**Mode: The SQL Tutorial for Data Analysis**

# Basic:

SELECT \*

FROM tutorial.us\_housing\_units

SELECT year,

month,

west

FROM tutorial.us\_housing\_units

SELECT west AS West\_Region,

south AS South\_Region

FROM tutorial.us\_housing\_units

-- Using Limits

SELECT \*

FROM tutorial.us\_housing\_units

LIMIT 100

-- Using where clause

SELECT \*

FROM tutorial.us\_housing\_units

WHERE month = 1

-- SQL Comparison Operators

SELECT \*

FROM tutorial.us\_housing\_units

WHERE west > 30

-- non-numerical data

SELECT \*

FROM tutorial.us\_housing\_units

WHERE month\_name != 'January'

SELECT \*

FROM tutorial.us\_housing\_units

WHERE month\_name > 'January'

-- Arithmetic in SQL

SELECT year,

month,

west,

south,

west + south AS south\_plus\_west

FROM tutorial.us\_housing\_units

SELECT year,

month,

west,

south,

west + south - 4 \* year AS nonsense\_column

FROM tutorial.us\_housing\_units

SELECT year,

month,

west,

south,

(west + south)/2 AS south\_west\_avg

FROM tutorial.us\_housing\_units

-- SQL Logical Operators

SELECT \* FROM tutorial.billboard\_top\_100\_year\_end

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

ORDER BY year DESC, year\_rank

-- SQL Like

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE "group\_name" LIKE 'Snoop%'

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE "group\_name" ILIKE 'snoop%

-- SQL IN Operator

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year\_rank IN (1, 2, 3)

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE artist IN ('Taylor Swift', 'Usher', 'Ludacris')

-- SQL Between operator

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year\_rank BETWEEN 5 AND 10

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year\_rank >= 5 AND year\_rank <= 10

-- IS NULL

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE artist IS NULL

-- AND

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2012 AND year\_rank <= 10

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2012

AND year\_rank <= 10

AND "group\_name" ILIKE '%feat%'

-- OR

-- % returns values the contains the specified word or more

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2013

AND ("group\_name" ILIKE '%macklemore%' OR "group\_name" ILIKE '%timberlake%')

-- SQL NOT

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2013

AND year\_rank NOT BETWEEN 2 AND 3

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2013

AND "group\_name" NOT ILIKE '%macklemore%'

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2013

AND artist IS NOT NULL

-- Order by

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

ORDER BY artist

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2013

ORDER BY year\_rank

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year = 2013

ORDER BY year\_rank DESC

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year\_rank <= 3

ORDER BY year DESC, year\_rank

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year\_rank <= 3

ORDER BY year\_rank, year DESC

SELECT \*

FROM tutorial.billboard\_top\_100\_year\_end

WHERE year\_rank <= 3

ORDER BY 2, 1 DESC

# Advanced:

-- SQL Data types

/\*

Imported as Stored as With these rules

String VARCHAR(1024) Any characters, with a maximum field length of 1024 characters.

Date/Time TIMESTAMP Stores year, month, day, hour, minute and second values as YYYY-MM-DD hh:mm:ss.

Number DOUBLE PRECISION Numerical, with up to 17 significant digits decimal precision.

Boolean BOOLEAN Only TRUE or FALSE values.

\*/

-- Data Format

SELECT permalink,

founded\_at,

founded\_at\_clean

FROM tutorial.crunchbase\_companies\_clean\_date

ORDER BY founded\_at\_clean

-- In this example, you can see that the time\_to\_acquisition column is an interval, not another date.

-- You can introduce intervals using the INTERVAL function as well:

SELECT companies.permalink,

companies.founded\_at\_clean,

acquisitions.acquired\_at\_cleaned,

acquisitions.acquired\_at\_cleaned -

companies.founded\_at\_clean::timestamp AS time\_to\_acquisition

FROM tutorial.crunchbase\_companies\_clean\_date companies

JOIN tutorial.crunchbase\_acquisitions\_clean\_date acquisitions

ON acquisitions.company\_permalink = companies.permalink

WHERE founded\_at\_clean IS NOT NULL

-- Data Wrangling with SQL

/\*

Data munging or data wrangling is loosely the process of manually converting or mapping data from one "raw"

form into another format that allows for more convenient consumption of the data with the help of semi-automated tools

\*/

-- Using SQL String Functions to Clean Data

/\*

You can use LEFT to pull a certain number of characters from the left side of a string and present them as a

separate string. The syntax is LEFT(string, number of characters)

