SQL 102 – Beginners Guide to SQL for Clinical Data

ARCUS Education

Children's Hospital of Philadelphia

Today's Itinerary

We will be touching on the following SQL topics:

- Case Statements
- Aggregate Functions
- String ("Text") Functions & Filters
- Joins – ("Left" & 'Inner" only for now)

Children's Hospital of Philadelphia®

Quick PSA about "Text Editors”

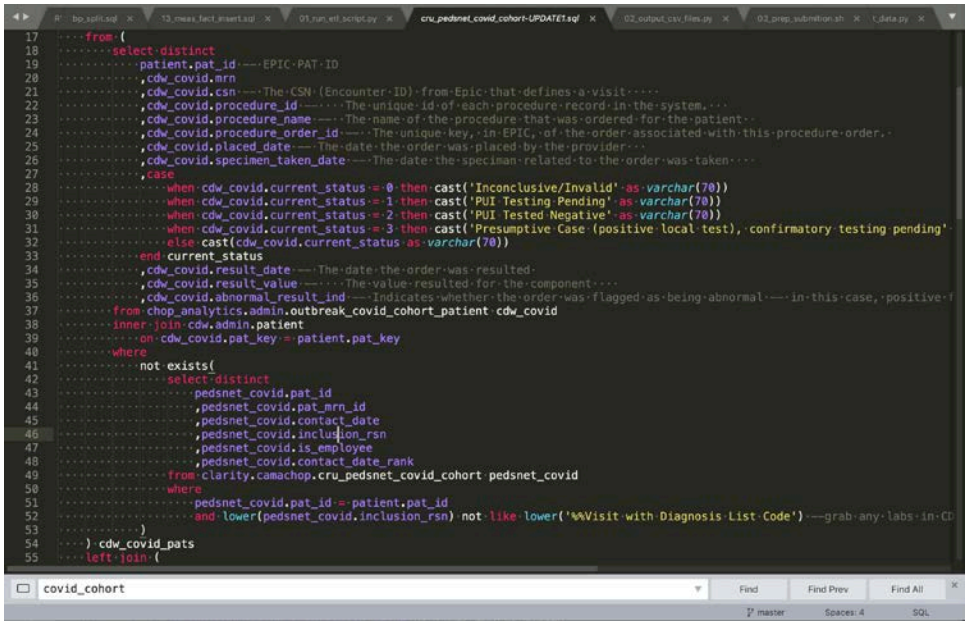
When Viewing or Editing Code it is a good idea to us a text editor specifically designed for working with Code (rather than trying to save your code into something like a word file).

Quick Plug for Sublime Text

Sublime Text is a simple, easy to use, open source text/code editor that is both powerful and easy on the eyes. Its Free! (**despite the occasional pop-up that will ask you if you want to buy a license; but you can say no every time)

App Install Link:

https://www.sublimetext.com/3



Children's Hospital of Philadelphia®

Sample Data

For most of todays lesson we will be using the following **SQLite Dataset**

Download **synthea_sample.db** file from the below sharefile link:

link https://chop.sharefile.com/d-s57101c68c9148cc9

You can navigate this sample database by using the “DB Browser for SQLite” application (download link below):

link https://sqlitebrowser.org/dl/

Children's Hospital of Philadelphia®

Case Statements

CASE WHEN [...] THEN [...] ELSE [...] END

```
SELECT
     p.id as pat_id
    ,case
        when p.gender ='M' then 'Male'
        when p.gender = 'F' then 'Female'
        else null
    end as sex
    , p.race
    , p.ethnicity
    ,case when p.ethnicity = 'hispanic' then 1 end is_hispanic_ind
FROM patients as p
WHERE p.city = 'Boston'
```

Children's Hospital of Philadelphia®

Aggregate Functions

SQL Aggregate Functions:

COUNT() – counts rows in a specified table or view.
SUM() – calculates the sum of values.
MIN() – gets the minimum value in a set of values.
MAX() – gets the maximum value in a set of values.
AVG() – calculates the average of a set of values.

