Xi'an Jiaotong-Liverpool University
西交利物浦大学

DTS207TC - Database Development and Design : Introduction to Basic SQL

Course Leader: Dr.Zhang Di, Co-Teacher: Dr. Affan Yasin

September 11, 2025

Contents

Instructions

1. **Installation of PostgreSQL.** The step-by-step procedure (with images) for installing **PostgreSQL** is provided in the file uploaded on the **Learning Mall course folder**. Please refer to that document if you need to install PostgreSQL on your personal system. **For the lab sessions, all lab systems are already pre-installed with PostgreSQL.** If you are prompted for a password while connecting to the server, use:

   admin
2. After installation, the next step is to **create a database**, then create tables and populate them with data. Detailed step-by-step instructions (with images) are provided in the file uploaded on the **Learning Mall course folder**. Kindly refer to that document if you face any confusion.

Quote

"Behind every piece of data, there's a story waiting to be discovered."

1 Part A - Basic Data Retrieval

Basic data retrieval is the foundation of SQL. It helps us extract the required information from a database by selecting specific columns, applying conditions, and combining filters.

Part A

In this part, students will learn how to **retrieve data** from tables using SQL. Key concepts covered include:

- Selecting **all columns** or **specific columns** with `SELECT`.
- Filtering rows using the `WHERE` clause.
- Combining conditions with `AND` / `OR`.
- Practicing queries with **fill-in-the-blank exercises** and **challenge questions** to reinforce understanding.

1.1 Select All Data

Example: Retrieve all data from the `students` table.

```
SELECT *
FROM students;
```

This query returns all rows and all columns from the table.

1.2 Select Specific Columns

Example: Display only student names and majors.

```sql
SELECT name , major
FROM students;
```

1.3 Filtering Data with WHERE

Example: Find all students older than 21.

```sql
SELECT name , age
FROM students
WHERE age > 21;
```

1.4 Using AND / OR in Conditions

Example: Find students who are majoring in `Computer Sci` **and** are 20 years old.

```sql
SELECT name , age , major
FROM students
WHERE major = 'Computer Sci' AND age = 20;
```

Example: Find students who are in either `AI` **or** `Math`.

```sql
SELECT name , major
FROM students
WHERE major = 'AI' OR major = 'Math';
```

1.5 Practice (Fill in the Blanks)

Complete the missing parts of the queries.

1. Show all courses from the `courses` table.

```sql
    SELECT ___
    FROM courses;
```

2. Display the names of students who are 20 years old.

```sql
    SELECT name
    FROM students
    WHERE ___ = 20;
```

3. Find student names and majors where the major is `Data Science`.

```sql
    SELECT name , ___
    FROM students
    WHERE major = 'Data Science';
```

4. Show all enrollments for student with `student_id = 1`.

```sql
    SELECT *
    FROM enrollments
    WHERE ___ = 1;
```

1.6 Challenge / Practice Questions of Part A

1. ***Retrieve the names of students who are NOT studying Computer Science.***
2. ***Display all courses that have more than 2 credits.***

2 Part B - Sorting and Limiting

Sorting and limiting in SQL allow us to organize query results in a meaningful way and control how many rows are displayed. This is especially useful when working with large datasets to quickly find the most relevant information.

Part B

By the end of this part, students will be able to:
- Sort query results using `ORDER BY`.
- Sort in ascending (`ASC`) or descending (`DESC`) order.
- Limit the number of rows returned using `LIMIT`.
- Combine sorting and limiting for focused results.

2.1 Examples

a) Sorting by one column (ascending by default)

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

b) Sorting in descending order

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

c) Sorting by multiple columns

```sql
SELECT name, major, age
FROM students
ORDER BY major ASC, age DESC;
```

d) Limiting the number of rows

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

e) Combining `ORDER BY` and `LIMIT`

```
SELECT name, grade
FROM enrollments
ORDER BY grade DESC
LIMIT 3;
```

2.2 Practice / Fill-in-the-Blanks

1. Show the first 10 students sorted by name:

```
    SELECT name, age
    FROM students
    ORDER BY ___
    LIMIT ___;
```

2. Display students sorted by major (descending) and then by age (ascending):

```
    SELECT name, major, age
    FROM students
    ORDER BY ___ DESC, ___ ASC;
```

3. Find the top 5 grades in the `enrollments` table:

```
    SELECT student_id, grade
    FROM enrollments
    ORDER BY grade ___
    LIMIT ___;
```

2.3 Challenge / Practice Questions of Part B

1. ***Retrieve the 3 youngest students in AI major.***
2. ***Show courses with the highest credits first, limiting to top 5.***

3 Part C - Removing Duplicates

Sometimes tables contain repeated values. The `DISTINCT` keyword helps us remove duplicates so we only see unique results, making the data clearer and easier to analyze.

Part C

By the end of this part, students will be able to:
- Retrieve **unique values** from a table using `DISTINCT`.
- Understand the difference between **all rows** and **distinct rows**.
- Apply `DISTINCT` on single or multiple columns.

3.1 Examples

a) Distinct on a single column

```
SELECT DISTINCT major
FROM students;
```

This returns all unique majors in the students table.

b) Distinct on multiple columns

```
SELECT DISTINCT major, age
FROM students;
```

This returns unique combinations of major and age.

c) Counting unique values

```
SELECT COUNT(DISTINCT major)
FROM students;
```

This counts how many different majors exist.

