THE UNIVERSITY of ADELAIDE

Web and Database Computing

adelaide.edu.au

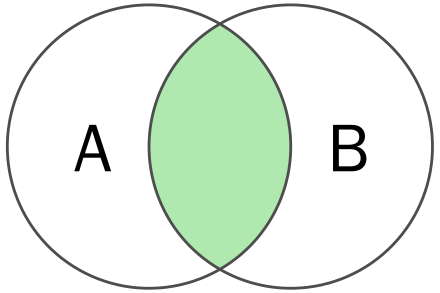
Databases in Web Applications: Combining Tables and Queries

Doing more with Joins

Joins; Recap

- Joins combine tables, usually on a common column.
- The most common type of join is an **inner join** which only returns rows with matching values in a given column.

```
SELECT * FROM TableA INNER JOIN TableB
  ON TableA.column1 = TableB.column2;
```



- The ON clause is used to specify which columns should be matched
  - i.e. Any rows from `TableA` & `TableB` whose `column1` and `column2` values match will be combined into a single row in the result.

Other Joins: Cartesian

While an inner join may be the most common type of join that we use, there are other types of joins.

- A **Cartesian Join** is the simplest type of join.

```
SELECT * FROM tableA, tableB;
```

- Each row is combined with each column.
- This is usually VERY INNEFFICIENT; **avoid unless absolutely necessary**.

A

col1
a
b
c

B

col2
x
y
z

Result

col1	col2
a	x
a	y
a	z
b	x

...

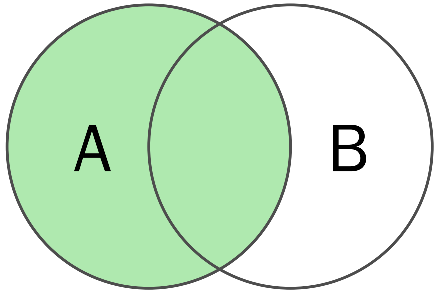
...

b	y
b	z
c	x
c	y
c	z

Other Joins; Left Outer Join

- The **Left Outer Join** joins two tables, keeping all rows of the first table.

```
SELECT * FROM TableA LEFT JOIN TableB
  ON TableA.column1 = TableB.column2;
```



- The ON clause is used to specify which columns should be matched
  - Rows from the first table that aren't matched will be padded out with NULL/default values.

TableA

column1
a
b
c

TableB

column2
a
y
c

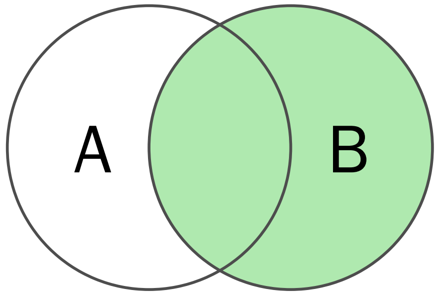
Result

column1	**column2**
a	a
b	NULL
c	c

Other Joins; Right Outer Join

- The **Right Outer Join** joins two tables, keeping all rows of the second table (reverse Left Outer Join)

```
SELECT * FROM TableA RIGHT JOIN TableB
  ON TableA.column1 = TableB.column2;
```



- The ON clause is used to specify which columns should be matched
  - Rows from the second table that aren't matched will be padded out with NULL/default values.
  - Some DBMS' do not support this because you could instead switch the order of the tables.

TableA

column1
a
b
c

TableB

column2
a
y
c

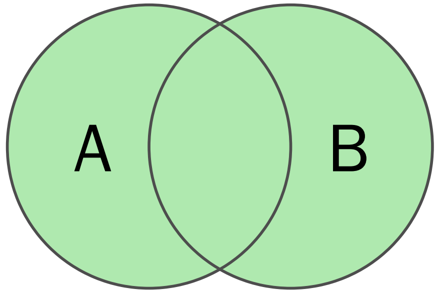
Result

column1	**column2**
a	a
NULL	y
c	c

Other Joins; Full Outer Join

- The **Full Outer Join** joins two tables, keeping all rows of the both tables

```
SELECT * FROM TableA FULL JOIN TableB
  ON TableA.column1 = TableB.column2;
```



- The ON clause is used to specify which columns should be matched
  - Rows from the either table that aren't matched will be padded out with NULL/default values.

TableA

column1
a
b
c

TableB

column2
a
y
c

Result

column1	column2
a	a
b	NULL
NULL	y
c	c

Other Joins; Natural Join

- A **Natural Join** is the same as an Inner Join, however the columns used to join are chosen automatically.

```
SELECT * FROM TableA NATURAL JOIN TableB;
```

- The column chosen will be one that has the exact same name and data type in both tables.

Combining Rows

Unions and Intersections

Sometimes we may want to combine the results of multiple queries into a single set of results.

- **Unions** append the results of one query to another:

```
SELECT * FROM TableA
UNION
SELECT * FROM TableB;
```

TableA

column1
a
b
c

TableB

column2
x
y

Result

column1
a
b
c
x
y

Unions

- Each SELECT statement within the Union must have the same number of fields in the result sets with similar data types.
- The column name in the result will be the name of the column from the first table.
- A standard union omits any duplicate rows.
    - To retain duplicate rows, use `UNION ALL`

Intersections

If we want only the rows that are returned from both queries, we can use an **Intersect**.

- **Intersections** return the matching results from two queries:

```
SELECT * FROM TableA
INTERSECT
SELECT * FROM TableB;
```

TableA

column1
a
b
c

TableB

column2
a
y
c

Result

column1
a
c

Modifying Column and Table Names

For use in queries and results

Aliases

Sometimes we may want to rename a column for outputting results, or make reading a query easier.

- This can be achieved using Aliases.
- Aliases use the `AS` keyword.

Aliases for columns

- We can alias columns like this:

```sql
SELECT column1 AS letters FROM TableA;
```

TableA

column1
a
b

Result

letters
a
b

Aliases

- We can alias tables like this:

```
SELECT column1 FROM TableA AS A;
```

- This is most useful for operations like joins:

```
SELECT A.column1,B.column2
FROM TableA AS A
INNER JOIN TableB AS B
ON A.column1 = B.column2
WHERE A.column1 = 'a';
```

Ordering/Limiting results

Sorting results using ORDER BY

A common desire is for the results of a query to be sorted on a particular column.

- This can be achieved using ORDER BY

```
SELECT * FROM Customers
  ORDER BY Country;
```

- You can order Ascending (A-Z) or Descending (Z-A)
- You can also specify secondary and tertiary columns

```
SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;
```

Restricting results

While most of the queries and databases we've been working with only have a few entries, some queries on larger databases could return thousands or millions of results.

- Large result sets can have performance and bandwidth consequences.
- We can limit the total number of results using LIMIT
- If two numbers provided, the first is the offset into the result set.

```sql
SELECT * FROM Customers
  LIMIT 50, 10;
```

- Use in conjunction with ORDER BY to ensure key results not omitted:

```sql
SELECT * FROM Customers
  ORDER BY Country ASC, CustomerName DESC
  LIMIT 100;
```

Removing Duplicates

Some queries may return duplicate results, especially where a Primary Key is not included in the columns returned.

- Often we wnat to exclude duplicate results.
- We can use the DISTINCT keyword to only return unique results.

```
SELECT DISTINCT Country FROM Customers;
```

- Can work with multiple columns:

```
SELECT DISTINCT Country, City FROM Customers;
```

SUB CRUCE LUMEN

THE UNIVERSITY of ADELAIDE