\*/

SELECT incidnt\_num,

date,

LEFT(date, 10) AS cleaned\_date,

RIGHT(date, 17) AS cleaned\_time

FROM tutorial.sf\_crime\_incidents\_2014\_01

SELECT incidnt\_num,

date,

LEFT(date, 10) AS cleaned\_date,

RIGHT(date, LENGTH(date) - 11) AS cleaned\_time

FROM tutorial.sf\_crime\_incidents\_2014\_01

-- Trim; The TRIM function is used to remove characters from the beginning and end of a string

SELECT location,

TRIM(both '()' FROM location)

FROM tutorial.sf\_crime\_incidents\_2014\_01

-- POSITION and STRPOS

/\*

Position allows you to specify a substring, then returns a numerical value equal to the character

number (counting from left) where that substring first appears in the target string.

For example, the following query will return the position of the character 'A' (case-sensitive) where

it first appears in the descript field:

\*/

SELECT incidnt\_num,

descript,

POSITION('A' IN descript) AS a\_position

FROM tutorial.sf\_crime\_incidents\_2014\_01

/\*

You can also use the STRPOS function to achieve the same results—just replace IN with a

comma and switch the order of the string and substring:

\*/

SELECT incidnt\_num,

descript,

STRPOS(descript, 'A') AS a\_position

FROM tutorial.sf\_crime\_incidents\_2014\_01

-- SUBSTR

SELECT incidnt\_num,

date,

SUBSTR(date, 4, 2) AS day

FROM tutorial.sf\_crime\_incidents\_2014\_01

-- CONCAT

SELECT incidnt\_num,

day\_of\_week,

LEFT(date, 10) AS cleaned\_date,

CONCAT(day\_of\_week, ', ', LEFT(date, 10)) AS day\_and\_date

FROM tutorial.sf\_crime\_incidents\_2014\_01

-- Changing case with UPPER and LOWER

SELECT incidnt\_num,

address,

UPPER(address) AS address\_upper,

LOWER(address) AS address\_lower

FROM tutorial.sf\_crime\_incidents\_2014\_01

-- Making dates cleaner

SELECT cleaned\_date,

EXTRACT('year' FROM cleaned\_date) AS year,

EXTRACT('month' FROM cleaned\_date) AS month,

EXTRACT('day' FROM cleaned\_date) AS day,

EXTRACT('hour' FROM cleaned\_date) AS hour,

EXTRACT('minute' FROM cleaned\_date) AS minute,

EXTRACT('second' FROM cleaned\_date) AS second,

EXTRACT('decade' FROM cleaned\_date) AS decade,

EXTRACT('dow' FROM cleaned\_date) AS day\_of\_week

FROM tutorial.sf\_crime\_incidents\_cleandate

-- COALESCE

SELECT incidnt\_num,

descript,

COALESCE(descript, 'No Description')

FROM tutorial.sf\_crime\_incidents\_cleandate

ORDER BY descript DESC

-- Writing Subqueries in SQL

SELECT sub.\*

FROM (

SELECT \*

FROM tutorial.sf\_crime\_incidents\_2014\_01

WHERE day\_of\_week = 'Friday'

) sub

WHERE sub.resolution = 'NONE'

/\*

What if you wanted to figure out how many incidents get reported on each day of the week? Better yet,

what if you wanted to know how many incidents happen, on average, on a Friday in December? In January?

\*/

SELECT LEFT(sub.date, 2) AS cleaned\_month,

sub.day\_of\_week,

AVG(sub.incidents) AS average\_incidents

FROM (

SELECT day\_of\_week,

date,

COUNT(incidnt\_num) AS incidents

FROM tutorial.sf\_crime\_incidents\_2014\_01

GROUP BY 1,2

) sub

GROUP BY 1,2

ORDER BY 1,2

-- subqueries in conditional logic

SELECT \*

FROM tutorial.sf\_crime\_incidents\_2014\_01

WHERE Date = (SELECT MIN(date)

FROM tutorial.sf\_crime\_incidents\_2014\_01

)

-- Joining subqueries

SELECT incidents.\*,

sub.incidents AS incidents\_that\_day

FROM tutorial.sf\_crime\_incidents\_2014\_01 incidents

JOIN ( SELECT date,

COUNT(incidnt\_num) AS incidents

FROM tutorial.sf\_crime\_incidents\_2014\_01

GROUP BY 1

) sub

ON incidents.date = sub.date

ORDER BY sub.incidents DESC, time

-- sub queries and unions

SELECT \*

FROM tutorial.crunchbase\_investments\_part1

UNION ALL

SELECT \*

FROM tutorial.crunchbase\_investments\_part2

-- SQL Window Functions

/\*

A window function performs a calculation across a set of table rows that are somehow related to the current row.

