SQL Querying: A Beginner's Guide - Student Handout

Welcome to the world of SQL (Structured Query Language)! This handout will guide you through the basics and some advanced concepts of SQL, the language used to interact with databases.

---

1. Basic SQL Queries

a. SELECT Statement

The `SELECT` statement retrieves data from a database.

Syntax:

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

Examples:

1. Retrieve names and ages from the students table:

   ```
   SELECT name, age FROM students;
   ```

2. Retrieve all columns from the courses table:

   ```
   SELECT * FROM courses;
   ```

3. Retrieve distinct course names from the courses table:

   ```
   SELECT DISTINCT course_name FROM courses;
   ```

b. INSERT Statement

The `INSERT` statement adds new data to a table.

Syntax:

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

Examples:

1. Add a new student:

   ```
   INSERT INTO students (name, age) VALUES ('Rahul', 22);
   ```

2. Add a new course:

   ```
   INSERT INTO courses (course_name) VALUES ('Mathematics');
   ```

3. Add multiple students:

   ```
   INSERT INTO students (name, age) VALUES ('Anjali', 21), ('Vikram', 23);
   ```

c. UPDATE Statement

The `UPDATE` statement modifies existing data in a table.

Syntax:

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Examples:

1. Update a student's age:

   ```
   UPDATE students SET age = 23 WHERE name = 'Rahul';
   ```

2. Change a course name:

```sql
UPDATE courses SET course_name = 'Physics' WHERE course_id = 1;
```

3. Update multiple columns:

```sql
UPDATE students SET age = 24, name = 'Rohan' WHERE student_id = 2;
```

d. DELETE Statement

The `DELETE` statement removes data from a table.

Syntax:

```sql
DELETE FROM table_name WHERE condition;
```

Examples:

1. Remove a student:

```sql
DELETE FROM students WHERE name = 'Rahul';
```

2. Delete a course:

```sql
DELETE FROM courses WHERE course_name = 'Mathematics';
```

3. Remove students older than 25:

```sql
DELETE FROM students WHERE age > 25;
```

---

2. Data Filtering Using WHERE Clause

The `WHERE` clause filters records based on a condition.

Syntax:

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

Examples:

1. Students older than 20:

```
SELECT name, age FROM students WHERE age > 20;
```

2. Courses with a specific ID:

```
SELECT course_name FROM courses WHERE course_id = 2;
```

3. Students named 'Anjali':

```
SELECT * FROM students WHERE name = 'Anjali';
```

---

3. Sorting and Limiting Results

a. ORDER BY Clause

Sorts the result set in ascending or descending order.

Syntax:

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1 ASC|DESC;
```

Examples:

1. Students sorted by age:

```
SELECT name, age FROM students ORDER BY age DESC;
```

2. Courses sorted by name:

```sql
SELECT * FROM courses ORDER BY course_name ASC;
```

3. Students sorted by name:

```sql
SELECT * FROM students ORDER BY name;
```

b. LIMIT Clause

Limits the number of records returned by a query.

Syntax:

```sql
SELECT column1, column2, ...
FROM table_name
LIMIT number;
```

Examples:

1. First 5 students:

```sql
SELECT name, age FROM students LIMIT 5;
```

2. Top 3 courses:

```sql
SELECT * FROM courses LIMIT 3;
```

3. First 10 students sorted by age:

```sql
SELECT * FROM students ORDER BY age LIMIT 10;
```

---

4. SQL Joins

Joins combine data from multiple tables based on a related column.

a. INNER JOIN

Returns records with matching values in both tables.

Syntax:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Examples:

1. Students and their courses:

```
SELECT students.name, courses.course_name
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id;
```

2. Students with specific course IDs:

```
SELECT students.name, courses.course_name
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id
WHERE courses.course_id = 1;
```

3. Courses with enrolled students:

```
SELECT courses.course_name, students.name
FROM courses
INNER JOIN students
ON courses.course_id = students.course_id;
```

b. LEFT JOIN

Returns all records from the left table and matched records from the right table.

Syntax:

