COM106: Introduction to Databases

Further SQL – More Operators and JOIN Queries

Further SQL – More Operators and JOIN Queries

SQL provides a number of **Set Operators** to combine sets of rows returned by queries

We will look at UNION, EXCEPT and INTERSECT

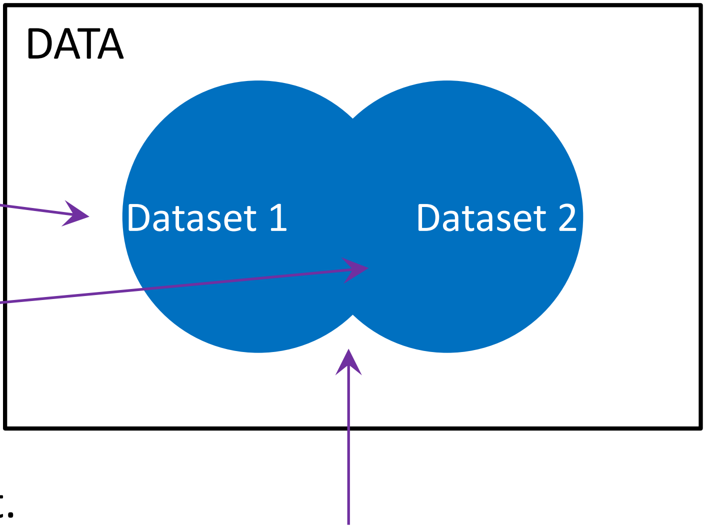
UNION - Combines two query statements (tables) across **compatible** attributes

- Requires the *same number of attributes* in each query
- Each corresponding pair of attributes must be *defined on the same domain* (compatible)
- Duplicate values are eliminated

```
SELECT <attribute_1>, …. < attribute_n>
FROM tables
[WHERE conditions]
UNION
SELECT <attribute_1>, …. < attribute_n>
FROM tables
[WHERE conditions];
```

Gives Dataset 1

Gives Dataset 2



*The **UNION** operator returns records in the shaded area - records that exist in Dataset 1 **or** Dataset 2*

The **UNION** operator selects only **distinct values** by default.

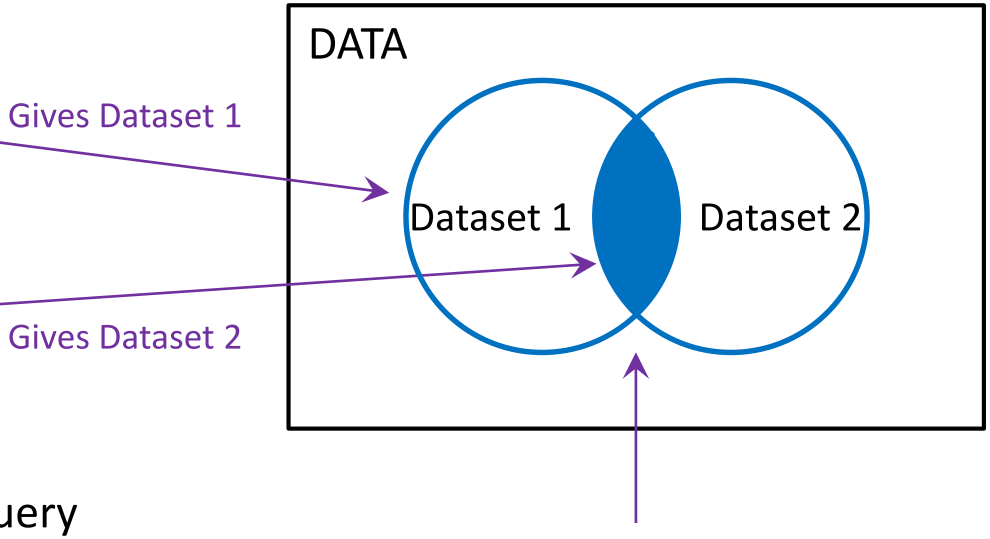
To allow duplicate values, use the ALL keyword - **UNION ALL**

Further SQL – More Operators and JOIN Queries

INTERSECT - Returns values found in both queries

- Requires the *same number of attributes* in each query
- Each corresponding pair of attributes must be *defined on the same domain* (compatible)

```
SELECT <attribute_1>, …. < attribute_n>
FROM tables
[WHERE conditions]
INTERSECT
SELECT <attribute_1>, …. < attribute_n>
FROM tables
[WHERE conditions];
```