This is comparable to the type of calculation that can be done with an aggregate function.

But unlike regular aggregate functions, use of a window function does not cause rows to become grouped into a single

output row — the rows retain their separate identities.

\*/

SELECT duration\_seconds,

SUM(duration\_seconds) OVER (ORDER BY start\_time) AS running\_total

FROM tutorial.dc\_bikeshare\_q1\_2012

-- Basic Windowing Syntax

SELECT start\_terminal,

duration\_seconds,

SUM(duration\_seconds) OVER

(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_total

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

-- The usual suspects: SUM, COUNT, and AVG

SELECT start\_terminal,

duration\_seconds,

SUM(duration\_seconds) OVER

(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_total,

COUNT(duration\_seconds) OVER

(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_count,

AVG(duration\_seconds) OVER

(PARTITION BY start\_terminal ORDER BY start\_time)

AS running\_avg

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

-- Row\_number; ROW\_NUMBER() does just what it sounds like—displays the number of a given row.

SELECT start\_terminal,

start\_time,

duration\_seconds,

ROW\_NUMBER() OVER (ORDER BY start\_time)

AS row\_number

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

-- Using the PARTITION BY clause will allow you to begin counting 1 again in each partition

SELECT start\_terminal,

start\_time,

duration\_seconds,

ROW\_NUMBER() OVER (PARTITION BY start\_terminal

ORDER BY start\_time)

AS row\_number

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

-- RANK() and DENSE\_RANK()

/\*

RANK() is slightly different from ROW\_NUMBER(). If you order by start\_time, for example, it might be the case that

some terminals have rides with two identical start times. In this case, they are given the same rank, whereas ROW\_NUMBER()

gives them different numbers. In the following query, you notice the 4th and 5th observations for start\_terminal 31000—they

are both given a rank of 4, and the following result receives a rank of 6:

\*/

SELECT start\_terminal,

duration\_seconds,

RANK() OVER (PARTITION BY start\_terminal

ORDER BY start\_time)

AS rank

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

-- NTILE

/\*

You can use window functions to identify what percentile (or quartile, or any other subdivision) a given row falls into.

The syntax is NTILE(\*# of buckets\*). In this case, ORDER BY determines which column to use to determine the quartiles

(or whatever number of 'tiles you specify). For example:

\*/

SELECT start\_terminal,

duration\_seconds,

NTILE(4) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds)

AS quartile,

NTILE(5) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds)

AS quintile,

NTILE(100) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds)

AS percentile

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

ORDER BY start\_terminal, duration\_seconds

-- LAG and LEAD

/\*

It can often be useful to compare rows to preceding or following rows, especially if you've got the data in an order

that makes sense. You can use LAG or LEAD to create columns that pull values from other rows—all you need to do is

enter which column to pull from and how many rows away you'd like to do the pull. LAG pulls from previous rows and LEAD

pulls from following rows:

\*/

SELECT start\_terminal,

duration\_seconds,

LAG(duration\_seconds, 1) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds) AS lag,

LEAD(duration\_seconds, 1) OVER

(PARTITION BY start\_terminal ORDER BY duration\_seconds) AS lead

FROM tutorial.dc\_bikeshare\_q1\_2012

WHERE start\_time < '2012-01-08'

ORDER BY start\_terminal, duration\_seconds

-- Performance Tuning SQL Queries

-- Reducing table size

SELECT \*

FROM benn.sample\_event\_table

WHERE event\_date >= '2014-03-01'

AND event\_date < '2014-04-01'

SELECT COUNT(\*)

