SQL Unit 3
Joins

Ryan Nixon

Joining Tables

- Allows for data to be pulled from multiple tables at once
- Connects the tables using their relationships
- Still produces a single resultset

The Importance of Aliases

- When working with two tables names can get very confusing. It’s advisable to use aliases often to keep things understandable

-
  ```
  SELECT first AS firstname
  FROM person AS student
  ```

Ambiguity

- When joining two tables that have columns with the same name, an ambiguity takes place
- The database doesn't know which column to work with and requires the column to be explicitly defined

-
  ```
  SELECT p.first, p.last
  FROM person AS p
  WHERE p.age IS NOT NULL
  ```

Joins Illustrated

id	student_id	assn_id	points
1	1	1	5.00
2	6	3	27.00
3	6	1	9.00
4	5	2	13.50
5	3	3	25.00

id	first	last	age
1	Jacob	Williams	37
2	Emma	Emmerich	25
3	Nathan	Drake	32
4	Albert	Wesker	41
5	Alexandra	Vance	29
6	Christopher	Garden	46

id	student_id	assn_id	points	first	last	age
1	1	1	5.00	Jacob	Williams	37
2	6	3	27.00	Christopher	Garden	46
3	6	1	9.00	Christopher	Garden	46
4	5	2	13.50	Alexandra	Vance	29
5	3	3	25.00	Nathan	Drake	32

Unit 2 Assn

- Thinking back to Unit 2, there were many questions that required results from one query to be used in the second query
- Joins allow this to occur using a single query

Two Join Syntaxes

-
  ```
  SELECT *
  FROM person p, person_student s
  WHERE p.id = s.person_id
  ```

-
  ```
  SELECT *
  FROM person p
  INNER JOIN person_student s
     ON (p.id = s.person_id)
  ```

Two Join Syntaxes

- We will be using the second syntax for 2 reasons:
    - The first syntax is typically only used for INNER JOINs. Additional types must use the second syntax
    - The second syntax can handle much more complex queries without losing readability

Inner Joins

-
  ```
  INNER JOIN model
      ON (model.make_id = make.id)
  ```
- Inner joins are the most common join
- Connects two tables with a boolean condition, just like WHERE
- Any connection that fails (i.e. there are no records on the “right” that exist for the “left”) do not show up in the final resultset

Left & Right

- The directions on a join correspond to the location of the table names

-
  ```
  FROM person AS Left
  INNER JOIN person_student AS Right
  ```
- Direction is extremely important. The left is the “original” data whereas the right is new data being added

Inner Joins

```
SELECT *
FROM grade
INNER JOIN person
    ON person.id = grade.student_id
```

id	student_id	assn_id	points
1	1	1	5.00
2	6	3	27.00
3	6	1	9.00
4	5	2	13.50
5	3	3	25.00

id	first	last	age
1	Jacob	Williams	37
2	Emma	Emmerich	25
3	Nathan	Drake	32
4	Albert	Wesker	41
5	Alexandra	Vance	29
6	Christopher	Garden	46

id	student_id	assn_id	points	first	last	age
1	1	1	5.00	Jacob	Williams	37
2	6	3	27.00	Christopher	Garden	46
3	6	1	9.00	Christopher	Garden	46
4	5	2	13.50	Alexandra	Vance	29
5	3	3	25.00	Nathan	Drake	32

Inner Joins

```
SELECT person.*, grade.*
FROM person
INNER JOIN grade
    ON person.id = grade.student_id
```

id	first	last	age
1	Jacob	Williams	37
2	Emma	Emmerich	25
3	Nathan	Drake	32
4	Albert	Wesker	41
5	Alexandra	Vance	29
6	Christopher	Garden	46

id	student_id	assn_id	points
1	1	1	5.00
2	6	3	27.00
3	6	1	9.00
4	5	2	13.50
5	3	3	25.00

id	first	last	age	student_id	assn_id	points
1	Jacob	Williams	37	1	1	5.00
2	~~Emma~~	~~Emmerich~~	~~25~~			
3	Nathan	Drake	32	3	3	25.00
4	~~Albert~~	~~Wesker~~	~~41~~			
5	Alexandra	Vance	29	5	2	13.50
6	Christopher	Garden	46	6	3	27.00
6	Christopher	Garden	46	6	1	9.00

id	first	last	age	student_id	assn_id	points
1	Jacob	Williams	37	1	1	5.00
3	Nathan	Drake	32	3	3	25.00
5	Alexandra	Vance	29	5	2	13.50
6	Christopher	Garden	46	6	3	27.00
6	Christopher	Garden	46	6	1	9.00

Regarding Duplicates

- Note the duplication of Christopher Garden
- While a failed connection (NULL) will remove rows from the resultset, multiple connections will cause duplicates
- The duplication's purpose is to keep the ability to reference a single row without depending on other rows

