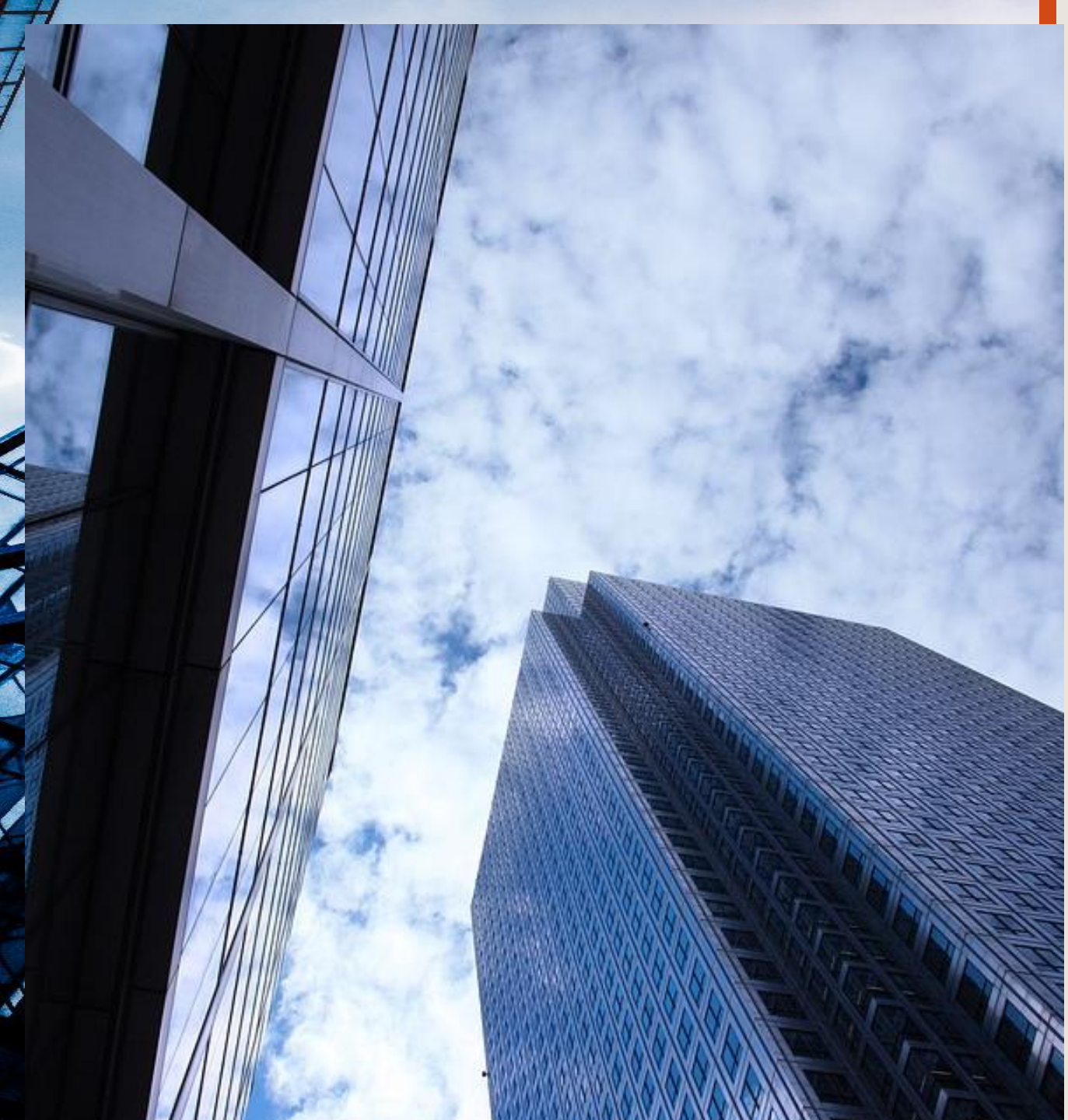


SQL
PART 03

Jayathma Chathurangani

ejc@ucsc.cmb.ac.lk

OUTLINE

DML:

- **SQL INSERT**
- **SQL UPDATE**
- **SQL DELETE**
- **SQL SELECT (Variations in SELECT, Sorting Results, Aggregate Functions, Grouping Results)**

1

INTRODUCTION

1.1 INTRODUCTION

- This section looks at the following SQL DML statements:
  - SELECT – to query data in the database (This topic Span across sections)
  - INSERT – to insert data into a table
  - UPDATE – to update data in a table
  - DELETE – to delete data from a table

Database Modification

1.2 LITERALS

- Literals are **constants** that are used in SQL statements.
- There are different forms of literals for every data type supported by SQL.
- However, for simplicity, we can distinguish between literals that are enclosed in single quotes and those that are not.
- All nonnumeric data values must be enclosed in single quotes; all numeric data values must not be enclosed in single quotes.
- For example, we could use literals to insert data into a table:

INSERT INTO PropertyForRent(propertyNo, street, city, postcode, type, rooms, rent,

ownerNo, staffNo, branchNo)

VALUES ('PA14', '16 Holhead', 'Aberdeen', 'AB7 5SU', 'House', 6, 650.00, 'CO46', 'SA9', 'B007');

EXAMPLE TABLES USED IN THIS LESSON

Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

EXAMPLE TABLES USED IN THIS LESSON (CONTINUED)

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-13	too small
CR76	PG4	20-Apr-13	too remote
CR56	PG4	26-May-13	
CR62	PA14	14-May-13	no dining room
CR56	PG36	28-Apr-13	

Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-13
CR56	B003	SG37	11-Apr-12
CR74	B003	SG37	16-Nov-11
CR62	B007	SA9	7-Mar-12

2

DATABASE UPDATE

2.1 INSERT – ADDS NEW ROWS OF DATA TO A TABLE

```
INSERT INTO TableName [(columnList)]
VALUES (dataValueList)
```

- ***Type 1*** **- INSERT . . . VALUES**

INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M', **DATE** '1957-05-25', 8300, 'B003');

- ***Type 2*** **– INSERT using defaults**

INSERT INTO Staff (staffNo, fName, lName, position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');

INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL, NULL, 8100, 'B003');

- ***Type 3*** **– INSERT . . . SELECT**

```
INSERT INTO StaffPropCount
        (SELECT s.staffNo, fName, lName, COUNT(*)
        FROM Staff s, PropertyForRent p
        WHERE s.staffNo = p.staffNo
        GROUP BY s.staffNo, fName, lName)
        UNION
        (SELECT staffNo, fName, lName, 0
        FROM Staff s
        WHERE NOT EXISTS (SELECT *
                FROM PropertyForRent p
                WHERE p.staffNo = s.staffNo));
```

2.2 UPDATE – MODIFYING DATA IN THE DATABASE

```
UPDATE TableName
SET columnName1 = dataValue1 [, columnName2 = dataValue2 . . .]
[WHERE searchCondition]
```

- ***Type 1*** **- UPDATE all rows** (Where condition omitted)

```
UPDATE Staff
SET salary = salary*1.03;
```

- ***Type 2*** **- UPDATE specific rows**

```
UPDATE Staff
SET salary = salary*1.05
WHERE position = 'Manager';
```

- ***Type 3*** **- UPDATE multiple columns**

```
UPDATE Staff
SET position = 'Manager', salary = 18000
WHERE staffNo = 'SG14';
```

2.3 DELETE – DELETING DATA IN THE DATABASE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- ***Type 1*** **- DELETE all rows** (Where condition omitted)

DELETE FROM Viewing;

- ***Type 2*** **- DELETE specific rows**

DELETE FROM Viewing
WHERE propertyNo = 'PG4';

ACTIVITY

3

SIMPLE SQL SELECT

3.1 INTRODUCTION

- The purpose of the **SELECT** statement is to retrieve and display data from one or more database tables.
- It has the below general form

SELECT [DISTINCT | ALL] {* | [columnExpression [**AS** newName]] [, . . .]}
FROM TableName [alias] [, . . .]
[WHERE condition]
[GROUP BY columnList] **[HAVING** condition]
[ORDER BY columnList]

