An Introduction to Joins

DATA 604

Leanne Wu

lewu@ucalgary.ca

Department of Computer Science

Joins

- Joins (in MySQL, also called *table references*) are used to merge data between tables
- Tables built with foreign keys mean that joins can be used to navigate relationships between tables
- As in pandas, MySQL supports many join variations:
  - CROSS
  - NATURAL
  - LEFT/RIGHT
  - INNER

Cartesian (CROSS) Products – Revisiting for joins

- Given two relations R and S, combines every tuple of R with every tuple of S

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x1 (1)	y1 (2)	a2 (2)	b2 (4)
x2 (3)	y2 (4)	a1 (1)	b1 (3)
x2 (3)	y2 (4)	a2 (2)	b2 (4)

Cross Joins in MySQL

left_table

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)

right_table

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)

```
SELECT * FROM left_table CROSS JOIN right_table;
```

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x1 (1)	y1 (2)	a2 (2)	b2 (4)
x2 (3)	y2 (4)	a1 (1)	b1 (3)
x2 (3)	y2 (4)	a2 (2)	b2 (4)

Cross Joins in MySQL (with selection)

left_table

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)

right_table

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)

```
SELECT * FROM left_table CROSS JOIN right_table
        on (left_table.x = right_table.a);
```

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x1 (1)	y1 (2)	a2 (2)	b2 (4)
x2 (3)	y2 (4)	a1 (1)	b1 (3)
x2 (3)	y2 (4)	a2 (2)	b2 (4)

Natural Joins

left_table

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)

right_table

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)

```
SELECT * FROM left_table JOIN right_table
        ON (left_table.x = right_table.a);
```

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x1 (1)	y1 (2)	a2 (2)	b2 (4)
x2 (3)	y2 (4)	a1 (1)	b1 (3)
x2 (3)	y2 (4)	a2 (2)	b2 (4)

Left Joins

left_table

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)
x3 (5)	y3 (NULL)

right_table

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)
a3 (5)	b3 (NULL)

```
INSERT into left_table (x, y) VALUES (5, NULL);
INSERT into right_table (a, b) VALUES (5, NULL);

SELECT * FROM left_table LEFT JOIN right_table
         ON (left_table.x = right_table.a);
```

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x3 (5)	y3 (NULL)	a3 (5)	b3 (NULL)
x2 (3)	y2 (4)	NULL	NULL

Right Joins

left_table

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)
x3 (5)	y3 (NULL)

right_table

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)
a3 (5)	b3 (NULL)

```
SELECT * FROM left_table RIGHT JOIN right_table
        ON (left_table.x = right_table.a);
```

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x3 (5)	y3 (NULL)	a3 (5)	b3 (NULL)
NULL	NULL	a2 (2)	b2 (4)

Inner Joins

left_table

x	**y**
x1 (1)	y1 (2)
x2 (3)	y2 (4)
x3 (5)	y3 (NULL)

right_table

a	**b**
a1 (1)	b1 (3)
a2 (2)	b2 (4)
a3 (5)	b3 (NULL)

```
SELECT * FROM left_table INNER JOIN right_table
        ON (left_table.x = right_table.a);
```

x	y	a	b
x1 (1)	y1 (2)	a1 (1)	b1 (3)
x3 (5)	y3 (NULL)	a3 (5)	b3 (NULL)

Join Algorithms

- Nested loop join
  - most commonly used technique
  - tables are iterated through to find matching rows
  - not efficient
- Merge join
  - assumes tables are in sorted order
- Hash join
  - only works with selected types of columns

Joins and NoSQL

- What is our intuition about joins for documents in a document store?

Consider air_quality_index.json and GHG_emissions.json.

Propose an algorithm which could join these documents.