Anytime you use an aggregate function you will (*almost always*) also need to use the **GROU BY** clause to let you database know what "level" you want it to aggregate your data on.

You can also use the **HAVING** clause anytime **GROU BY** is used.

This **HAVING** clause will let you preform filtering on the results of your Aggregate Functions (since you'll notice you can't reference them in the **WHERE** clause

```
select
    patients.state
    ,count(patients.id) patient_count
from patients
where
    patients.gender = 'M'
group by
    patients.state
having
    count(patients.id) > 100
```

Children's Hospital of Philadelphia®

Aggregate Functions

```
-- Example #2 – Patient “Wellness Visit” Count.
select
    patient as patient_id
    ,sum(case when encounterclass = 'wellness' then 1 else 0 end) wellness_visit_count
from ENCOUNTERS
where start > '2019-01-01'
group by
    patient_id
having
    wellness_visit_count > 1
order by
    wellness_visit_count
```

Children's Hospital of Philadelphia®

String Functions & Filters

The LIKE Operator can be used for “fuzzy” text filtering

- **%** The percent sign represents zero, one, or multiple characters (i.e. it’s a wild card)
- **_** The underscore can be used to represents any single character

```
select
    count(distinct patient) patients_with_egg_allergy
from allergies as a
where
     lower(a.description) LIKE '%egg%'  --text like “egg”
    and a.stop = '' --blank stop date (i.e. active)
```

Aside for this LIKE operator, there are lots of other useful string manipulation functions exist that you can used based on the need:

- Extract a specific sub-section from an existing string (see **substr()** function)
- Replace any character(s) from a string with something else (see **replace()** function)

Joins

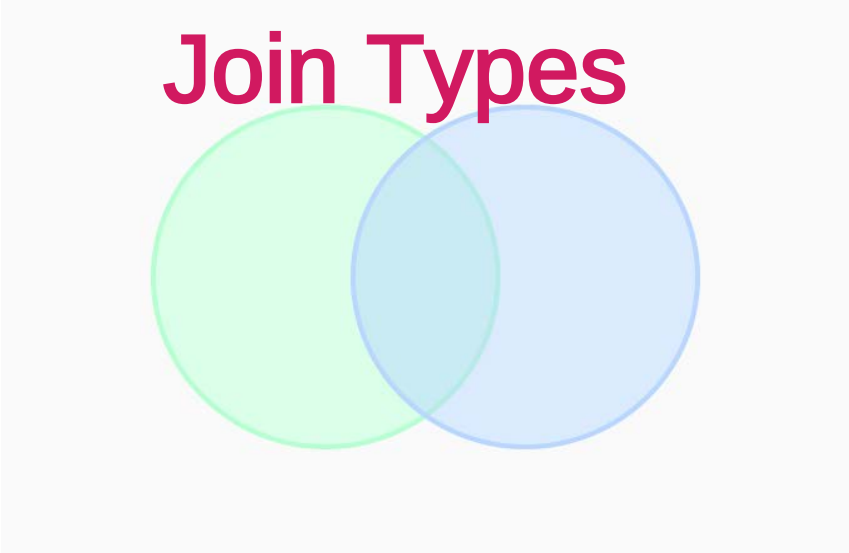
Most queries require something more complex than the examples shown above where we are only referencing a single table.

You may need to **combine data from two or more tables**. This is where SQL's **JOIN** functionality comes into play.