1. SELECT specifies which columns are to appear in the output
2. FROM specifies the table or tables to be used
3. WHERE filters the rows subject to some condition (Optional)
4. GROUP BY forms groups of rows with the same column value (Optional)
5. HAVING filters the groups subject to some condition (Optional)
6. ORDER BY specifies the order of the output (Optional)

(The sequence of processing in a SELECT statement is in the above order)

3.1 INTRODUCTION (CONTINUED)

- The order of the clauses in the SELECT statement cannot be changed.
- The only two mandatory clauses are the first two: SELECT and FROM; the remainder are optional.
- The SELECT operation is closed: **the result of a query on a table is another table**.
- It is an extremely powerful command, capable of performing the equivalent of the relational algebra's Selection, Projection, and Join operations in a single statement.

3.2 SELECT CLAUSE

This section covers,

- Retrieve All Columns, All Rows
- Retrieve Specific Columns, All Rows
- Use of DISTINCT
- Calculated Fields

3.2.1 SELECT CLAUSE - RETRIEVE ALL COLUMNS, ALL ROWS

- There are no restrictions specified in this query, the WHERE clause is unnecessary and all columns are required.

Query: ***List full details of all staff.***

- ***Type 1*** (Specifying the all column names)

SELECT staffNo, fName, lName, position, sex, DOB, salary, branchNo
FROM Staff;

- ***Type 2***

SELECT *
FROM Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

3.2.2 SELECT CLAUSE - RETRIEVE SPECIFIC COLUMNS, ALL ROWS

- Note that, unless specified, the rows in the result table may not be sorted. Some DBMSs do sort the result table based on one or more columns

Query: Produce a list of salaries for all staff, showing only the staff number, the first and last names, and the salary details.

SELECT staffNo, fName, lName, salary
FROM Staff;

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

3.2.3 SELECT CLAUSE - USE OF DISTINCT

- To eliminate the duplicates, we use the **DISTINCT** keyword.

Query: List the property numbers of all properties that have been viewed.

- Without DISTINCT – Duplicates Exist

SELECT propertyNo
FROM Viewing;

propertyNo
PA14
PG4
PG4
PA14

- With DISTINCT – Duplicates Removed

SELECT DISTINCT propertyNo
FROM Viewing;

propertyNo
PA14
PG4
PG36

3.2.4 SELECT CLAUSE - CALCULATED FIELDS

- In general, to use a calculated field (sometimes called a computed or derived field), you specify an SQL expression in the SELECT list.
- An SQL expression can involve addition, subtraction, multiplication, and division, and parentheses can be used to build complex expressions.
- More than one table column can be used in a calculated column; however, the columns referenced in an arithmetic expression **must have a numeric type**.
- Normally, a column in the result table takes its name from the corresponding column of the database table from which it has been retrieved.
- However, in this case, SQL does not know how to label the column. Some dialects give the column a name corresponding to its position in the table (for example, col4); some may leave the column name blank or use the expression entered in the SELECT list.
- The ISO standard allows the column to be named using an **AS** clause.

3.2.4 SELECT CLAUSE - CALCULATED FIELDS (CONTINUED)

Query: Produce a list of monthly salaries for all staff, showing the staff number, the first and last names, and the salary details.

SELECT staffNo, fName, lName, salary/12
FROM Staff;

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00

SELECT staffNo, fName, lName, salary/12 **AS** monthlySalary
FROM Staff;

staffNo	fName	lName	monthlySalary
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00

3.3 WHERE CLAUSE

- This restrict the rows that are retrieved.
- It consists of the keyword WHERE followed by a search condition that specifies the rows to be retrieved.
- The five basic search conditions (or predicates, using the ISO terminology) which will be covered in this section are as follows:
  1. *Comparison*: Compare the value of one expression to the value of another expression.
  2. *Range Test*: whether the value of an expression falls within a specified range of values. (BETWEEN/NOT BETWEEN)
  3. *Set membership*: Test whether the value of an expression equals one of a set of values. (IN/NOT IN)
  4. *Pattern match:* Test whether a string matches a specified pattern. (LIKE/NOT LIKE)
  5. *Null Test:* whether a column has a null (unknown) value. (IS NULL/IS NOT NULL)