FROM benn.sample\_event\_table

LIMIT 100

-- Making joins less complicated

SELECT teams.conference AS conference,

players.school\_name,

COUNT(1) AS players

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

ON teams.school\_name = players.school\_name

GROUP BY 1,2

--

SELECT players.school\_name,

COUNT(\*) AS players

FROM benn.college\_football\_players players

GROUP BY 1

--

SELECT teams.conference,

sub.\*

FROM (

SELECT players.school\_name,

COUNT(\*) AS players

FROM benn.college\_football\_players players

GROUP BY 1

) sub

JOIN benn.college\_football\_teams teams

ON teams.school\_name = sub.school\_name

-- Pivoting Data in SQL

-- This makes the data neater and more insightful

SELECT teams.conference AS conference,

players.year,

COUNT(1) AS players

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

ON teams.school\_name = players.school\_name

GROUP BY 1,2

ORDER BY 1,2

--

SELECT conference,

SUM(players) AS total\_players,

SUM(CASE WHEN year = 'FR' THEN players ELSE NULL END) AS fr,

SUM(CASE WHEN year = 'SO' THEN players ELSE NULL END) AS so,

SUM(CASE WHEN year = 'JR' THEN players ELSE NULL END) AS jr,

SUM(CASE WHEN year = 'SR' THEN players ELSE NULL END) AS sr

FROM (

SELECT teams.conference AS conference,

players.year,

COUNT(1) AS players

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

ON teams.school\_name = players.school\_name

GROUP BY 1,2

) sub

GROUP BY 1

ORDER BY 2 DESC

-- Another example

SELECT years.\*,

earthquakes.magnitude,

CASE year

WHEN 2000 THEN year\_2000

WHEN 2001 THEN year\_2001

WHEN 2002 THEN year\_2002

WHEN 2003 THEN year\_2003

WHEN 2004 THEN year\_2004

WHEN 2005 THEN year\_2005

WHEN 2006 THEN year\_2006

WHEN 2007 THEN year\_2007

WHEN 2008 THEN year\_2008

WHEN 2009 THEN year\_2009

WHEN 2010 THEN year\_2010

WHEN 2011 THEN year\_2011

WHEN 2012 THEN year\_2012

ELSE NULL END

AS number\_of\_earthquakes

FROM tutorial.worldwide\_earthquakes earthquakes

CROSS JOIN (

SELECT year

FROM (VALUES (2000),(2001),(2002),(2003),(2004),(2005),(2006),

(2007),(2008),(2009),(2010),(2011),(2012)) v(year)

) years

# Intermediate:

-- SQL Count

SELECT COUNT(\*)

FROM tutorial.aapl\_historical\_stock\_price

SELECT COUNT(high)

FROM tutorial.aapl\_historical\_stock\_price

-- Counting non-numerical coloumns

SELECT COUNT(date) AS count\_of\_date

FROM tutorial.aapl\_historical\_stock\_price

-- SQL SUM

SELECT SUM(volume)

FROM tutorial.aapl\_historical\_stock\_price

-- Min/Max

SELECT MIN(volume) AS min\_volume,

MAX(volume) AS max\_volume

FROM tutorial.aapl\_historical\_stock\_price

-- Avg function

SELECT AVG(high)

FROM tutorial.aapl\_historical\_stock\_price

WHERE high IS NOT NULL

-- The above query produces the same result as the following query:

SELECT AVG(high)

FROM tutorial.aapl\_historical\_stock\_price

-- Group By

SELECT year,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY year

-- group by multiple group

SELECT year,

month,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY year, month

-- Group by column numbers

SELECT year,

month,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY 1, 2

SELECT year,

month,

COUNT(\*) AS count

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY year, month

ORDER BY month, year

-- Having Clause

SELECT year,

month,

MAX(high) AS month\_high

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY year, month

HAVING MAX(high) > 400

ORDER BY year, month

-- Case

SELECT player\_name,

year,

CASE WHEN year = 'SR' THEN 'yes'

ELSE NULL END AS is\_a\_senior

FROM benn.college\_football\_players

SELECT player\_name,

weight,

CASE WHEN weight > 250 THEN 'over 250'

WHEN weight > 200 THEN '201-250'

WHEN weight > 175 THEN '176-200'

ELSE '175 or under' END AS weight\_group

FROM benn.college\_football\_players

SELECT player\_name,

weight,

CASE WHEN weight > 250 THEN 'over 250'

WHEN weight > 200 AND weight <= 250 THEN '201-250'

WHEN weight > 175 AND weight <= 200 THEN '176-200'

ELSE '175 or under' END AS weight\_group

FROM benn.college\_football\_players

-- Case with aggregates

SELECT CASE WHEN year = 'FR' THEN 'FR'

ELSE 'Not FR' END AS year\_group,

COUNT(1) AS count

FROM benn.college\_football\_players

GROUP BY CASE WHEN year = 'FR' THEN 'FR'