There are two basic things you need for a successful join:

- **Join Criteria**:
  - What columns between your 2 input tables has to match in order for the join to be a match?
    - Such as Patient MRN, Account Number, Etc.
- **Join Type**:
  - What Type of Join do you want to use (inner, left, full, etc)?
    - This will effect the total number of rows returned by your join

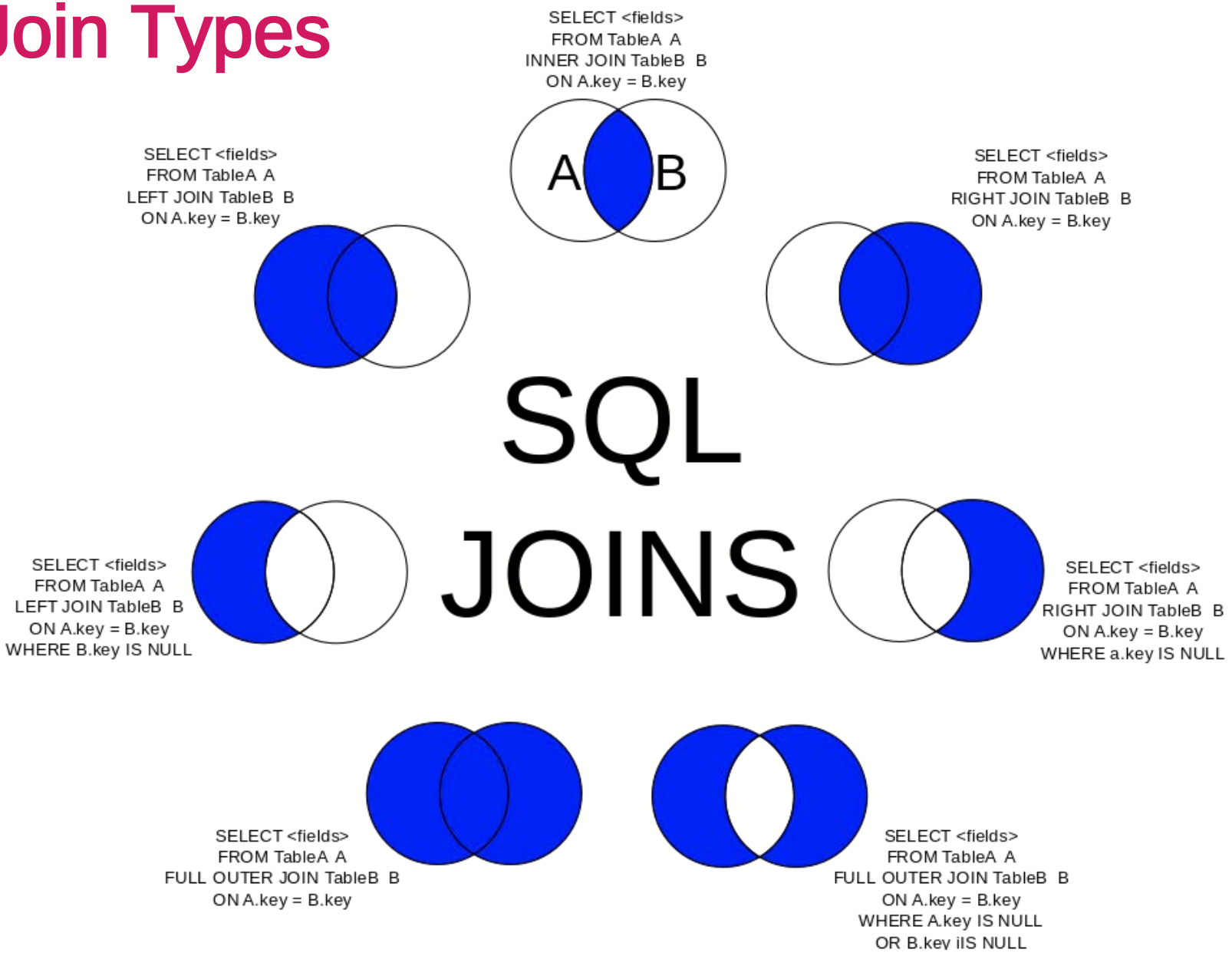
Join Types



You can imagine a join as one of the combinations of a Venn diagram. If you're joining two tables, you can choose:

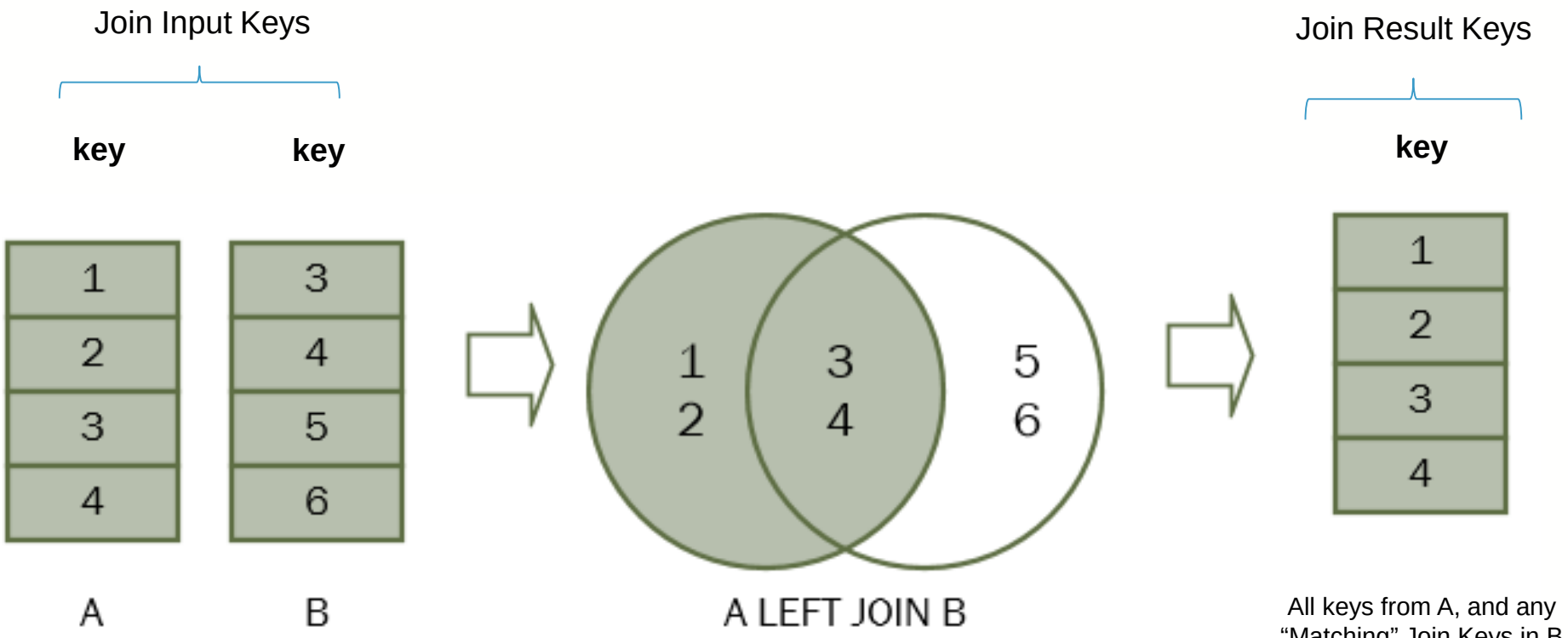
- Just the intersection, or an "**inner**" **join** – default "**join**" behavior
- The portion of the circle on the left as well as the intersection, known as a "**left**" **join**
- Or all three sections of the diagram, know as a "**full**" **join**

Join Types



Children's Hospital of Philadelphia®

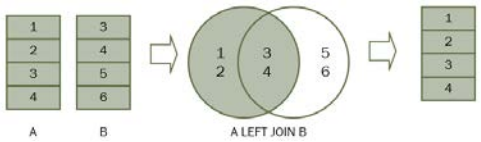
Left Join




```
select *
from A
left join B
    on A.key = B.key
```

Children's Hospital of Philadelphia®

Left Join



What does the result of the joined tables "look like"?