*The **INTERSECT** operator returns records in the shaded area - records that exist in **both** Dataset 1 **and** Dataset 2*

EXCEPT - Returns values found in the first query not found in right query

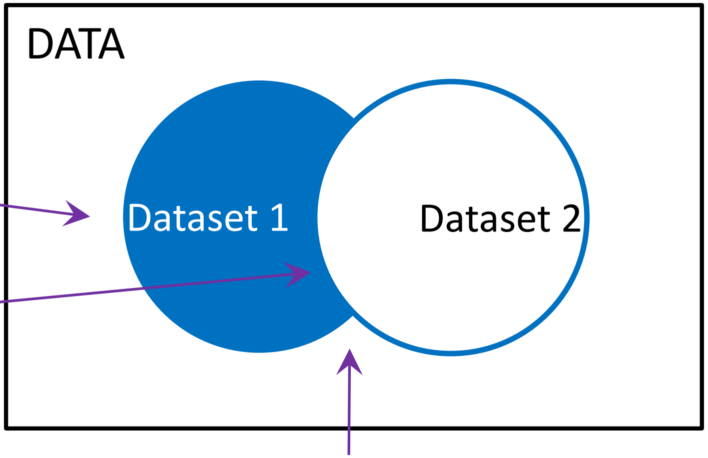
- Requires the *same number of attributes* in each query
- Each corresponding pair of attributes must be *defined on the same domain* (compatible)

Further SQL – More Operators and JOIN Queries

```
SELECT <attribute_1>, …. < attribute_n>
FROM tables
[WHERE conditions]
EXCEPT
SELECT <attribute_1>, …. < attribute_n>
FROM tables
[WHERE conditions];
```

Gives Dataset 1

Gives Dataset 2



*The **EXCEPT** operator returns records in the shaded area - records that exist in Dataset 1 **and not in** Dataset 2*

An Example of EXCEPT

A company holds details of its **SUPPLIERS** and its **CUSTOMERS**

SUPPLIER

sup_no	s_name	country
s121	Big IT	Germany
s135	Top Design	Italy
s147	ALY Ltd	USA
s216	Microsoft	France
s227	Apple	Zambia
s251	PC Sport	Vietnam

CUSTOMER

cus_no	c_name	contact
c164	Apple	Jobs
c045	IT r us	Sinclair
c237	Microsoft	Gates
c256	Myco	Jones
c290	MFN Ltd	Smith
c515	HP	Bloggs

```
SELECT s_name
FROM SUPPLIER
EXCEPT
SELECT c_name
FROM CUSTOMER
```

s_name
Big IT
Top Design
ALY Ltd
PC Sport

Get the names of suppliers that are not also customers

s_name and c_name must have the same datatype

***Microsoft** and **Apple** from SUPPLIER are not in the results because they occur in CUSTOMER*

Further SQL – More Operators and JOIN Queries

An Example of **INTERSECT**

SUPPLIER

sup_no	s_name	country
s121	Big IT	Germany
s135	Top Design	Italy
s147	ALY Ltd	USA
s216	Microsoft	France
s227	Apple	Zambia
s251	PC Sport	Vietnam

CUSTOMER

cus_no	c_name	contact
c164	Apple	Jobs
c045	IT r us	Sinclair
c237	Microsoft	Gates
c256	Myco	Jones
c290	MFN Ltd	Smith
c515	HP	Bloggs

```
SELECT s_name
FROM SUPPLIER
INTERSECT
SELECT c_name
FROM CUSTOMER
```

s_name
Microsoft
Apple

***Microsoft** and **Apple** are in both SUPPLIER and CUSTOMER*

Get the names of suppliers that are also customers

s_name and c_name must have the same datatype

An Example of **UNION**

SUPPLIER

sup_no	s_name	country
s121	Big IT	Germany
s135	Top Design	Italy
s147	ALY Ltd	USA
s216	Microsoft	France
s227	Apple	Zambia
s251	PC Sport	Vietnam

CUSTOMER

cus_no	c_name	contact
c164	Apple	Jobs
c045	IT r us	Sinclair
c237	Microsoft	Gates
c256	Myco	Jones
c290	MFN Ltd	Smith
c515	HP	Bloggs

```
SELECT s_name
FROM SUPPLIER
UNION
SELECT c_name
FROM CUSTOMER
```

s_name
Big IT
Top Design
ALY Ltd
Microsoft
Apple
PC Sport
IT r us
Myco
MFN Ltd
HP

***All** records in SUPPLIER and CUSTOMER*

Get the names of all suppliers and customers

s_name and c_name must have the same datatype

Further SQL – More Operators and JOIN Queries

SQL also provides a number of **Logical Operators** used to test for the truth of some condition. We have already seen AND, OR, BETWEEN, NOT and IN – (*there are a number of others*)

The LIKE Operator

LIKE allows the use of **wildcards** to perform **pattern matching** on an attribute in a query. It can be used in the WHERE clause of a SELECT, INSERT, UPDATE, or DELETE statement.