Other Join Names

- Equi-Join - The only condition connecting the tables is the equality of their keys
- Natural Join - If the columns performing the connections are equal, merge them into a single column
- Cross Join - No conditions are specified; this causes the resultset to be the cartesian product of the two tables

Join Conditions

- Equi-Joins are not required. Additional conditions may be supplied to further limit results
- In rare cases this improves performance over using filters in the WHERE clause
    - In some products, the WHERE is applied to the final resultset after all joins have taken place
    - JOINs apply their filters as they build the results, keeping the final resultset small

Join Conditions

```
SELECT *
FROM grade
INNER JOIN person
    ON (person.id = grade.student_id AND person.age > 32)
```

id	student_id	assn_id	points
1	1	1	5.00
2	6	3	27.00
3	6	1	9.00
4	5	2	13.50
5	3	3	25.00

id	first	last	age
1	Jacob	Williams	37
2	Emma	Emmerich	25
3	Nathan	Drake	32
4	Albert	Wesker	41
5	Alexandra	Vance	29
6	Christopher	Garden	46

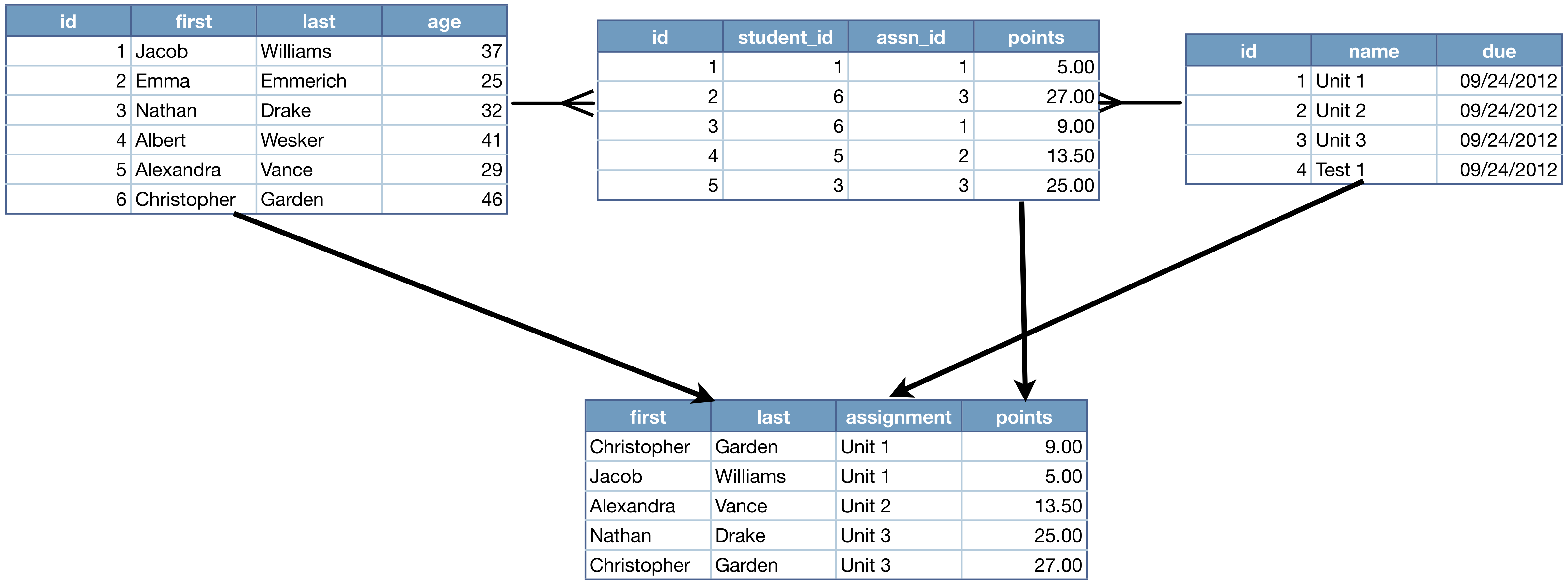
id	student_id	assn_id	points	first	last	age
1	1	1	5.00	Jacob	Williams	37
2	6	3	27.00	Christopher	Garden	46
3	6	1	9.00	Christopher	Garden	46
4	~~5~~	~~2~~	~~13.50~~	~~Alexandra~~	~~Vance~~	~~29~~
~~5~~	~~3~~	~~3~~	~~25.00~~	~~Nathan~~	~~Drake~~	~~32~~

Multiple Joins

- There is no limit to the number of joins that can be specified. Any join will work off of the resultset from the prior join. This allows columns from the Right table to be used in the next join
- This is perfect for Many-to-Many relationships

