SQL and Data Manipulation

- Press `Space` to navigate through the slides
- Use `Shift+Space` to go back

Quick Recap 

- **SELECT**
  - **DISTINCT**
  - **Caltulated fields**
  - **5 Basic Search Conditions**

Continuing with SQL

- **Column Ordering**
- Use **aggregate functions**
- Group data using **GROUP BY** and **HAVING**
- **Join** tables together

SINGLE COLUMN ORDERING

- List 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

- `DESC` means descending order
- `ASC` is used for ascending order

MULTIPLE COLUMN ORDERING

- To arrange in order of type and rent, specify **type** as major and specify **rent** as minor order:

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

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

SELECT STATEMENT – AGGREGATES

- **ISO** standard defines **5 aggregate functions:**
  - `COUNT` returns number of values in specified column.
  - `SUM` returns sum of values in specified column.
  - `AVG` returns average of values in specified column.
  - `MIN` returns smallest value in specified column.
  - `MAX` returns largest value in specified column.

SELECT STATEMENT – AGGREGATES

- Each operates on a **single column of a table** and **returns a single value.**
- `COUNT`, `MIN` and `MAX` apply to numeric and non-numeric fields
- `SUM` and `AVG` may be used on numeric fields only.
- Apart from `COUNT(*)`, each function eliminates nulls first and operates only on remaining non-null values.

- `COUNT(*)` counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- `DISTINCT` has no effect with `MIN`/`MAX`, but may have with `SUM`/`AVG`.

SELECT STATEMENT – AGGREGATES

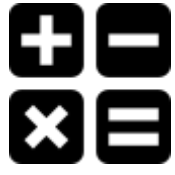
- How many properties cost more than £350 per month to rent?

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

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

myCount
5

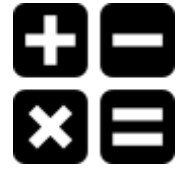
- Note the use of `AS` as an **alias**

USE OF COUNT AND SUM 

- **Find number of Managers and sum of their salaries.**

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

myCount	mySum
2	54000.00

USE OF MIN, MAX, AVG 

- **Find 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

SELECT STATEMENT REVIEW

```
SELECT [DISTINCT | ALL]

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

SELECT STATEMENT REVIEW

- `SELECT` Specifies which columns are to appear in output.
- `FROM` Specifies table(s) to be used.
- `WHERE` Filters rows.
- `GROUP BY` Forms groups of rows with same column value.
- `HAVING` Filters groups subject to some condition.
- `ORDER BY` Specifies the order of the output.

SELECT STATEMENT – GROUPING

- Use `GROUP BY` clause to get sub-totals.
- `SELECT` and `GROUP BY` closely integrated: each item in `SELECT` list must be single-valued per group and `SELECT` clause may only contain:
    - column names
    - aggregate functions
    - constants
    - expression involving combinations of the above

SELECT STATEMENT – GROUPING

- Find number of staff in each branch and their total 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

RESTRICTED GROUPING – HAVING CLAUSE

- `HAVING` clause is designed for use with `GROUP BY` to restrict groups that appear in final result table.
- Similar to `WHERE`, but `WHERE` filters individual rows whereas `HAVING` filters groups.
- Column names in `HAVING` clause must also appear in the `GROUP BY` list or be contained within an aggregate function.

USE OF HAVING CLAUSE

- For each branch with more than 1 member of staff, find number of staff in each branch and 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

MULTIPLE TABLE QUERIES 

- All the examples we have considered so far have a limitation: the columns that are to appear in the result table must all come from a single table - in many cases this is not sufficient
- To combine columns from several tables into a result table we need to use a **join operation**
- The SQL join operation combines information from two tables by forming pairs of related rows from the two tables
- To perform a join, we simply include more than one table name in the `FROM` clause, using a comma as a separator, and typically including a `WHERE` clause to specify the join column(s)
- It is also possible to use an alias for a table named in the `FROM` clause
- The alias is separated from the table name with a space
- An alias can be used to qualify a column name whenever there is ambiguity regarding the source of the column name
- It can also be used as shorthand notation for the table name

MULTIPLE TABLE QUERIES

- List names of all **clients** who have viewed a **property** along with any comment supplied.

```
SELECT c.clientNo, fName, lName, propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```

- Only those rows from both tables that have identical values in the `clientNocolumns` (**c.clientNo** = **v.clientNo**) are included in result.
- Equivalent to equi-join in relational algebra.