%	wildcard representing any string of characters
_	wildcard representing a single character
[ ]	looks for any match within a range of characters e.g. [a-m]
[^]	looks for any match not in the specified range e.g. [^a-m]

Some Examples:

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Smart	19000	3
852407	Brown	16000	3
852455	Bruce	25100	2
852491	Thrower	30500	1
852514	Dale	11650	2
852530	Dole	26980	4

Get the names of employees starting with 'S'

```
SELECT ename
FROM EMPLOYEE
WHERE ename LIKE 'S%';
```

ename
Smith
Smart

Get the names of employees containing the string 'row' anywhere

```
SELECT ename
FROM EMPLOYEE
WHERE ename LIKE '%row%';
```

ename
Brown
Thrower

Further SQL – More Operators and JOIN Queries

Get the names of employees matching 'D?le'

```
SELECT ename
FROM EMPLOYEE
WHERE ename LIKE 'D_le';
```

ename
Dale
Dole

EMPLOYEE

enum	ename	salary	floor
852341	Smith	15000	1
852358	Smart	19000	3
852407	Brown	16000	3
852455	Bruce	25100	2
852491	Thrower	30500	1
852514	Dale	11650	2
852530	Dole	26980	4

Get the names of employees with any characters from n to r

```
SELECT ename
FROM EMPLOYEE
WHERE ename LIKE '%[n-r]%';
```

ename
Smart
Brown
Bruce
Thrower

The IS NULL Operator

Not the same as a space (' ') or a zero

NULL is a special marker used in SQL to indicate that a data value **does not exist**

NULL is not a value – it is a **state** indicating 'unknown' or the **absence of a value**.

SQL provides two special Null-specific comparison **predicates**:

IS NULL – tests whether an attribute has a value and returns **TRUE** if it is NULL

IS NOT NULL – tests whether an attribute has a value and returns **TRUE** if it does

```
e.g., SELECT attribute_a
      FROM A_TABLE
      WHERE attribute_b IS NULL
```

Further SQL – More Operators and JOIN Queries

JOIN Fundamentals

A **JOIN** is a means for combining columns from two or more tables by using values common to each.

There are a number of different ways in which two tables can be joined - ANSI-standard SQL specifies five types of join: **INNER**, **LEFT OUTER, RIGHT OUTER, FULL OUTER** and **CROSS**.

As a special case, a table can join to itself in a **SELF-JOIN**.

*We've already seen examples of **INNER** JOINS and **SELF** JOINS*

We will not consider CROSS JOINS here

The JOIN condition is specified in the FROM clause as:

*FROM table1 **join_type** table2 [**ON** (**join_condition**)]*

The **join_condition** defines the way two tables are related in a query by specifying:

- The column from each table to be used for the join (typically, *but not necessarily*, on a PK/FK match).
- A logical operator (for example, = or <>, etc) to be used in comparing values in the columns.

*'=' is most commonly used – also known as an **equi-join***

INNER JOINs **only** can also be specified in the WHERE clause

Mark 1 - *implicit join notation*

```
SELECT ename, role
FROM EMPLOYEE, WORKS_ON
WHERE EMPLOYEE.enum = WORKS_ON.enum
AND floor = 1;
```

Mark 2 - *explicit join notation*

```
SELECT ename, role
FROM EMPLOYEE INNER JOIN WORKS_ON
ON EMPLOYEE.enum = WORKS_ON.enum
WHERE floor = 1;
```

Further SQL – More Operators and JOIN Queries

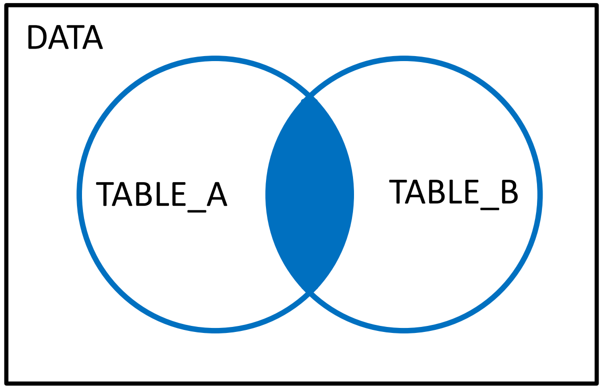
Explicit join notation is considered **best practice** since the join conditions are separated from any other conditions in the WHERE clause.