Multiple Joins

```
SELECT first, last, name AS assignment, points
FROM person
INNER JOIN grade ON (grade.student_id = person.id)
INNER JOIN assn ON (assn.id = grade.assn_id)
ORDER BY assn.due, assn.name, person.last
```



id	first	last	age
1	Jacob	Williams	37
2	Emma	Emmerich	25
3	Nathan	Drake	32
4	Albert	Wesker	41
5	Alexandra	Vance	29
6	Christopher	Garden	46

id	student_id	assn_id	points
1	1	1	5.00
2	6	3	27.00
3	6	1	9.00
4	5	2	13.50
5	3	3	25.00

id	name	due
1	Unit 1	09/24/2012
2	Unit 2	09/24/2012
3	Unit 3	09/24/2012
4	Test 1	09/24/2012

first	last	assignment	points
Christopher	Garden	Unit 1	9.00
Jacob	Williams	Unit 1	5.00
Alexandra	Vance	Unit 2	13.50
Nathan	Drake	Unit 3	25.00
Christopher	Garden	Unit 3	27.00

Self-Joining

- Tables can also be joined with themselves. This adds further importance to the use of aliases
- A perfect example of this is the parent-child relationship, where both sets of data are considered people

Self-Joining

```
SELECT parent.first AS parent_first, parent.last AS parent_last,
    child.first AS child_first, child.last AS child_last
FROM person AS parent
INNER JOIN person AS child
    ON (child.father_id = parent.id
        OR child.mother_id = parent.id)
```

id	mother_id	father_id	first	last	age
1	4	3	Todd	Flanders	8
2	6	5	Stewie	Griffin	1
3	NULL	NULL	Ned	Flanders	47
4	NULL	NULL	Maude	Flanders	NULL
5	NULL	NULL	Peter	Griffin	42
6	NULL	NULL	Lois	Griffin	39

parent_first	parent_last	child_first	child_last
Ned	Flanders	Todd	Flanders
Maude	Flanders	Todd	Flanders
Peter	Griffin	Stewie	Griffin
Lois	Griffin	Stewie	Griffin

Self-Joining

```
SELECT child.first AS child_first, child.last AS child_last,
    mother.first AS mother_first, mother.last AS mother_last,
    father.first AS father_first, father.last AS father_last
FROM person AS child
INNER JOIN person AS mother ON (mother.id = child.mother_id)
INNER JOIN person AS father ON (father.id = child.father_id)
```

id	mother_id	father_id	first	last	age
1	4	3	Todd	Flanders	8
2	6	5	Stewie	Griffin	1
3	NULL	NULL	Ned	Flanders	47
4	NULL	NULL	Maude	Flanders	NULL
5	NULL	NULL	Peter	Griffin	42
6	NULL	NULL	Lois	Griffin	39

$\downarrow$

child_first	child_last	mother_first	mother_last	father_first	father_last
Todd	Flanders	Maude	Flanders	Ned	Flanders
Stewie	Griffin	Lois	Griffin	Peter	Griffin

Outer Joins

- Exactly like Inner Joins, except in the case where connections fail. When a NULL is encountered the empty data will be left in the resultset
- Outer joins also need their direction specified. They use LEFT JOIN or RIGHT JOIN for this purpose

Outer Joins

- With a LEFT JOIN, any data on the left side that doesn't have a corresponding row will stay in the resultset, i.e. NULLs on the right will be preserved
- With a RIGHT JOIN, data on the right side will be present in the resultset, regardless of the connection. This causes NULLs to appear in the original resultset

Compatibility

- For the purposes of this class, RIGHT JOINs will not be used as they are not fully supported by all databases

Outer Joins

```
SELECT *
FROM person AS child
LEFT JOIN person AS mother ON (mother.id = child.mother_id)
LEFT JOIN person AS father ON (father.id = child.father_id)
```

id	mother_id	father_id	first	last	age
1	4	3	Todd	Flanders	8
2	6	5	Stewie	Griffin	1
3	NULL	NULL	Ned	Flanders	47
4	NULL	NULL	Maude	Flanders	NULL
5	NULL	NULL	Peter	Griffin	42
6	NULL	NULL	Lois	Griffin	39

$\downarrow$

first	last	first	last	first	last
Todd	Flanders	Maude	Flanders	Ned	Flanders
Stewie	Griffin	Lois	Griffin	Peter	Griffin
Ned	Flanders	NULL	NULL	NULL	NULL
Maude	Flanders	NULL	NULL	NULL	NULL
Peter	Griffin	NULL	NULL	NULL	NULL
Lois	Griffin	NULL	NULL	NULL	NULL

Outer Joins

- Because outer joins never remove data from the resultset, they are excellent for statistics and data aggregation
- "All students and classes they are taking, *if any*"
- "All sales that these car models have had, *if any*"
- In these cases you would want to still have the original source intact in your data to track any non-participants or low-sellers

Reminders

- Assignment 3 up tonight. Due 9/24
- Lab time Wednesday in SSB 172
- Test Monday in this classroom on 9/24
- Additional office hours on Sunday in SSB 172, 2pm-4pm