

COMP 7990

***Principles and Practices of
Data Analytics***

Data Management - II

Dr. Zhang Lu/Dr. Jin Yuchen/Dr. Kevin Wang

Data Retrieval: Queries

- This topic is considered very important in SQL
- Most applicable skills in your workplace [database related]

SQL for Data Retrieval: Queries

- SELECT is the best known SQL statement
- SELECT will retrieve information from the database that matches the specified criteria using SELECT/FROM/WHERE framework

SELECT
FROM
WHERE

empName
Employee
empId = 29;

Sheldon

SELECT
FROM

empName
Employee;

Sheldon

Penny

Sheldon

Employee

<u>empId</u>	empName
10	Dan
21	Penny
29	Sheldon

SQL for Data Retrieval: The result is a relation

- A query pulls information from one or more relations and creates (temporarily) a new relation.
- This allows a query to:
 - Create a relation
 - Feed information into another query (as a subquery)

SQL for Data Retrieval: Displaying Multiple Columns

- To show values for two or more specific columns, use a comma-separated list of column names

SELECT
FROM
WHERE

empId, empName
Employee
empId = 29;

29	Sheldon
----	---------

SELECT
FROM

empId, empName
Employee;

10	Dan
21	Penny
29	Sheldon

Employee

<u>empId</u>	empName
10	Dan
21	Penny
29	Sheldon

SQL for Data Retrieval: Displaying All Columns

- To show all of the columns values from the rows that match the specified criteria, use a star/asterisk (*)

```
SELECT  
FROM  
WHERE
```

```
*  
Employee  
empId = 29;
```

29	Sheldon
----	---------

```
SELECT  
FROM
```

```
*  
Employee;
```

10	Dan
21	Penny
29	Sheldon

Employee

<u>empId</u>	empName
10	Dan
21	Penny
29	Sheldon

SQL for Data Retrieval: Showing Each Row only Once

- The DISTINCT keyword may be added to the SELECT statement to suppress the display of duplicate rows

```
SELECT DISTINCT  
FROM
```

```
empName  
Employee;
```

Dan
Sheldon

Employee

<u>empId</u>	empName
10	Dan
15	Dan
29	Sheldon

```
SELECT DISTINCT  
FROM
```

```
empId, empName  
Employee;
```

10	Dan
15	Dan
29	Sheldon

SQL for Data Retrieval: Showing Search Criteria

- The WHERE clause specifies the matching or filtering criteria for the records (rows) that are to be displayed

```
SELECT      empId, empName  
FROM        Employee  
WHERE       empId = 29;
```

29	Sheldon
----	---------

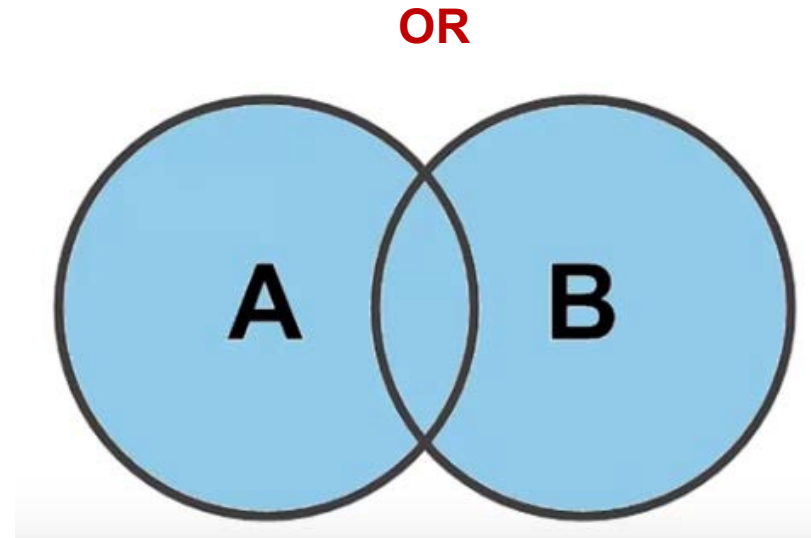
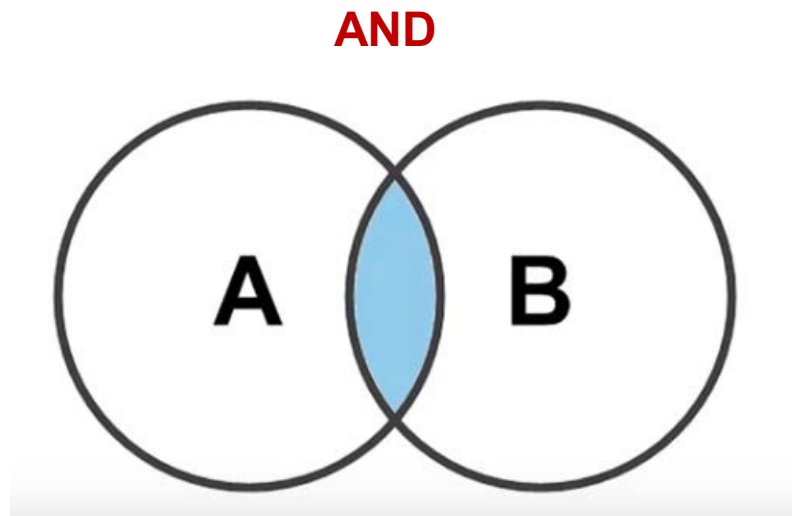
Employee	
<u>empld</u>	empName
10	Dan
21	Penny
29	Sheldon

