X THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/

SQL

Course Content

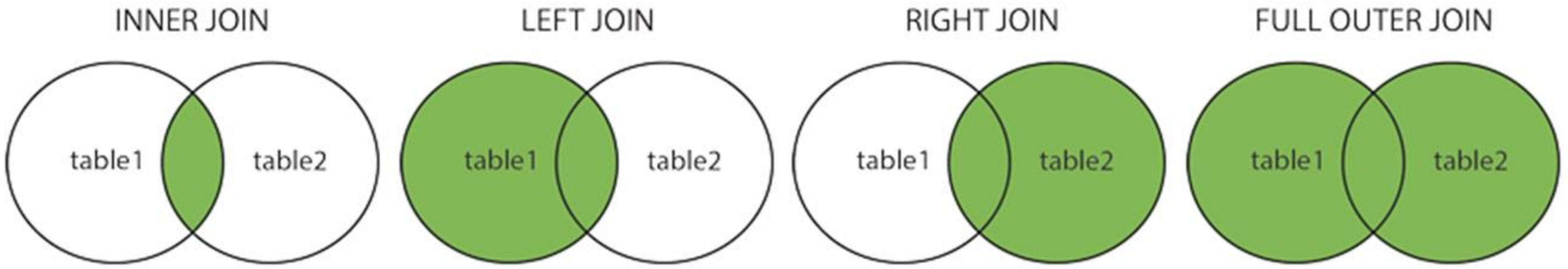
1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

To do...

- Quiz 1 closes - 5pm today
- Class Work 1 - Wed, 12 February 2025, 12:00 PM

Joins

- The complex sequence of operations on the previous two slides was necessary to produce a *sensible* combination of rows in the result.
- This sequence follows a pattern that is so frequently used that a special operation called *join* has been defined as a shortcut.
- The *join* operator is basically a combination of a Cartesian product and a restriction operation.
- There are a number of variations, some more useful than others.
  – *Natural-join, Equi-join, Inner-join, Outer-join, Self-join*



Natural joins

- Combines rows from two tables based on columns with the **same name** and **data type.**
- The natural join operator is a combination of Cartesian Product, Restriction, and Projection.
- We denote a natural join on two relations R and S as follows:

$$R \bowtie S$$

- In effect, the natural join operator does the following:
  - It first forms the Cartesian Product of R and S
  - It then **Restricts** the result to one in which common attributes from R and S have the *same* value. (Usually, the common attributes are in fact primary and foreign keys.)
  - Finally, it applies the **Projection** operator so that each of the common attributes from R and S appears only *once* in the final result.

Natural Join Example

- Repeating the previous example using a natural join gives the following result.

RENTER $\bowtie$ VIEWING

Rno	***Name***	***Address***	***Pno***	***Date***	***Time***	***Comment***
CR76	John Kay	56 High St	PA14	21/02/97	11:15	no dining room
CR74	Mike Ritchie	18 Tain St	PA14	21/02/97	09:00	too small
CR74	Mike Ritchie	18 Tain St	PG21	15/06/97	03:45	
CR62	Mary Tregear	5 Tarbot Rd	PL94	18/08/97	09:00	too remote

- The common attribute is the *Rno* field.
  - It is the primary key of the *RENTER* relation.
  - It is a foreign key in the *VIEWING* relation
- The *Rno* field occurs only once in the result.

Natural Join Example

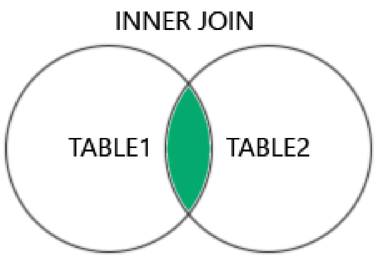
- Automatically joins tables on columns with the same name and datatype.
- No need to specify the ON condition.
- Can lead to unexpected joins if tables have multiple common column names.
- If no common columns exist, it returns a Cartesian product.

```
SELECT
        C.CUSTOMER_ID,
        C.FULL_NAME,
        O.ORDER_ID
FROM
        CUSTOMERS C NATURAL JOIN ORDERS O;
```

Joins on the column CUSTOMER_ID automatically if both tables have it.