3.3.1 WHERE CLAUSE - COMPARISON SEARCH CONDITION

- In SQL, the following simple comparison operators are available:
  - ✓ = equals
  - ✓ <> is not equal to (ISO standard)
  - ✓ ! = is not equal to (allowed in some dialects)
  - ✓ < is less than
  - ✓ < = is less than or equal to
  - ✓ > is greater than
  - ✓ > = is greater than or equal to

- More complex predicates can be generated using the logical operators AND, OR, and NOT, with parentheses (if needed or desired) to show the order of evaluation.

  The rules for evaluating a conditional expression are:

  - ✓ an expression is evaluated left to right;
  - ✓ subexpressions in brackets are evaluated first;
  - ✓ NOTs are evaluated before ANDs and ORs;
  - ✓ ANDs are evaluated before ORs.

- The use of parentheses is always recommended, in order to remove any possible ambiguities.

3.3.1 WHERE CLAUSE - COMPARISON SEARCH CONDITION (CONTINUED)

- **Comparison**

Query: List all staff with a salary greater than £10,000.

SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000**;**

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

- **Compound Comparison**

Query: List the addresses of all branch offices in London or Glasgow.

SELECT *
FROM Branch
WHERE city = 'London' **OR** city = 'Glasgow'**;**

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

3.3.2 WHERE CLAUSE - RANGE SEARCH CONDITION (BETWEEN/NOT BETWEEN)

- Negated version of BETWEEN is NOT BETWEEN

Query: List all staff with a salary between £20,000 and £30,000.

SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary **BETWEEN** 20000 **AND** 30000**;**

- **Same can be write using comparison search**

SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary **> =** 20000 **AND** salary **< =** 30000**;**

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

3.3.3 WHERE CLAUSE - SET MEMBERSHIP SEARCH CONDITION (IN/NOT IN)

- Negated version of IN is NOT IN

Query: List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

- **Same can be write using comparison search**

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position = 'Manager' OR position = 'Supervisor';
```

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

3.3.4 WHERE CLAUSE - PATTERN MATCH SEARCH CONDITION (LIKE/NOT LIKE)

- SQL has two special pattern-matching symbols:
  - The % percent character represents any sequence of zero or more characters (wildcard).
  - The _ underscore character represents any single character.
- Example:
  - address **LIKE** '**H%**' means the first character must be H, but the rest of the string can be anything.
  - address **LIKE** '**H**_ _ _' means that there must be exactly four characters in the string, the first of which must be an H.
  - address **LIKE** '**%e**' means any sequence of characters, of length at least 1, with the last character an e.
  - address **NOT LIKE** '**H%**' means the first character cannot be an H.
- If the search string can include the pattern-matching character itself, we can use an escape character to represent the pattern-matching character.
- Example:
  - **LIKE** '**15#%**' **ESCAPE** '**#**' will check for the string '15%'.
  - **SELECT** * **FROM** emp **WHERE** ENAME **LIKE** 'JOE$_JOHN' **ESCAPE** '$'; This matches only records with name 'JOE_JOHN'.

3.3.4 WHERE CLAUSE - PATTERN MATCH SEARCH CONDITION (LIKE/NOT LIKE) (CONTINUED)

Query: Find all owners with the string 'Glasgow' in their address.

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE '%Glasgow%';
```

- (address LIKE '%Glasgow%' means a sequence of characters of any length containing *Glasgow.*)

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

3.3.5 WHERE CLAUSE - NULL SEARCH CONDITION (IS NULL/IS NOT NULL)

- **A null is considered to have an unknown value, so we cannot test whether it is equal or not equal to another string**.

Query: List the details of all viewings on property PG4 where a comment has not been supplied.

- If we tried to execute the SELECT statement using either of these compound conditions, we would get an empty result table.