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG14	too remote

JOINS 

- The SQL **Joins** clause is used to combine records from two or more tables in a database.
    - A `JOIN` is a means for combining fields from two tables by using values common to each.
    - It creates a set of rows in a temporary table.

Types of SQL JOIN 

- **EQUI JOINI**
    - EQUI JOIN is a simple SQL join.
    - Uses the equal sign = as the comparison operator for the condition
- **NON EQUI JOIN**
    - NON EQUI JOIN uses comparison operator other than the equal sign.
    - The operators uses like >, <, >=, <= with the condition.

Types of SQL EQUI JOIN 

- `INNER JOIN`
    - Returns only matched rows from the participating tables.
    - Match happened only at the key record of participating tables.
- `OUTER JOIN`
    - Returns all rows from one table and
    - Matching rows from the secondary table and
    - Comparison columns should be equal in both the tables.

List of SQL JOINS 

- `INNER JOIN`
- `LEFT JOIN` OR `LEFT OUTER JOIN`
- `RIGHT JOIN` OR `RIGHT OUTER JOIN`
- `FULL OUTER JOIN`
- `NATURAL JOIN`
- `CROSS JOIN`
- `SELF JOIN`

INNER JOIN 

- The `INNER JOIN` selects all rows from both participating tables as long as there is a match between the columns.
- An SQL `INNER JOIN` is same as `JOIN` clause, combining rows from two or more tables.

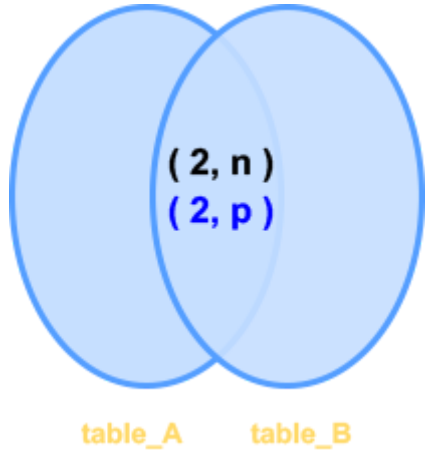
Example: INNER JOIN 

```
SELECT * FROM table1
INNER JOIN table2
ON table1.A=table2.A;
```

A	M
1	2
2	n
4	o

A	N
2	p
3	q
5	r

A	M	A	N
2	n	2	p


(2, n)
(2, p)
table_A
table_B

LEFT JOIN or LEFT OUTER JOIN 

- The SQL LEFT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.
- The unmatched rows will also be available from the table before the JOIN clause.

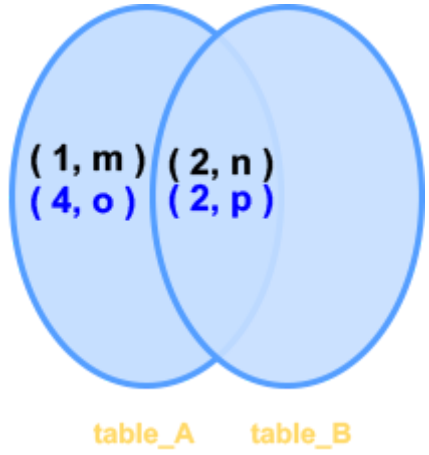
Example: LEFT JOIN or LEFT OUTER JOIN 

```
SELECT * FROM table1
LEFT JOIN table2
ON table1.A=table2.A;
```

A	M
1	m
2	n
4	o

A	N
2	p
3	q
5	r

A	M	A	N
1	m	null	null
2	n	2	p
4	o	null	null


(1, m)
(4, o)
(2, n)
(2, p)
table_A
table_B

RIGHT JOIN or RIGHT OUTER JOIN 

- The SQL RIGHT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.
- The unmatched rows will also be available from the table written after the JOIN clause.