INNER JOIN

- A type of join where rows from two or more tables are combined based on a condition, usually a shared column, and only the rows that satisfy the condition are returned.
- It can involve any type of condition (e.g., =, <, >, etc.), not necessarily only equality.




```sql
SELECT
        employees.name,
        departments.department_name
FROM employees
INNER JOIN departments
ON employees.salary > departments.average_salary;
```

The above query will return a list of employees and their respective department names, but only for those employees whose salary is greater than the average salary of the department they are associated with.

Example – Inner join

customers table
tableA/table1/left

customerID	customerName
1	Microsoft
2	Apple
3	Google

orders table
tableB/table2/right

orderID	customerID	orderDate
1	1	2003-09-15
2	2	2004-05-12
3	2	2006-03-19

```
SELECT *
    FROM Customer
    INNER JOIN Orders ON
    Customer.CustomerID = Orders.CustomerID;
```

customerID	customerName	orderID	customerID	orderDate
1	Microsoft	1	1	15-Sep-03
2	Apple	2	2	12-May-04
2	Apple	3	2	19-Mar-06

EQUI JOIN (INNER JOIN with = condition)

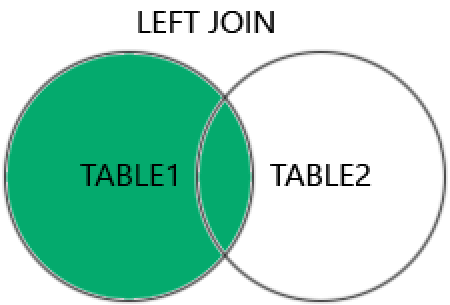
- A type of join where the condition uses the **equality** operator **(=).** It is a subset of inner join.
- Doesn’t include non-matching records.

```
SELECT
        C.CUSTOMER_ID,
        C.FULL_NAME,
        O.ORDER_ID
FROM CUSTOMERS C
INNER JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID;
```

Only customers who have orders are shown.

Left Join (Left Outer Join)

- Returns all records from the left table and matching records from the right table.
- If there is no match, NULLs are placed for missing right-table values.




```
SELECT
        C.CUSTOMER_ID,
        C.FULL_NAME,
        O.ORDER_ID
FROM CUSTOMERS C
LEFT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID;
```

Shows all customers even if they haven't placed orders (NULL ORDER_ID).

https://www.w3schools.com/sql/sql_join.asp

Example – Left join

customers table
tableA/table1/left

customerID	customerName
1	Microsoft
2	Apple
3	Google

orders table
tableB/table2/right

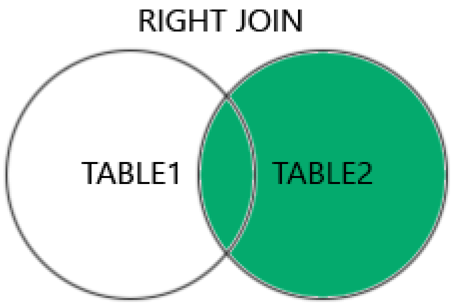
orderID	customerID	orderDate
1	1	2003-09-15
2	2	2004-05-12
3	2	2006-03-19

```
SELECT *
    FROM Customer
    LEFT JOIN Orders ON
    Customer.CustomerID = Orders.CustomerID;
```

customerID	customerName	orderID	customerID	orderDate
1	Microsoft	1	1	15-Sep-03
2	Apple	2	2	12-May-04
2	Apple	3	2	19-Mar-06
3	Google	-	-	-

RIGHT JOIN (RIGHT OUTER JOIN)

- Returns **all rows** from the right table and matching rows from the left table.
- If **no** match is found, **NULL** values are returned for columns from the **left** table.




```
SELECT
        C.CUSTOMER_ID,
        C.FULL_NAME,
        O.ORDER_ID
FROM CUSTOMERS C
RIGHT JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID;
```

Shows all orders even if they don't belong to a customer (NULL CUSTOMER_ID).