(propertyNo = 'PG4' **AND** comment = ' ') or (propertyNo = 'PG4' **AND** comment < > 'too remote')

- Instead, we have to test for null explicitly using the special keyword IS NULL:

SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo **= 'PG4' AND** comment **IS NULL;**

clientNo	viewDate
CR56	26-May-13

4

SORTING RESULTS (ORDER BY CLAUSE)

4.1 INTRODUTION

- In general, the rows of an SQL query result table are not arranged in any particular order (although some DBMSs may use a default ordering based, for example, on a primary key).
- The results of a query are sorted using the ORDER BY clause in the SELECT statement.
- The ORDER BY clause consists of a list of column identifiers that the result is to be sorted on, separated by commas.
- A column identifier may be either a column name or a column number that identifies
- Column numbers could be used if the column to be sorted on is an expression and no AS clause is specified to assign the column a name that can subsequently be referenced.
- The ORDER BY clause allows the retrieved rows **to be ordered in ascending (ASC) or descending (DESC) order on any column or combination of column**s, regardless of whether that column appears in the result.
- However, some dialects insist that the ORDER BY elements appear in the SELECT list.
- ORDER BY clause must always be the last clause of the SELECT statement.

4.1 ORDER BY CLAUSE – SINGLE COLUMN ORDERING

Query: Produce a list of salaries for all staff, arranged in descending order of salary.

SELECT staffNo, fName, lName, salary
FROM Staff
ORDER BY salary **DESC;**

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

4.3 ORDER BY CLAUSE – MULITPLE COLUMN ORDERING

- The **major sort key** determines the overall order of the result table. If the values of the major sort key are unique, there is no need for additional keys to control the sort.
- If the values of the major sort key are not unique, there may be multiple rows in the result table with the same value for the major sort key.
- In this case, it may be desirable to order rows with the same value for the major sort key by some additional sort key.
- If a second element appears in the ORDER BY clause, it is called a **minor sort key**.

3.4.2 ORDER BY CLAUSE – MULTIPLE COLUMN ORDERING

Query: Produce an abbreviated list of properties arranged in order of property type.

SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type**;**

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type, rent DESC**;**

- The result is ordered first by property type, in ascending alphabetic order (ASC being the default setting), and within property type, in descending order of rent.

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

ACTIVITY

5

SQL AGGREGATE FUNCTIONS

5.1 INTRODUCTION

- As well as retrieving rows and columns from the database, we often want to perform some form of summation or aggregation of data, similar to the totals at the bottom of a report.
- The ISO standard defines five aggregate functions:
  - COUNT – returns the number of values in a specified column
  - SUM – returns the sum of the values in a specified column
  - AVG – returns the average of the values in a specified column
  - MIN – returns the smallest value in a specified column
  - MAX – returns the largest value in a specified column
- These functions operate on a single column of a table and return a single value.
- **COUNT, MIN, and MAX apply to both numeric and nonnumeric fields** (Considering Lexicographic order), but SUM and AVG may be used on numeric fields only.

5.1 INTRODUCTION (CONTINUED)

- **Apart from COUNT(*), each function eliminates nulls first and operates only on the remaining nonnull values.**
- COUNT(*) is a special use of COUNT that counts all the rows of a table, regardless of whether nulls or duplicate values occur.
- It is important to note that an aggregate function can be used only in the SELECT list and in the HAVING clause
- If the **SELECT list includes an aggregate function and no GROUP BY clause is being used to group data together, then no item in the SELECT list can include any reference to a column unless that column is the argument to an aggregate function.**
- *Example:* Below query is invalid because it does not have a GROUP BY clause and the column staffNo in the SELECT list is used outside an aggregate function.

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

5.2 AGGREGATE FUNCTIONS - USE OF COUNT(*) AND COUNT(DISTINCT)

- The total number of properties satisfying this condition can then be found by applying the aggregate function COUNT.

Query: How many properties cost more than £350 per month to rent?

```
SELECT COUNT(*) AS myCount
FROM PropertyForRent
WHERE rent > 350;
```

myCount
5

- As the same property may be viewed many times, we have to use the DISTINCT keyword to eliminate duplicate properties.