All JOIN queries create a **new result table** by combining column values of two tables (TABLE_A and TABLE_B) based upon the ***join_condition***. ← *Also known as the **join predicate***

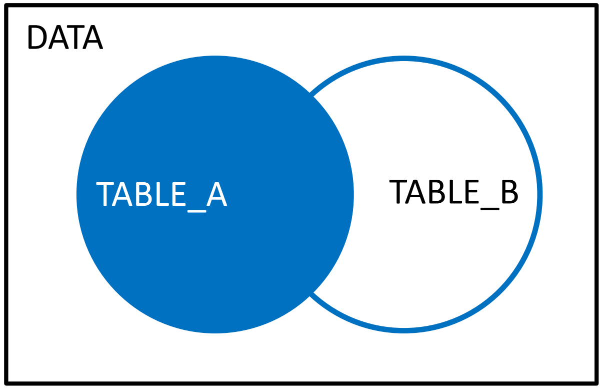
Each row of TABLE_A is compared with each row of TABLE_B to find all pairs of rows which satisfy the **join predicate**.

The ***join_type*** specifies how the tables are to be combined (*as in the **Venn Diagrams** below*):

TABLE_A INNER JOIN TABLE_B



TABLE_A LEFT OUTER JOIN TABLE_B

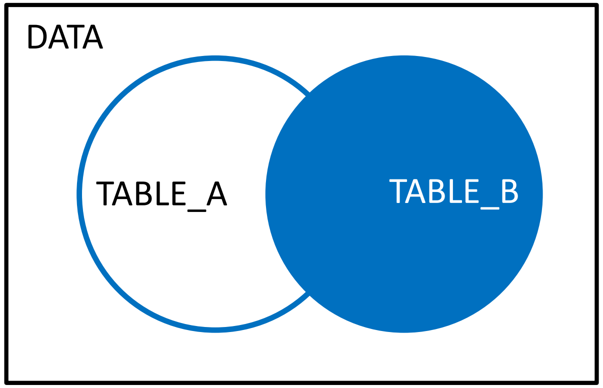


INNER JOIN - Select all records from TABLE_A and TABLE_B, where the join condition is met.

LEFT OUTER JOIN - Select all records from TABLE_A, along with records from TABLE_B for which the join condition is met (if at all).

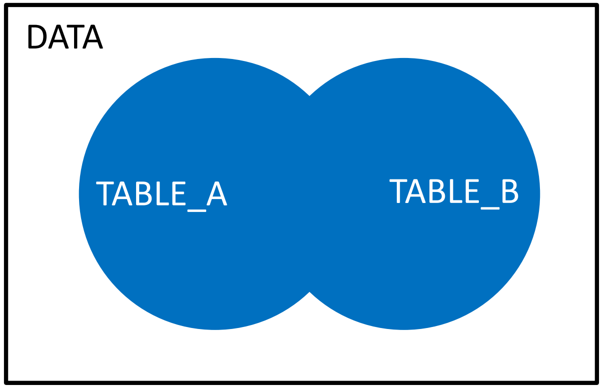
Further SQL – More Operators and JOIN Queries

TABLE_A RIGHT OUTER JOIN TABLE_B



RIGHT OUTER JOIN - Select all records from TABLE_B, along with records from TABLE_A for which the join condition is met (if at all).

TABLE_A FULL OUTER JOIN TABLE_B



FULL OUTER JOIN - Select all records from TABLE_A and TABLE_B, regardless of whether the join condition is met or not.

Consider the following schema:

CLUB (club_id, club_name)
MEMBER (member_id, member_name, club_id*)

With sample data as shown:

We will look at the result of the different ***join types*** based on the ***join condition***:-

MEMBER.club_id = CLUB.club_id

CLUB

club_id	club_name
C1	Tennis
C2	Archery
C3	Judo

MEMBER

member_id	member_name	club_id
M20	Smith	C3
M21	Jones	*Null*
M22	White	C1
M23	Black	C3
M24	Green	C1
M25	Brown	*Null*

Further SQL – More Operators and JOIN Queries

INNER JOIN

```
SELECT *
FROM MEMBER INNER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id;
```

Note: Members “Jones" and “Brown” and the “Archery” club do not appear in the results. Neither of these has any matching rows in the other respective table.