SQL for Data Retrieval: Match Criteria

- WHERE clause comparisons may include
 - Equals “=”
 - Not Equals “<>” or “!= ”
 - Greater than “>”
 - Less than “<”
 - Greater than or Equal to “>= ”
 - Less than or Equal to “<= ”

SQL for Data Retrieval: Match Operators

- Multiple matching criteria may be specified using
 - AND
 - Representing an intersection of the data sets
 - OR
 - Representing a union of the data sets
 - “intersection” and “union” are derived from relational algebra



SQL for Data Retrieval: AND/OR examples

```
SELECT empId, empName
FROM Employee
WHERE empId < 7 OR empId > 12;
```

5	Dan
19	Sheldon

```
SELECT empId, empName
FROM Employee
WHERE deptId = 9 AND salaryCode <= 3;
```

5	Dan
---	-----

Employee

<u>empId</u>	empName	salaryCode	deptId
5	Dan	3	9
9	Penny	4	9
19	Sheldon	5	10

Quick question

How can I select the empId of "Dan" from deptId "9"?

How can I select all info about employees from all department except the department with deptId "9"?

Employee

<u>empId</u>	empName	salaryCode	deptId
5	Dan	3	9
9	Penny	4	9
19	Sheldon	5	10

SQL for Data Retrieval: A List of Values

- The WHERE clause may include the **IN** keyword to specify that a particular column value must match one of the values in a list
 - This is much more convenient than using a series of “OR” operators

```
SELECT      empName  
FROM        Employee  
WHERE       deptId IN (4, 8, 9);
```

```
SELECT      empName  
FROM        Employee  
WHERE       deptId =4 OR deptId = 8 OR deptID = 9;
```

SQL for Data Retrieval: The Logical NOT Operator

- Any criteria statements can be preceded by a NOT operator in order to invert the results
 - Using NOT will return all information except the information matching the specified criteria

```
SELECT  
FROM  
WHERE
```

```
empName  
Employee  
deptId NOT IN (4, 8, 9);
```

Dan

Employee

<u>empId</u>	empName	salaryCode	deptId
5	Dan	3	2
9	Penny	4	9
19	Sheldon	5	8

SQL for Data Retrieval: Finding Data in a Range of Values

- SQL provides a BETWEEN keyword that allows a user to specify a minimum and maximum value on one line
 - BETWEEN is inclusive

```
SELECT      empName
FROM        Employee
WHERE       salaryCode BETWEEN 10 AND 45
```

Dan

Sheldon

```
SELECT      empName
FROM        Employee
WHERE       salaryCode >= 10 AND salaryCode <= 45
```

Employee

<u>empld</u>	empName	salaryCode
5	Dan	10
9	Penny	6
19	Sheldon	30

SQL for Data Retrieval: Wildcard Searches

- The SQL **LIKE** keyword allows for searches on partial data values
- **LIKE** can be paired with wildcards to find rows that partially match a string value
 - The multiple character wildcard character is a percent sign (%)
 - The single character wildcard character is an underscore (_)

```
SELECT  
FROM  
WHERE
```

```
empId  
Employee  
empName LIKE 'Da%';
```

18

29

```
SELECT  
FROM  
WHERE
```

```
empId  
Employee  
phone LIKE '3411_ _ _ _';
```

29

Employee

<u>empId</u>	empName	phone
18	Dan	69981245
29	Danny	34111001

Quick Question

- How to select empName of the employees who are in deptID 10 with the phone numbers end with 01
- How to select empName of the employees whose the phone numbers **do not** end with 01

Employee

<u>empID</u>	empName	phoneNum	deptId
10	Dan	33330001	10
9	Penny	32340001	9
11	Sheldon	32350110	10