Example – Right join

customers table
tableA/table1/left

customerID	customerName
1	Microsoft
2	Apple
3	Google

orders table
tableB/table2/right

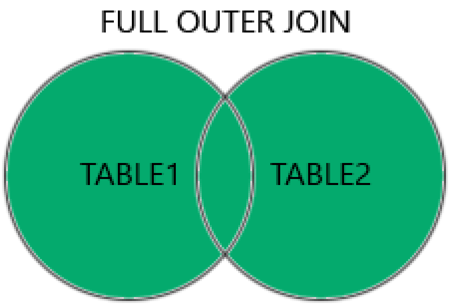
orderID	customerID	orderDate
1	1	2003-09-15
2	2	2004-05-12
3	2	2006-03-19

```
SELECT *
    FROM Customer
    RIGHT JOIN Orders ON
    Customer.CustomerID = Orders.CustomerID;
```

customerID	customerName	orderID	customerID	orderDate
1	Microsoft	1	1	15-Sep-03
2	Apple	2	2	12-May-04
2	Apple	3	2	19-Mar-06

FULL JOIN (FULL OUTER JOIN)

- Returns **all rows** from both tables, **matching** rows where possible.
- If no match is found, **NULL** values are returned for columns from the **non-matching table**.
- Combines the results of **both left join** and **right join**.




```sql
SELECT
    C.CUSTOMER_ID,
    C.FULL_NAME,
    O.ORDER_ID
FROM CUSTOMERS C
FULL JOIN ORDERS O ON C.CUSTOMER_ID = O.CUSTOMER_ID;
```

Includes all customers and all orders, even if no match exists.

Which join…

- **Natural Join** = if you want an automatic match (but be careful of unintended joins).
- **Left Join =** if you need all records from the left table, even if no match exists.
- **Right Join** = if you need all records from the right table, even if no match exists.
- **Full Join =** if you need all records from both tables, regardless of matches.
- **Inner Join** = You want to return only the rows that have matching values in both tables.
- **Equi Join (Inner Join) =** if you only need exact matches between tables.

Solving Real Queries

- As it happens, many real queries can be solved using a standard formula of *restrict*, *project*, and *natural join*.
- Here is another example:
- Query: List the names of the members of staff who work in the Computing Science department.

DEPT		
Dno	***Name***	***Rooms***
31	Computing Science	18
49	Management	15
55	Basket-weaving	3

STAFF		
Sno	***Name***	***Dno***
SG86	Savi Maharaj	31
SP52	Paul Kingston	49
ST22	Richard Bland	31

- To solve this query we use only these three operators:

Restrict, Project, Natural Join

Solution to Query - I

- First we use the restrict operator to obtain only the information about the Computing Science department:

R1 = **RESTRICT** DEPT **TO** *Name* = 'Computing Science'

R1		
Dno	***Name***	***Rooms***
31	Computing Science	18

- Using this result, we can now project out only the *Dno* column for natural join.

R2 = **PROJECT** *Dno* **FROM** R1

R2
Dno
31

- We will then use this result to perform a natural join with the STAFF table….

Solution to Query - II

- The natural join is as follows:

R3 = R2 ⋈ STAFF

R3		
Sno	***Name***	***Dno***
SG86	Savi Maharaj	31
ST22	Richard Bland	31

- The query asked only for the names of staff, so to finish this query, we need to project this information from R3.

RESULT = **PROJECT** *Name* **FROM** R3

RESULT
Name
Savi Maharaj
Richard Bland

- The solution took four steps.
  - Many queries can be answered in this way.
  - The first *project* operation wasn't strictly necessary, so we could have done it in three steps.

Using the Relational Algebra – recap!