ELSE 'Not FR' END

SELECT COUNT(CASE WHEN year = 'FR' THEN 1 ELSE NULL END) AS fr\_count,

COUNT(CASE WHEN year = 'SO' THEN 1 ELSE NULL END) AS so\_count,

COUNT(CASE WHEN year = 'JR' THEN 1 ELSE NULL END) AS jr\_count,

COUNT(CASE WHEN year = 'SR' THEN 1 ELSE NULL END) AS sr\_count

FROM benn.college\_football\_players

-- SQL Distinct

SELECT DISTINCT year, month

FROM tutorial.aapl\_historical\_stock\_price

-- Using Distinct in aggregations

SELECT month,

AVG(volume) AS avg\_trade\_volume

FROM tutorial.aapl\_historical\_stock\_price

GROUP BY month

ORDER BY 2 DESC

-- Joins

SELECT teams.conference AS conference,

AVG(players.weight) AS average\_weight

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

ON teams.school\_name = players.school\_name

GROUP BY teams.conference

ORDER BY AVG(players.weight) DESC

SELECT \*

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

ON teams.school\_name = players.school\_name

-- Inner Join

SELECT players.\*,

teams.\*

FROM benn.college\_football\_players players

JOIN benn.college\_football\_teams teams

ON teams.school\_name = players.school\_name

-- Outer Joins

--Left Join

SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM tutorial.crunchbase\_companies companies

LEFT JOIN tutorial.crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink

--Right Join

SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM tutorial.crunchbase\_acquisitions acquisitions

RIGHT JOIN tutorial.crunchbase\_companies companies

ON companies.permalink = acquisitions.company\_permalink

-- will produce the same results as the left join in this case

-- SQL Joins Using WHERE or ON

SELECT companies.permalink AS companies\_permalink,

companies.name AS companies\_name,

acquisitions.company\_permalink AS acquisitions\_permalink,

acquisitions.acquired\_at AS acquired\_date

FROM tutorial.crunchbase\_companies companies

LEFT JOIN tutorial.crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink

WHERE acquisitions.company\_permalink != '/company/1000memories'

OR acquisitions.company\_permalink IS NULL

ORDER BY 1

-- Full outer join

SELECT COUNT(CASE WHEN companies.permalink IS NOT NULL AND acquisitions.company\_permalink IS NULL

THEN companies.permalink ELSE NULL END) AS companies\_only,

COUNT(CASE WHEN companies.permalink IS NOT NULL AND acquisitions.company\_permalink IS NOT NULL

THEN companies.permalink ELSE NULL END) AS both\_tables,

COUNT(CASE WHEN companies.permalink IS NULL AND acquisitions.company\_permalink IS NOT NULL

THEN acquisitions.company\_permalink ELSE NULL END) AS acquisitions\_only

FROM tutorial.crunchbase\_companies companies

FULL JOIN tutorial.crunchbase\_acquisitions acquisitions

ON companies.permalink = acquisitions.company\_permalink

-- Union

-- SQL joins allow you to combine two datasets side-by-side, but UNION allows you to stack one dataset on top of the other

SELECT \*

FROM tutorial.crunchbase\_investments\_part1

UNION

SELECT \*

FROM tutorial.crunchbase\_investments\_part2

-- SQL Joins with Comparison Operators

SELECT companies.permalink,

companies.name,

companies.status,

COUNT(investments.investor\_permalink) AS investors

FROM tutorial.crunchbase\_companies companies

LEFT JOIN tutorial.crunchbase\_investments\_part1 investments

ON companies.permalink = investments.company\_permalink

WHERE investments.funded\_year > companies.founded\_year + 5

GROUP BY 1,2, 3

-- SQL Joins on Multiple Keys

SELECT companies.permalink,

companies.name,

investments.company\_name,

investments.company\_permalink

FROM tutorial.crunchbase\_companies companies

LEFT JOIN tutorial.crunchbase\_investments\_part1 investments

ON companies.permalink = investments.company\_permalink

AND companies.name = investments.company\_name

-- Self Joins

SELECT DISTINCT japan\_investments.company\_name,

japan\_investments.company\_permalink

FROM tutorial.crunchbase\_investments\_part1 japan\_investments

JOIN tutorial.crunchbase\_investments\_part1 gb\_investments

ON japan\_investments.company\_name = gb\_investments.company\_name

AND gb\_investments.investor\_country\_code = 'GBR'

AND gb\_investments.funded\_at > japan\_investments.funded\_at

WHERE japan\_investments.investor\_country\_code = 'JPN'

ORDER BY 1