SQL for Data Retrieval: Sorting the Result

- Querying results may be sorted using the ORDER BY clause
 - Ascending vs. descending sorts

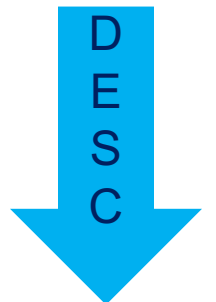
```
SELECT *  
FROM Employee  
ORDER BY empName;
```

<u>empld</u>	empName
1	Dan
2	Penny
3	Sheldon



```
SELECT *  
FROM Employee  
ORDER BY empName ASC;
```

<u>empld</u>	empName
3	Sheldon
2	Penny
1	Dan



```
SELECT *  
FROM Employee  
ORDER BY empName DESC;
```

Quick Question

- How to select employees from deptId = 10 sorted by their name?

Employee

<u>empID</u>	empName	phoneNum	deptId
10	Dan	33330001	10
9	Penny	32340001	9
11	Sheldon	32350110	10

SQL for Data Retrieval: Built-in Numeric Functions

- SQL provides several built-in numeric functions
 - COUNT
 - Counts the number of rows that match the specified criteria
 - MIN
 - Finds the minimum value for a specific columns for those rows matching the criteria
 - MAX
 - Finds the maximum value for a specific column for those rows matching the criteria
 - SUM
 - Calculates the sum for a specific column for those rows matching the criteria
 - AVG
 - Calculates the numerical average (mean) for a specific column for those rows matching the criteria
 - STDEV
 - Calculate the standard deviation of the values in a numeric column whose rows match the criteria

SQL for Data Retrieval: Built-in Function Examples

```
SELECT  
FROM
```

```
COUNT(*)  
Employee;
```

3

```
SELECT
```

```
MIN(hours) AS minimumHours,  
MAX(hours) AS maximumHours,  
AVG(hours) AS averageHours
```

```
FROM  
WHERE
```

```
Project  
ProjectId > 7;
```

Employee

<u>empld</u>	empName
10	Dan
15	Penny
29	Sheldon

minimumHours	maximumHours	averageHours
2	8	5

Quick Question

- How to get the total number of employee in deptID 10?
- How to get the average salary in all departments other than deptID 10?

Employee

<u>empID</u>	empName	salary	deptId
10	Dan	33330	10
9	Penny	32340	9
12	Sheldon	32350	10

SQL for Data Retrieval: Providing Subtotals: GROUP BY

- Categorized results can be retrieved using GROUP BY clause

```
SELECT  
FROM  
GROUP BY
```

```
deptId, COUNT(*) as numberOfEmployees  
Employee  
deptId;
```

Alias



deptId	numberOfEmployees
1	14
2	18
3	25
4	12

GROUP BY

- Categorized results can be retrieved using GROUP BY clause

programme	studentName	mark
BBA	Ada	28
BBA	Bob	69
BCDA	Carol	23
COMM	Dave	10
COMM	Eva	54
CS	Fred	5
CS	Gina	56
CS	Hulk	82
CS	Ian	5
CS	Jan	17
CS	Kevin	46
CS	Leo	24
Exchange	Mav	21
JOUR	Nancy	65
JOUR	Olivia	74
MATH	Peter	46
SCI	Queenie	47
SCI	Robert	25
SCI	Step	64
BBA	Timmy	89
CS	Ulsa	43
BSocSc	Victor	18
CS	Wendy	50
BSocSc	Xyler	41
BCDA	Yuki	65
CS	Zoe	51

Row Labels	Sum of Mark
BBA	186
Ada	28
Bob	69
Timmy	89
BCDA	88
Carol	23
Yuki	65
BSocSc	59
Victor	18
Xyler	41
COMM	64
Dave	10
Eva	54
CS	379
Jan	17
Fred	5
Gina	56
Hulk	82
Ian	5
Kevin	46
Leo	24
Ulsa	43
Wendy	50
Zoe	51
Exchange	21
Mav	21
JOUR	139
Nancy	65
Olivia	74
MATH	46
Peter	46
SCI	136
Queenie	47

```
SELECT  programme, sum(mark) as Mark
FROM    students
GROUP BY programme
```

programme	Mark
BBA	186
BCDA	88
BSocSc	59
COMM	64
CS	379
Exchange	21
JOUR	139

SQL for Data Retrieval: Providing Subtotals: GROUP BY with HAVING

- The **HAVING** clause may optionally be used with a GROUP BY in order to filter which categories are displayed.

```
SELECT      deptId, COUNT(*) as numberOfEmployees
FROM        Employee
GROUP BY    deptId
HAVING      numberOfEmployees >= 15;
```

deptId	numberOfEmployees
2	18
3	25

Difference between WHERE and HAVING

- WHERE is used to filter the original data. After applying WHERE, the filtered data will be displayed and further processed
 - WHERE do not apply on aggregated value (SUM, AVG, COUNT, etc..)
-
- HAVING is used to filter the GROUP result. After result are grouped, HAVING filter the groups that does not meet the requirement
 - HAVING always apply on aggregated value (SUM, AVG, COUNT, etc..)