- We have covered the majority of operations commonly used in the relational algebra.
- As our examples have shown, we can combine operations to produce answers to queries.
- Most queries can be solved by a combination of *restriction*, *projection*, and *join*.
- However, some complex queries may also require *union* or *difference*.
- Consider the data and the queries on the following slides:

Sample Relations

RENTER				
Rno	***Fname***	***Lname***	***Address***	***Phone***
CR76	John	Kay	56 High St	0171-774-5632
CR56	Aline	Stewart	64 Fern Dr	0141-848-1825
CR74	Mike	Ritchie	18 Tain St	01475-392178
CR62	Mary	Tregear	5 Tarbot Rd	*01224-196720*

VIEWING			
Rno	***Pno***	***Date***	***Comment***
CR56	*PA14*	*20-Apr-95*	*too small*
CR76	*PG4*	*20-Apr-95*	*too remote*
CR56	*PG4*	*26-May-95*	
CR62	*PA14*	*14-May-95*	*no dining room*
CR56	*PG36*	*28-Apr-95*	

PROPERTY							
Pno	***Street***	***Area***	***City***	***Postcode***	***Type***	***Rooms***	***Rent***
PA14	16 Holhead	Dee	Aberdeen	AB7 5SU	House	6	650.00
PL94	6 Argyll St	Kilburn	London	NW2	Flat	4	400.00
PG4	6 Lawrence St	Partick	Glasgow	G11 9QX	Flat	3	350.00
PG36	2 Manor Rd		Glasgow	G32 4QX	Flat	3	375.00
PG21	18 Dale Rd	Hyndland	Glasgow	G12	House	5	600.00

A Complex Query

- List the names of renters who have viewed *all properties with three rooms*.
- (Hint: Consider the different steps required, then combine them.)

Steps

1 List the names and numbers of all renters.

2 List the numbers of properties with 3 rooms.

3 List all possible combinations of renters and 3-room properties.

4 List the names and numbers of renters who have viewed 3-room properties (including the property numbers).

5 Use the results of steps 3 and 4 to list the renter and 3-room property combinations that have *not* actually happened.

6 Using the result of step 5, list the names and numbers of renters who have not viewed all 3-room properties.

7 Finally, use the results of steps 1 and 6 to solve the query!

Steps 1 and 2

- Step 1: List the names and numbers of all renters.
- This is a simple *projection* operation:

R1 = PROJECT *Rno, Fname, Lname* FROM *Renter*

R1		
Rno	***Fname***	***Lname***
CR76	John	Kay
CR56	Aline	Stewart
CR74	Mike	Ritchie
CR62	Mary	Tregear

- Step 2: List the numbers of properties with 3 rooms.
- This requires a simple combination of *restriction* and *projection*.

R2 = PROJECT *Pno* FROM

(RESTRICT *Property TO Rooms = 3*)

R2
Pno
PG4
PG36

Step 3

- Step 3: List all possible combinations of renters and 3 room properties.
- We use a raw *cartesian product* to combine the results from steps 1 and 2.

R3 = R1 * R2

R3			
Rno	***Fname***	***Lname***	***Pno***
CR76	John	Kay	PG4
CR76	John	Kay	PG36
CR56	Aline	Stewart	PG4
CR56	Aline	Stewart	PG36
CR74	Mike	Ritchie	PG4
CR74	Mike	Ritchie	PG36
CR62	Mary	Tregear	PG4
CR62	Mary	Tregear	PG36

Note that not all of these viewings have happened! It is a list of all *possible* viewings.

