# 3.2. Student Handout

# Student Handout: Introduction to Databases and SQL for Data Analysis

## Overview

Welcome to your guide on using SQL for data analysis! This handout will provide you with a foundational understanding of SQL and its application in data analysis. By the end, you'll be equipped to write basic SQL queries and perform data analysis tasks.

---

## What is SQL?

**SQL (Structured Query Language)** is a programming language used for managing and manipulating databases. It allows you to communicate with a database to retrieve, update, or delete data.

### Why Use SQL for Data Analysis?

- **Efficiency**: Optimized for handling large datasets.
- **Simplicity**: Easy-to-learn syntax.
- **Flexibility**: Compatible with various databases like MySQL, PostgreSQL, and SQLite.

---

## Relational Databases

A **relational database** stores data in tables, similar to an Excel spreadsheet. Each table consists of rows (records) and columns (attributes).

### Example Table: `Students`

| Student_ID | Name | Age | Grade |
|---|---|---|---|
| 1 | Rahul | 15 | 10 |
| 2 | Priya | 14 | 9 |

| Student_ID | Name | Age | Grade |
|---|---|---|---|
| 3 | Ankit | 16 | 11 |

## Popular Relational Databases

- **MySQL**
- **PostgreSQL**
- **SQLite**

---

# Basic SQL Queries for Data Extraction

## SELECT Statement

Retrieve data from a table.

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

**Examples:**

1. Retrieve names and ages from `Students`:

```
SELECT Name, Age FROM Students;
```

2. Retrieve all columns from `Students`:

```
SELECT * FROM Students;
```

3. Retrieve student IDs and grades:

```
SELECT Student_ID, Grade FROM Students;
```

## Filtering Data with WHERE

Filter records based on specific conditions.

```sql
SELECT column1, column2 FROM table_name WHERE condition;
```

**Examples:**

1. Students aged 15 or older:

```sql
SELECT Name, Age FROM Students WHERE Age >= 15;
```

2. Students in grade 10:

```sql
SELECT Name FROM Students WHERE Grade = 10;
```

3. Students named "Priya":

```sql
SELECT * FROM Students WHERE Name = 'Priya';
```

## Sorting Data with ORDER BY

Sort data in ascending or descending order.

```sql
SELECT column1, column2 FROM table_name ORDER BY column1 ASC/DESC;
```

**Examples:**

1. Sort students by age descending:

```sql
SELECT Name, Age FROM Students ORDER BY Age DESC;
```

2. Sort students by name ascending:

```sql
SELECT Name, Age FROM Students ORDER BY Name ASC;
```

3. Sort by grade, then by age:

```sql
SELECT Name, Age, Grade FROM Students ORDER BY Grade, Age;
```

---

## Grouping Data with GROUP BY

Group rows with the same values in specified columns.

```sql
SELECT column1, COUNT(*) FROM table_name GROUP BY column1;
```

**Examples:**

1. Count students in each grade:

```sql
SELECT Grade, COUNT(*) FROM Students GROUP BY Grade;
```

2. Average age per grade:

```sql
SELECT Grade, AVG(Age) FROM Students GROUP BY Grade;
```

3. Maximum age per grade:

```sql
SELECT Grade, MAX(Age) FROM Students GROUP BY Grade;
```

---

## Aggregation Functions in SQL

Perform calculations on data.

- **SUM()**: Adds values.
- **AVG()**: Calculates average.
- **COUNT()**: Counts rows.
- **MAX()**: Finds maximum value.
- **MIN()**: Finds minimum value.

**Examples:**

1. Average age of students:

```sql
SELECT AVG(Age) FROM Students;
```

2. Total number of students:

```sql
SELECT COUNT(*) FROM Students;
```

3. Sum of ages:

```sql
SELECT SUM(Age) FROM Students;
```

---

# Joins in SQL

Combine data from multiple tables.

## Types of Joins

- **INNER JOIN**: Rows with matching values in both tables.
- **LEFT JOIN**: All rows from the left table, matching rows from the right.
- **RIGHT JOIN**: All rows from the right table, matching rows from the left.
- **FULL JOIN**: All rows with matches in either table.

**Examples:**

1. Inner join `Students` and `Courses`:

```sql
SELECT Students.Name, Courses.Course_Name

FROM Students

INNER JOIN Courses ON Students.Student_ID = Courses.Student_ID;
```

2. Left join `Students` and `Courses`:

```sql
SELECT Students.Name, Courses.Course_Name

FROM Students

LEFT JOIN Courses ON Students.Student_ID = Courses.Student_ID;
```

3. Right join `Students` and `Courses`:

```sql
SELECT Students.Name, Courses.Course_Name

FROM Students

RIGHT JOIN Courses ON Students.Student_ID = Courses.Student_ID;
```

## Subqueries and Nested Queries

A query within another query for complex analysis.

**Examples:**

1. Students older than average age:

```sql
SELECT Name FROM Students WHERE Age > (SELECT AVG(Age) FROM Students);
```

2. Courses with more than one student:

```sql
SELECT Course_Name FROM Courses WHERE Course_ID IN (SELECT Course_ID FROM
Enrollments GROUP BY Course_ID HAVING COUNT(Student_ID) > 1);
```

3. Students in the top 10% by age:

```sql
SELECT Name FROM Students WHERE Age > (SELECT PERCENTILE_CONT(0.9) WITHIN
GROUP (ORDER BY Age) FROM Students);
```

## Writing Efficient SQL Queries for Large Datasets

- **Use Indexes**: Speed up data retrieval.
- **Limit the Data**: Use **LIMIT** to restrict rows.
- **Avoid SELECT ***: Select only needed columns.
- **Use Joins Wisely**: Join tables only when necessary.

## Hands-On: Performing Data Analysis Using SQL

## Example Queries

1. **Total Sales for Each Product**:

```sql
SELECT Products.Product_Name, SUM(Sales.Quantity * Products.Price) AS Total_Sales

FROM Sales

INNER JOIN Products ON Sales.Product_ID = Products.Product_ID

GROUP BY Products.Product_Name;
```

2. **Top 2 Most Sold Products**:

```sql
SELECT Products.Product_Name, SUM(Sales.Quantity) AS Total_Quantity

FROM Sales

INNER JOIN Products ON Sales.Product_ID = Products.Product_ID

GROUP BY Products.Product_Name

ORDER BY Total_Quantity DESC

LIMIT 2;
```

3. **Sales on a Specific Date**:

```sql
SELECT Products.Product_Name, Sales.Quantity

FROM Sales

INNER JOIN Products ON Sales.Product_ID = Products.Product_ID

WHERE Sales.Sale_Date = '2023-01-01';
```

## Conclusion

SQL is a powerful tool for data analysis. Practice writing queries and analyzing data to become proficient in SQL. Happy querying!