Query: How many different properties were viewed in May 2013?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-13' AND '31-May-13';
```

myCount
2

5.2 AGGREGATE FUNCTIONS - USE OF COUNT AND SUM

- The number of Managers and the sum of their salaries can be found by applying the COUNT and the SUM functions respectively.

Query: Find the total number of Managers and the sum of their salaries.

```
SELECT COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
WHERE position = 'Manager';
```

myCount	mySum
2	54000.00

5.3 AGGREGATE FUNCTIONS - USE OF MIN, MAX, AVG

- The required values can be calculated using the MIN, MAX, and AVG functions based on the salary column.

Query: Find the minimum, maximum, and average staff salary

```
SELECT MIN(salary) AS myMin, MAX(salary) AS myMax, AVG(salary) AS myAvg
FROM Staff;
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00

6

GROUPING RESULTS (GROUP BY CLAUSE)

6.1 INTRODUCTION

- A query that includes the GROUP BY clause is called a **grouped query**, because it groups the data from the SELECT table(s) and produces a single summary row for each group.
- The columns named in the GROUP BY clause are called the **grouping columns**.
- The ISO standard requires the SELECT clause and the GROUP BY clause to be closely integrated.
- When GROUP BY is used, each item in the SELECT list must be **single-valued per group**.
- In addition, the SELECT clause may contain only:
  - column names;
  - aggregate functions;
  - constants;
  - an expression involving combinations of these elements.

6.1 INTRODUCTION (CONTINUED)

- All column names in the SELECT list must appear in the GROUP BY clause unless the name is used only in an aggregate function.
- There may be column names in the GROUP BY clause that do not appear in the SELECT list.
- When the WHERE clause is used with GROUP BY, the WHERE clause is applied first, then groups are formed from the remaining rows that satisfy the search condition.
- The ISO standard considers two nulls to be equal for purposes of the GROUP BY clause. If two rows have nulls in the same grouping columns and identical values in all the nonnull grouping columns, they are combined into the same group.

6.2 GROUPING RESULTS – GROUP BY CLAUSE

Query: Find the number of staff working in each branch and the sum of their salaries.

```
SELECT branchNo, COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

- Below **Nested Query** will results the same.

```
SELECT branchNo, (SELECT COUNT(staffNo) AS myCount
                  FROM Staff s
                  WHERE s.branchNo = b.branch
                  (SELECT SUM(salary) AS mySum
                  FROM Staff s
                  WHERE s.branchNo = b.branchNo)
FROM Branch b
ORDER BY branchNo;
```

branchNo	staffNo	salary
B003	SG37	12000.00
B003	SG14	18000.00
B003	SG5	24000.00
B005	SL21	30000.00
B005	SL41	9000.00
B007	SA9	9000.00

COUNT(staffNo)	SUM(salary)
3	54000.00
2	39000.00
1	9000.00

6.3 GROUPING RESULTS – RESTRICTING GROUPINGS (HAVING CLAUSE)

- The HAVING clause is designed for use with the GROUP BY clause to restrict the groups that appear in the final result table.
- Although similar in syntax, HAVING and WHERE serve different purposes.
  - **WHERE** clause filters individual rows going into the final result table
  - **HAVING** filters groups going into the final result table.
- The ISO standard requires that column names used in the HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.
- In practice, the search condition in the HAVING clause always includes at least one aggregate function; otherwise the search condition could be moved to the WHERE clause and applied to individual rows. (Remember that aggregate functions cannot be used in the WHERE clause.)

Query: For each branch office with more than one member of staff, find the number of staff working in each branch and the sum of their salaries.