Tables are then "appended" to on another based on the **Join Keys** provided

And because we used a **LEFT JOIN,** all rows of **Table A** along with any rows from **Table B** with "overlapping Join Keys" will be returned.

key	col_1	col_2
1	This	is
2	fake	data
3	blah	blah
4	yata	yata

key	col_4	col_5
3	black	sheep
4	bata	bing
5	bata	boom
6	multi	pass

key	col_1	col_2	key	col_4	col_5
1	This	is			
2	fake	data			
3	blah	blah	**3**	black	sheep
4	yata	yata	**4**	bata	bing

```
select *
from A
left join B
    on A.key = B.key
```

Children's Hospital of Philadelphia®

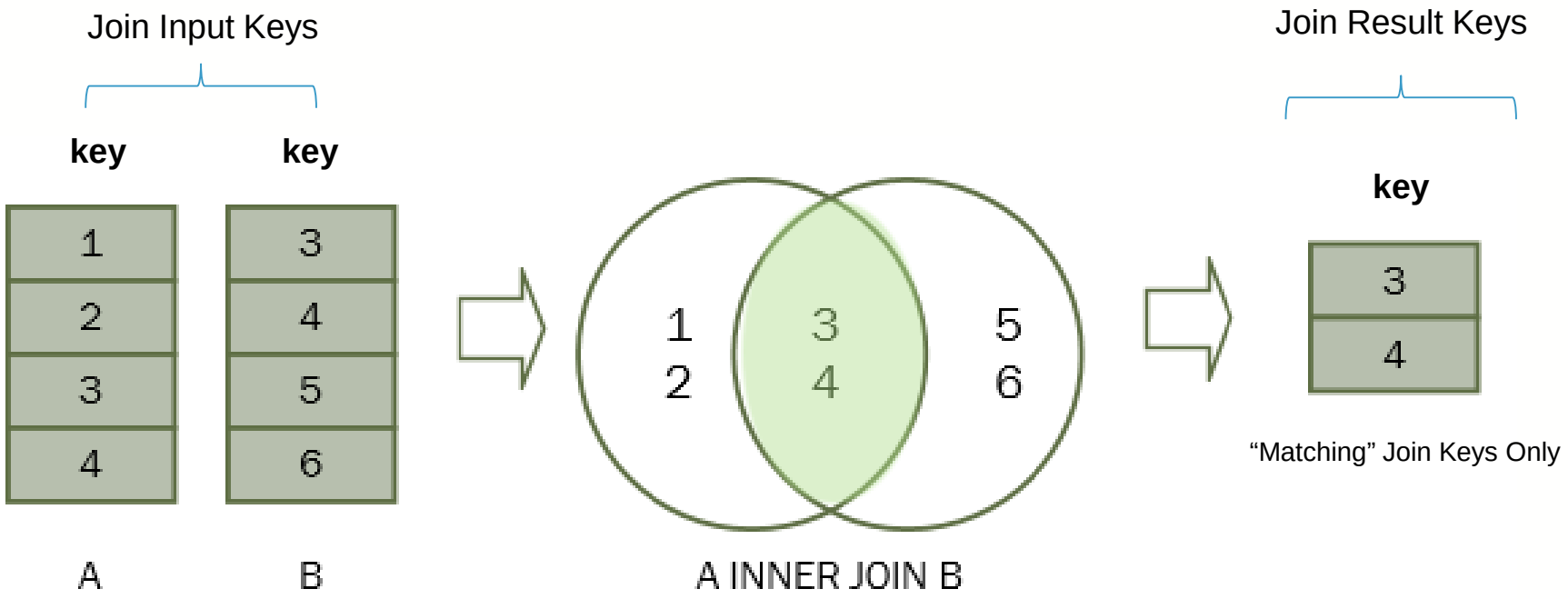
Left Join Example

Synthea Data

```
Select distinct
        Patients.id         as   patient_id
        , patients.gender
        , patients.birthdate
        ,count   (*) as       num_encounters
from    patients
left join          encounters      as   e
        on   e.patient        =   patients.id
group by
        patients.id
        , patients.gender
        , patients.birthdate
order by
        count(*)       desc
```

Children's Hospital of Philadelphia®

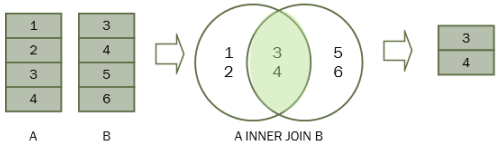
Inner Join




```
select *
from A
inner join B
    on A.key = B.key
```

Children's Hospital of Philadelphia®

Inner Join



What does the result of the joined tables "look like"?