3.2 Practice / Fill-in-the-Blanks

1. Show all unique course names:

```
    SELECT DISTINCT ___
    FROM courses;
```

2. Display unique combinations of student age and major:

```
    SELECT DISTINCT age, ___
    FROM students;
```

3. Count the number of different grades in enrollments:

```
    SELECT COUNT(DISTINCT ___)
    FROM enrollments;
```

3.3 Challenge / Practice Question(s)

1. ***Find how many students are in each unique major (combine `DISTINCT` with `COUNT`).***

4 Part D - Aggregate Functions

Aggregate functions allow us to perform calculations on groups of data, such as counting rows, finding averages, or identifying the smallest and largest values. They help transform raw data into useful summaries.

Part D

By the end of this part, students will be able to:
- Use SQL **aggregate functions** to summarize data.
- Work with functions such as `COUNT`, `SUM`, `AVG`, `MIN`, and `MAX`.
- Apply aggregate functions with or without `GROUP BY` to get meaningful summaries.

a) Count rows in a table

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

Counts the total number of students.

b) Sum of numeric values

```
SELECT SUM(credits)
FROM courses;
```

Adds up all course credits.

c) Average value

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

Calculates the average age of students.

d) Minimum and Maximum

```
SELECT MIN(age), MAX(age)
FROM students;
```

Finds the youngest and oldest student.

e) Using GROUP BY

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

Counts how many students are in each major.

f) Combining GROUP BY and aggregate functions

```
SELECT student_id, AVG(age)
FROM students
GROUP BY major;
```

Calculates the average age of students for each major.

4.1 Practice / Fill-in-the-Blanks

1. Count the number of courses:

```
    SELECT COUNT(___)
    FROM courses;
```

2. Find the total credits of all courses:

```
SELECT SUM(___)
FROM courses;
```

3. Calculate the average age of students:

```
SELECT AVG(___)
FROM students;
```

4. Find the minimum and maximum age of students:

```
SELECT MIN(___), MAX(___)
FROM students;
```

5. Count the number of students in each major:

```
SELECT major, COUNT(___)
FROM students
GROUP BY ___;
```

4.2 Challenge / Practice Question(s)

1. ***Display the maximum credits among all courses.***

5 Part E - Grouping Data

Grouping in SQL allows us to organize rows that share the same values into groups. When combined with aggregate functions, it helps summarize data for each group, such as finding the average grade per course or counting students in each major.

Part E

By the end of this part, students will be able to:
- Group rows based on column values using `GROUP BY`.
- Summarize data for each group using aggregate functions.
- Combine `GROUP BY` with `ORDER BY` to organize results.

a) Group students by major and count them

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

Counts the number of students in each major.

b) Average age of students by major

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

Calculates the average age of students for each major.

c) Sum of credits grouped by course

```
SELECT course_name, SUM(credits)
FROM courses
GROUP BY course_name;
```

Finds total credits for each course (useful when a course has multiple sections).

d) Group with HAVING to filter aggregated results

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

Shows only majors with more than 5 students.

e) Combining GROUP BY and ORDER BY

```
SELECT major, COUNT(*)
FROM students
GROUP BY major
ORDER BY COUNT(*) DESC;
```

Lists majors by the number of students in descending order.

5.1 Practice / Fill-in-the-Blanks

1. Count students in each major:

```
    SELECT major, COUNT(___)
    FROM students
    GROUP BY ___;
```

2. Find average grade per course:

```
    SELECT course_id, AVG(___)
    FROM enrollments
    GROUP BY ___;
```

3. Display courses with total credits greater than 3:

```
    SELECT course_name, SUM(credits)
    FROM courses
    GROUP BY ___
    HAVING SUM(credits) > ___;
```

5.2 Challenge / Practice Questions

1. ***List the number of students in each age group.***
2. ***Find majors with less than 3 students.***

6 Part F - NULL Processing

In SQL, NULL represents missing or unknown data. It is not the same as zero or an empty string. Learning how to check and handle NULL values ensures accurate query results and prevents errors in calculations.

Part F

By the end of this part, students will be able to:

- Understand what **NULL** values represent in a database.
- Identify NULL values using `IS NULL` and `IS NOT NULL`.
- Handle NULL values in queries with aggregate functions.
- Avoid errors in calculations caused by NULL values.

6.1 Examples

a) Select rows with NULL values

```sql
SELECT name, age, major
FROM students
WHERE major IS NULL;
```

Finds all students whose major is not defined.

b) Select rows without NULL values

```sql
SELECT name, age, major
FROM students
WHERE major IS NOT NULL;
```

Finds students with a defined major.

c) Counting NULLs

```sql
SELECT COUNT(*)
FROM students
WHERE major IS NULL;
```

Counts how many students have no major.

d) Using COALESCE to handle NULLs

```sql
SELECT name, COALESCE(major, 'Undeclared') AS major
FROM students;
```

Replaces NULL with a default value ('Undeclared') when displaying data.

e) NULL with aggregate functions

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

Aggregate functions ignore NULLs by default, so only non-NULL ages are considered.

6.2 Practice / Fill-in-the-Blanks

1. Find all students with no grade in enrollments:

```
SELECT *
FROM enrollments
WHERE grade ___;
```

2. Display students, replacing NULL majors with 'Unknown':

```
SELECT name, COALESCE(___, 'Unknown') AS major
FROM students;
```

3. Count the number of students without a major:

```
SELECT COUNT(*)
FROM students
WHERE major ___;
```

6.3 Challenge / Practice Question(s)

1. ***Display all courses, showing 'No Credits' for NULL values in credits.***

Quote

"Data is the new oil, and SQL is the key to unlock its value."