Quick Question - Should I use WHERE or HAVING?

- If I want to find all deptID such that those department have more than 5 employees?

- If I want to find number of employees of each department such that those employee have more than 20000 salary?

Employee

<u>empld</u>	empName	salary	deptID
5	Dan	10000	5
9	Penny	65000	5
19	Sheldon	300000	7

Quick Question - Should I use WHERE or HAVING?

- If I want to list all deptID so that those department have more than 5 high-paid employees (high-paid means more than 50000 salary)?

- If I want to know the highest salary from each small department (< 5 people)?

Employee

<u>empld</u>	empName	salary	deptID
5	Dan	10000	5
9	Penny	65000	5
19	Sheldon	300000	7

SQL for Data Retrieval: Retrieving Information from Multiple Tables

- Subqueries

- As stated earlier, the result of a query is a relation. The result from one query may therefore be used as input for another query. This is called **subquery**.
 - Noncorrelated subqueries
 - Correlated subqueries

SQL for Data Retrieval: **Noncorrelated** Subquery Example

- In a noncorrelated subquery, the inner query only needs to run once in order for the database engine to solve the problem

```
SELECT      empName
FROM      Employee
WHERE      deptId IN
            (SELECT      deptID
             FROM      Department
             WHERE      deptName LIKE 'Account%');
```

Department		Employee		
<u>deptId</u>	deptName	<u>empld</u>	empName	deptId
8	AccountFan	5	Dan	2
9	Comp	9	Penny	9
		19	Sheldon	8


8

Sheldon

SQL for Data Retrieval: **Correlated** Subquery Example

- In a correlated subquery, the inner query needs to be run repeatedly in order for the database engine to solve the problem.
 - The inner query needs a value from the outer query in order to run.

```
SELECT      empName
FROM        Employee e
WHERE       empSalary >
            (SELECT      AVG(empSalary)
             FROM        Employee
             WHERE       deptId = e.deptId);
```



Alias

SQL for Data Retrieval: Correlated Subquery Example

empId	empName	empSalary	deptId
1	Dan	20000	5
2	Raj	50000	2
3	Sheldon	110000	5
4	Penny	90000	3
5	Bernad	120000	5

SELECT
FROM
WHERE

empName
Employee e
empSalary >
(SELECT
FROM
WHERE

AVG(empSalary)
Employee
deptId = e.deptId);

SQL for Data Retrieval: Correlated Subquery Example

<u>empId</u>	empName	empSalary	deptId
1	Dan	20000	5
2	Raj	50000	2
3	Sheldon	110000	5
4	Penny	90000	3
5	Bernad	120000	5

SELECT
FROM
WHERE

empName
Employee e
empSalary >

(SELECT
FROM
WHERE

AVG(empSalary)
Employee
deptId = e.deptId);

SQL for Data Retrieval: Retrieving Information from Multiple Tables

- Join
 - Another way of combining data from multiple tables is by using a *join*
- Different Types of Joins
 - Outer Join
 - Left outer join
 - Full outer join
 - Right outer join
 - Inner Join

SQL for Data Retrieval: WHERE Clause Inner Join Example

SELECT
FROM
WHERE

empName, deptName
Employee AS E, Department AS D
E.deptId = D.deptId;

Employee

<u>empId</u>	empName	deptId
1	Dan	102
2	Penny	101
3	Sheldon	

Department

<u>deptId</u>	deptName
101	Research
102	Customer Service

Inner Join



empName	deptName
Dan	Customer Service
Penny	Research

Inner join requires matching values from both tables.