Step 4

- Step 4: List the names and numbers of renters who have actually viewed 3 room properties (including the property numbers).
- Information about viewings that have actually occurred is held in the *Viewing* relation.
- We can extract it by performing a *natural join* of the *Viewing* relation with the results from steps 1 and 2:

$$R4 = R1 \bowtie (\text{PROJECT } \mathit{Rno}, \mathit{Pno} \text{ FROM Viewing}) \bowtie R2$$

R4			
Rno	***Fname***	***Lname***	***Pno***
CR76	John	Kay	PG4
CR56	Aline	Stewart	PG4
CR56	Aline	Stewart	PG36

Step 5

- Step 5: Use the relations in steps 3 and 4 to list the renter and 3 room property combinations that have <u>not</u> actually happened.
- From step 3 we have a set of all possible viewings of three room properties.
- In step 4, we produced a list of viewings of three room properties that had actually happened.
- We can solve step 5 using *set difference* to eliminate viewings in step 4 from viewings in step 3:

R5 = R3 – R4

R5			
Rno	***Fname***	***Lname***	***Pno***
CR76	John	Kay	PG36
CR74	Mike	Ritchie	PG4
CR74	Mike	Ritchie	PG36
CR62	Mary	Tregear	PG4
CR62	Mary	Tregear	PG36

Step 6

- Step 6: Using the relation in step 5, list the names and numbers of renters who have <u>not</u> viewed all 3 room properties.

- This can be achieved using a simple *projection* operation:

R6 = PROJECT *Rno*, *Fname*, *Lname* FROM R5

R6		
Rno	***Fname***	***Lname***
CR76	John	Kay
CR74	Mike	Ritchie
CR62	Mary	Tregear

- This operation seemingly just omits the *Pno* column from R5, which we no longer need.

- However, this operation illustrates the way in which the projection operation eliminates duplicate rows.

Step 7

- Step 7: Use the relations from steps 1 and 6 to solve the query!
- In step 1 we created a list of all renters.
- From the previous step we have a list of renters who have <u>not</u> viewed all three room properties.
  - i.e. The opposite of what we require to solve the query.
- To finish off, we simply use set difference to subtract step 6 from step 1:

Result = R1 – R6

RESULT		
<u>Rno</u>	***Fname***	***Lname***
CR56	Aline	Stewart

- We have found one person who has viewed all properties with three rooms.
- In closing, it is worth noting that there is special relational operator called *division* that could do this more concisely.

DATABASE LANGUAGES

- Databases require two (main) sorts of languages :
  - the data definition language and the data manipulation language.

- The **data definition language** (DDL) is used to create databases, tables, to authorize users etc.
  - CREATE, ALTER, DROP, and TRUNCATE

- The **data manipulation language** (DML) is used to retrieve data to add and delete data and to modify data.
  - SELECT, INSERT, UPDATE and DELETE

- Data Query Language (DQL)
- Data Control Language (DCL)
- Transaction Control Language (TCL)

Case Sensitivity

Quoted strings are case sensitive

```
SELECT
    CUST_NUM,
    CUST_NAME,
    CUST_ADDRESS
FROM CUSTOMER
WHERE CUST_NUM >= 12212;
```

IS THE SAME AS:

```
select
    cust_num,
    cust_name,
    cust_address
from customer
where cust_num >= 12212;
```

BUT:

```
SELECT
  CUST_NUM,
  CUST_NAME,
  CUST_ADDRESS
FROM CUSTOMER
WHERE CUST_ADDRESS = ‘SPINKHILL’;
```

IS DIFFERENT FROM:

```
SELECT
    CUST_NUM,
    CUST_NAME,
    CUST_ADDRESS
FROM CUSTOMER
WHERE CUST_ADDRESS = ‘Spinkhill’;
```

Questions

Course Content

1. Introduction to Relational Databases *(Introduction + Relational Model)*
2. Data Modelling - *(Entity Relationship Modelling + The Enhanced Entity Relationship Model)*
3. Database Design and SQL - *(Logical modelling + Introduction to SQL)*
4. Further SQL - *(Advanced SQL queries + Creating tables with SQL)*
5. Normalisation - *(Normalisation to second normal form + Third normal form)*

X THE PLACE OF USEFUL LEARNING

University of Strathclyde Glasgow

Database Fundamentals – CS990

Database and Web Systems Development - CS952

Image source: https://www.strath.ac.uk/branding/