```
SELECT columns
FROM table1
LEFT JOIN table2
ON table1.column = table2.column;
```

Examples:

1. All students and their courses:

```
SELECT students.name, courses.course_name
FROM students
LEFT JOIN courses
ON students.course_id = courses.course_id;
```

2. Students with or without courses:

```
SELECT students.name, courses.course_name
FROM students
LEFT JOIN courses
ON students.course_id = courses.course_id
WHERE courses.course_name IS NULL;
```

3. All courses with student names:

```
SELECT courses.course_name, students.name
FROM courses
LEFT JOIN students
ON courses.course_id = students.course_id;
```

c. RIGHT JOIN

Returns all records from the right table and matched records from the left table.

Syntax:

```
SELECT columns
FROM table1
RIGHT JOIN table2
ON table1.column = table2.column;
```

Examples:

1. All courses and enrolled students:

```
SELECT students.name, courses.course_name
FROM students
RIGHT JOIN courses
ON students.course_id = courses.course_id;
```

2. Courses with or without students:

```
SELECT courses.course_name, students.name
FROM courses
RIGHT JOIN students
ON courses.course_id = students.course_id
WHERE students.name IS NULL;
```

3. All students with course names:

```
SELECT students.name, courses.course_name
FROM students
RIGHT JOIN courses
ON students.course_id = courses.course_id;
```

d. FULL JOIN

Returns all records when there is a match in either left or right table.

Syntax:

```
SELECT columns
FROM table1
FULL JOIN table2
ON table1.column = table2.column;
```

Examples:

1. All students and courses:

```sql
SELECT students.name, courses.course_name
FROM students
FULL JOIN courses
ON students.course_id = courses.course_id;
```

2. Students and courses with no matches:

```sql
SELECT students.name, courses.course_name
FROM students
FULL JOIN courses
ON students.course_id = courses.course_id
WHERE students.course_id IS NULL OR courses.course_id IS NULL;
```

3. All data from both tables:

```sql
SELECT * FROM students
FULL JOIN courses
ON students.course_id = courses.course_id;
```

---

5. Advanced SQL

a. Aggregate Functions

Perform calculations on a set of values and return a single value.

Examples:

1. Count students:

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

2. Average student age:

```sql
SELECT AVG(age) FROM students;
```

3. Maximum age of students:

```sql
SELECT MAX(age) FROM students;
```

b. GROUP BY and HAVING Clauses

Group rows with the same values and filter groups based on a condition.

Syntax:

```sql
SELECT column1, COUNT(*)
FROM table_name
GROUP BY column1
HAVING COUNT(*) > 1;
```

Examples:

1. Students per course:

```sql
SELECT course_id, COUNT(*)
FROM students
GROUP BY course_id;
```

2. Courses with more than 5 students:

```sql
SELECT course_id, COUNT(*)
FROM students
GROUP BY course_id
HAVING COUNT(*) > 5;
```

3. Average age per course:

```sql
SELECT course_id, AVG(age)
FROM students
GROUP BY course_id;
```

c. Subqueries and Nested Queries

A query within another query.

Syntax:

```
SELECT column1
FROM table_name
WHERE column2 = (SELECT column2 FROM another_table WHERE condition);
```

Examples:

1. Oldest student:

```
SELECT name
FROM students
WHERE age = (SELECT MAX(age) FROM students);
```

2. Students in a specific course:

```
SELECT name
FROM students
WHERE course_id = (SELECT course_id FROM courses WHERE course_name =
'Mathematics');
```

3. Students older than average age:

```
SELECT name
FROM students
WHERE age > (SELECT AVG(age) FROM students);
```

Activity: Write SQL Queries for a Sample Database

Task 1: Fetch student names and their courses.

```
SELECT students.name, courses.course_name
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id;
```

Task 2: Find the total number of students enrolled in each course.

```sql
SELECT courses.course_name, COUNT(*)
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id
GROUP BY courses.course_name;
```

Conclusion

This handout provides a comprehensive overview of SQL querying, covering basic commands like `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, as well as advanced topics like joins, aggregate functions, and subqueries. Practice these queries to become proficient in SQL!