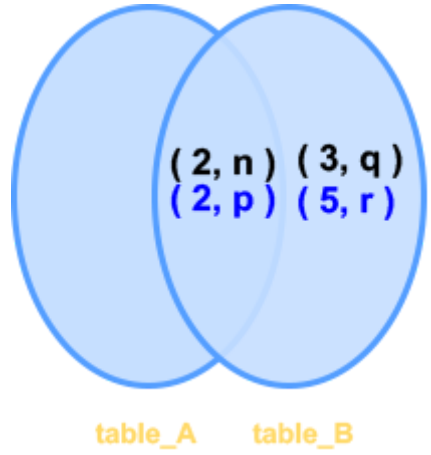
Example : RIGHT JOIN or RIGHT OUTER JOIN 

```
SELECT * FROM table1
RIGHT JOIN table2
ON table1.A=table2.A;
```

A	M
1	m
2	n
4	o

A	N
2	p
3	q
5	r

A	M	A	N
2	n	2	p
null	null	3	q
null	null	5	r


(2, n)
(2, p)
(3, q)
(5, r)
table_A
table_B

FULL OUTER JOIN

- Combines the results of both left and right outer joins.
- Returns all matched or unmatched rows.
- Includes tables on both sides of the join clause.

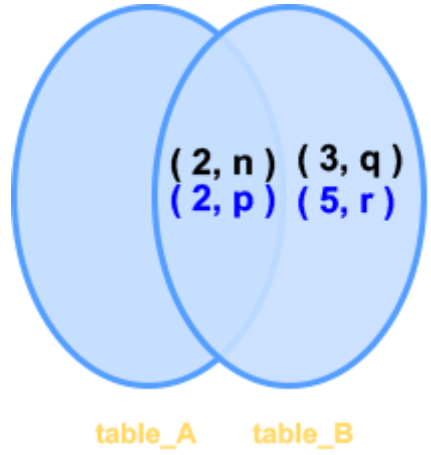
Example: FULL OUTER JOIN

```
SELECT * FROM table1
FULL OUTER JOIN table2
ON table1.A=table2.A;
```

A	M
1	2
2	n
4	o

A	N
2	p
3	q
5	r

A	M	A	N
2	n	2	p
1	m	null	null
4	o	null	null
null	null	3	q
null	null	5	r


(2, n)
(2, p)
(3, q)
(5, r)
table_A
table_B

NATURAL JOIN

- The SQL `NATURAL JOIN` is a type of `EQUI JOIN` and is structured in such a way that, columns with same name of associate tables will appear once only.
- The associated tables have one or more pairs of identically named columns.
- The columns must be the same data type.
- Don’t use ON clause in a natural join.

Example: NATURAL JOIN

```
SELECT *
FROM table1
NATURAL JOIN table2;
```

A	M
1	2
2	n
4	o

A	N
2	p
3	q
5	r

A	M	N
2	n	p

CROSS JOIN

- The SQL `CROSS JOIN` produces a result set which is the number of rows in the first table multiplied by the number of rows in the second table, if no WHERE clause is used along with `CROSS JOIN`.
- This kind of result is called as Cartesian Product.
- If, `WHERE` clause is used with `CROSS JOIN`, it functions like an `INNER JOIN`.

Example: CROSS JOIN

```
SELECT *
FROM table1
CROSS JOIN table2;
```

A	M
1	m
2	n
4	o

A	N
2	p
3	q
5	r

A	M	A	N
1	m	2	p
2	n	2	p
4	o	2	p
1	m	3	q
2	n	3	q
4	o	3	q
1	m	5	r
2	n	5	r
4	o	5	r

SELF JOIN

- A self join is a join in which a table is joined with itself (Unary relationships), specially when the table has a FOREIGN KEY which references its own PRIMARY KEY.
- To join a table itself means that each row of the table is combined with itself and with every other row of the table.
- The self join can be viewed as a join of two copies of the same table.

Example: SELF JOIN

```
SELECT *
FROM table1 X, table1 Y
WHERE X.A=Y.A;
```

A	M
1	m
2	n
4	o

A	M	A	N
1	m	1	m
2	n	2	n
4	o	4	o

A	M	A	M
1	m	1	m
2	n	2	n
4	o	4	o

Summary 

- `ORDER BY`
- Use **aggregate functions**
    - `COUNT` returns number of values in specified column.
    - `SUM` returns sum of values in specified column.
    - `AVG` returns average of values in specified column.
    - `MIN` returns smallest value in specified column.
    - `MAX` returns largest value in specified column.
- Group data using `GROUP BY` and `HAVING`
- `JOIN` tables together

Tutorial 

- SQLZOO (https://sqlzoo.net)
    - SUM & COUNT (https://sqlzoo.net/wiki/SUM_and_COUNT)
    - JOIN (https://sqlzoo.net/wiki/The_JOIN_operation)
    - More JOIN Operations (https://sqlzoo.net/wiki/More_JOIN_operations)
    - Self JOIN (https://sqlzoo.net/wiki/Self_join)