```
SELECT branchNo, COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

ACTIVITY

EXTRA

- **SQL LIMIT, TOP and FETCH FIRST**

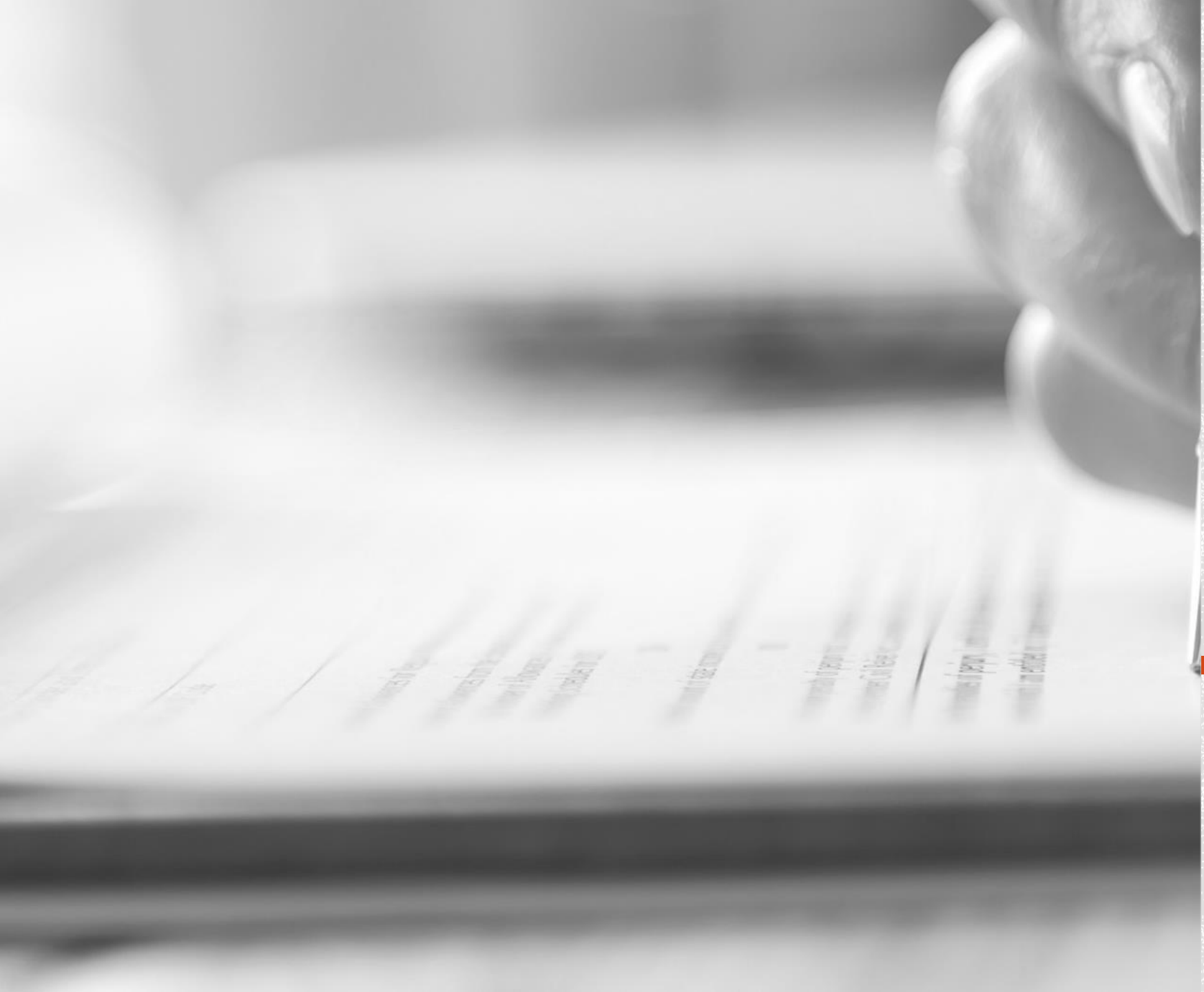
```
SELECT TOP 2 *
FROM Customers;

SELECT first_name, age
FROM Customers
LIMIT 2;

SELECT first_name, last_name
FROM Customers
LIMIT 2 OFFSET 3;

SELECT *
FROM Customers
FETCH FIRST 2 ROWS ONLY;
```

Keyword	Database System
`TOP`	SQL Server, MS Access
`LIMIT`	MySQL, PostgreSQL, SQLite
`FETCH FIRST`	Oracle



WRAP UP

THANK YOU!