need to be accessed
in the **from** clause, and specify the matching condition in
the **where** clause. The
above query can be written in SQL as

select *name*, *instructor*.*dept_name*, *building*
from *instructor*, *department*
where *instructor*.*dept_name*= *department*.*dept_name*;

Output:

name	*dept_name*	*building*
Srinivasan	Comp. Sci.	Taylor
Wu	Finance	Painter
Mozart	Music	Packard
Einstein	Physics	Watson
El Said	History	Painter
Gold	Physics	Watson
Katz	Comp. Sci.	Taylor
Califieri	History	Painter
Singh	Finance	Painter
Crick	Biology	Watson
Brandt	Comp. Sci.	Taylor
Kim	Elec. Eng.	Taylor

Figure 3.5 The result of "Retrieve the names of all instructors, along with their department names and department building name."