RESULT TABLE

member_id	member_name	club_id	club_name
M20	Smith	**C3**	Judo
M22	White	**C1**	Tennis
M23	Black	**C3**	Judo
M24	Green	**C1**	Tennis

LEFT OUTER JOIN

***LEFT** Table* ***RIGHT** Table*

```
SELECT *
FROM MEMBER LEFT OUTER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id;
```

Note: A **LEFT JOIN** returns all the values from the left table, plus matched values from the right table or *NULL* in case of no matching join predicate.

RESULT TABLE

member_id	member_name	club_id	club_name
M20	Smith	**C3**	Judo
M21	Jones	***Null***	*Null*
M22	White	**C1**	Tennis
M23	Black	**C3**	Judo
M24	Green	**C1**	Tennis
M25	Brown	***Null***	*Null*

Further SQL – More Operators and JOIN Queries

RIGHT OUTER JOIN

```
SELECT *
FROM MEMBER RIGHT OUTER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id;
```

Note: A **RIGHT JOIN** is similar to a LEFT JOIN, but with the treatment of the tables reversed. Returns all the values from the right table, plus matched values from the left table or *NULL* in case of no matching join predicate.

RESULT TABLE

member_id	member_ name	club_id	club_name
M20	Smith	**C3**	Judo
M22	White	**C1**	Tennis
M23	Black	**C3**	Judo
M24	Green	**C1**	Tennis
Null	*Null*	**C2**	Archery

Notice- MEMBER **RIGHT OUTER JOIN** CLUB gives the same result as CLUB **LEFT OUTER JOIN** MEMBER

The order of the tables is important

FULL OUTER JOIN

```
SELECT *
FROM MEMBER FULL OUTER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id;
```

Note: A **FULL JOIN** combines the results of both left and right outer joins.

The joined table will contain all records from both tables, with *NULL*s for missing matches on either side.

Some RDBMS do not support FULL JOINs, eg., MySQL

RESULT TABLE

member_id	member_ name	club_id	club_name
M20	Smith	**C3**	Judo
M21	Jones	***Null***	*Null*
M22	White	**C1**	Tennis
M23	Black	**C3**	Judo
M24	Green	**C1**	Tennis
M25	Brown	***Null***	*Null*
Null	*Null*	**C2**	Archery

Further SQL – More Operators and JOIN Queries

Some Examples of Use

List all members and the clubs to which they belong.

Using an INNER JOIN

```
SELECT *
FROM MEMBER INNER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id;
```

member_id	member_name	club_id	club_name
M20	Smith	C3	Judo
M22	White	C1	Tennis
M23	Black	C3	Judo
M24	Green	C1	Tennis

Using a LEFT JOIN

```
SELECT *
FROM MEMBER LEFT OUTER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id;
```

member_id	member_name	club_id	club_name
M20	Smith	C3	Judo
M21	Jones	*Null*	*Null*
M22	White	C1	Tennis
M23	Black	C3	Judo
M24	Green	C1	Tennis
M25	Brown	*Null*	*Null*

Notice:- The **INNER JOIN** excludes members that don't have a club
The **LEFT JOIN** lists all members, ***including*** those that don't have a club

INNER JOIN is the most commonly used join, but should be **treated with care**, especially if the attribute on which the tables are joined can contain **NULLs**.

Further SQL – More Operators and JOIN Queries

Get the members that do not belong to any club

Using a LEFT JOIN

```
SELECT member_id, member_name
FROM MEMBER LEFT OUTER JOIN CLUB
ON MEMBER.club_id = CLUB.club_id
WHERE CLUB.club_id IS NULL;
```

member_id	member_name
M21	Jones
M25	Brown

Get the clubs that do not have any members

Using a RIGHT JOIN

```
SELECT C.club_id, club_name
FROM MEMBER AS M RIGHT OUTER JOIN CLUB AS C
ON M.club_id = C.club_id
WHERE M.member_id IS NULL;
```

club_id	club_name
C2	Archery

Notice the use of aliases:
MEMBER AS M,
CLUB AS C

Compare INTERSECT and INNER JOIN

Some queries can be answered using either an **INTERSECT** or an **INNER JOIN** approach

e.g., Rewrite the **INTERSECT** query on slide 5 using an **INNER JOIN**

Can the **INNER JOIN** query on slide 13 be rewritten using **INTERSECT**? **NO**