SQL for Data Retrieval: **Join ... ON** Example

- The JOIN ... ON syntax can be also used to perform a join.
- It has the advantage of moving the JOIN syntax into the FROM clause

```
SELECT      empName, deptName
FROM        Employee e INNER JOIN Department d
           ON e.deptId = d.deptId
WHERE       d.deptName NOT LIKE 'Account%';
```

Join ... ON Example

```
SELECT  
FROM  
  
WHERE
```

```
empName, deptName  
Employee e INNER JOIN Department d  
ON e.deptId = d.deptId  
d.deptName NOT LIKE 'Account%';
```

Employee

<u>empId</u>	empName	deptId
1	Dan	102
2	Penny	101
3	Sheldon	

Department

<u>deptId</u>	deptName
101	Account Service
102	Customer Service



empName	deptName
Dan	Customer Service
Penny	Account Service



Dan	Customer Service
-----	------------------

SQL for Data Retrieval: LEFT OUTER JOIN Example

- The OUTER JOIN syntax can be used to obtain data that exists in one table without matching data in other table.

```
SELECT      empName, deptName
FROM        Employee e LEFT OUTER JOIN Department d
           ON e.deptId = d.deptId;
```

LEFT OUTER JOIN Example

Employee

<u>empId</u>	empName	deptId
1	Dan	102
2	Penny	101
3	Sheldon	

Left Outer Join



Department

<u>deptId</u>	deptName
101	Research
102	Customer Service

empName	deptName
Dan	Customer Service
Penny	Research
Sheldon	

SQL for Data Retrieval: RIGHT OUTER JOIN Example

- The unmatched data display can come from either table, depending on whether a RIGHT OUTER JOIN or LEFT OUTER JOIN is used.

```
SELECT      empName, deptName
FROM        Employee e RIGHT OUTER JOIN Department d
           ON e.deptId = d.deptId;
```


RIGHT OUTER JOIN Example

Employee

<u>empId</u>	empName	deptId
1	Dan	102
2	Penny	101
3	Sheldon	

Department

<u>deptId</u>	deptName
101	Research
102	Customer Service
103	Marketing

Right Outer Join



empName	deptName
Dan	Customer Service
Penny	Research
	Marketing

SQL for Data Retrieval: FULL OUTER JOIN Example

- The unmatched data from either table are included in the results if a FULL OUTER JOIN is used.

```
SELECT      empName, deptName  
FROM        Employee e FULL OUTER JOIN Department d  
            ON e.deptId = d.deptId;
```

FULL OUTER JOIN Example

Employee

<u>empId</u>	empName	deptId
1	Dan	102
2	Penny	101
3	Sheldon	

Department

<u>deptId</u>	deptName
101	Research
102	Customer Service
103	Marketing

Full Outer Join



empName	deptName
Dan	Customer Service
Penny	Research
Sheldon	
	Marketing

Quick Question

- Suppose deptID in Employee table now becomes a foreign key. i.e., must have a value and that value must appear in the table Department
- Which of the following table joining are equivalent?
 - a) Employee inner join Department
 - b) Employee left outer join Department
 - c) Department left outer join Employee
 - d) Employee right outer join Department
 - e) Employee full outer join Department

Department

<u>deptId</u>	deptName
101	Research
102	Customer Service
103	Marketing

Employee

<u>empId</u>	empName	deptId
1	Dan	102
2	Penny	101
3	Sheldon	103

Difficult Questions

- If I want to find the highest paid employees from each department
- If I want to find the highest paid employees from each small department (with less than 5 people)

Employee

<u>empld</u>	empName	salary	deptID
5	Dan	10000	5
9	Penny	65000	5
19	Sheldon	300000	7

Extra - Modifying Data using SQL: Deleting Database Objects: DROP

- To remove unwanted database objects from the database, use the SQL DROP statement.
- Warning: The DROP statement will permanently remove the object and all of its associated data.

```
DROP TABLE Employee;
```

Extra - Modifying Data using SQL: Removing a Constraint: ALTER & DROP

- To change the constraints on existing tables, you may need to remove the existing constraints before new constraints can be added

```
ALTER TABLE Employee DROP CONSTRAINT empFk;
```

Extra - Modifying Data using SQL: The CHECK Constraint

- The CHECK constraint can be used to create restrictions on the values that are allowed to appear in a column

```
ALTER TABLE Project
    ADD CONSTRAINT projectCheckDates
        CHECK (startDate < endDate);
```


Extra - SQL Views

- A SQL View is a virtual table created by a DBMS-stored SELECT statement which can combine access to data in multiple tables and even in other views.
- You can run a query against a view in the same way that you run a query against a table.

```
CREATE VIEW SalesDepartment AS
SELECT *
FROM Employee
WHERE deptId = (SELECT deptId from Department WHERE deptNAME = 'Sales');

SELECT empName
FROM SalesDepartment;
```

SQL Summary

- SQL for Data Definition
 - CREATE: create database objects
 - ALTER: modify the structure of existing database objects
 - DROP: delete existing database objects
- SQL for Data Manipulation
 - Change data
 - INSERT INTO: add a new row
 - UPDATE: change the data values
 - DELETE: delete rows from data
 - Retrieval Data
 - SELECT ... WHERE