Tables are then "appended" to on another based on the **Join Keys** provided

And only rows with "overlapping Join Keys" are returned (since this is an **Inner Join**).

key	col_1	col_2
1	This	is
2	fake	data
3	blah	blah
4	yata	yata

key	col_4	col_5
3	black	sheep
4	bata	bing
5	bata	boom
6	multi	pass

key	col_1	col_2	key	col_4	col_5
3	blah	blah	**3**	black	sheep
4	yata	yata	**4**	bata	bing

Can you see the difference between this and the left join result set?

```
select *
from A
inner join B
    on A.key = B.key
```

Children's Hospital of Philadelphia®

Inner Join Example

Synthea Data

```
SELECT DISTINCT
        patients.id         as   patient_id
        , encounters.start
        , encounters.encounterclass
FROM patients
INNER JOIN    encounters
        ON patients.id          =   encounters.patient
WHERE
        patients.city           = 'Boston'
```

Children's Hospital of Philadelphia®

Cartesian Join

A “**Cartesian**” Join is essentially a join that “results in more rows than it started with”.

This can be done intentionally by using something called a “**cross**” join, however it can also occur un-intentionally if your join criteria result in an n_A: n_B relationship (i.e. if its not a 1:1 or 1:n mapping).

“**Cartesian**” Joins can be very memory intensive operations as the size of the result set increases exponentially with the size of their 2 input tables. Additionally, if they go “unnoticed”, they can seriously effect the validity of your SQL reports (e.g. if your trying to infer a count of something by counting the number of rows, an unintentional cartesian join will result in you overcounting the value you were trying to measure)

That said, unless you have a super good reaso n, you should **never perform a join that isn’t 1:1 or 1:n.**

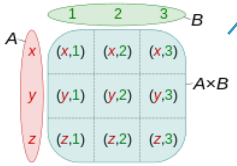
A

n
1
2
3

B

c
x
y
z

```
SELECT *
FROM A
CROSS JOIN B
```



“Cartesian Product”

A x B

n	c
1	x
1	y
1	z
2	x
2	y
2	z
3	x
3	y
3	z

Children’s Hospital of Philadelphia®

Cross Joins
"Stable" vs "Un - Stable"

"**Stable**" Cross Join (n:1)

```
select count(*)
from allergies as a
cross join (  --1 row table (results in a n:1 level join).
    select '2020-05-19' today
) b
where
    b.today between
        start
        and coalesce( –if no end date, supply default "far future" stop date.
                (case when stop='' then null else stop end) –replacing "blank" value with null
                , '2199-12-31'
        )
```

count(*) $\Leftrightarrow \sum_1^N a_i * b(i) = \sum_1^N a_i * 1$ = **N** rows

Where,

- a_i = row ***i*** from table ***a***
- $b(i)$ = rows from table ***b*** that match join key ***I***
- ***N*** = *number of rows in table with the most rows*

"**Un-Stable**" Cross Join (n:m)

```
select count(*)
from allergies as a
cross join ( --m row table (results in a n:m level join).
    select 'x' col1
    union
    select 'y' col1
) b
```

count(*) $\Leftrightarrow \sum_1^N a_i * b(i) = \sum_1^N a_i * 2$ = **2*N** rows

Where,

- a_i = row ***i*** from table ***a***
- $b(i)$ = rows from table ***b*** that match join key ***I***
- ***N*** = *number of rows in table with the most rows*

Children's Hospital of Philadelphia®

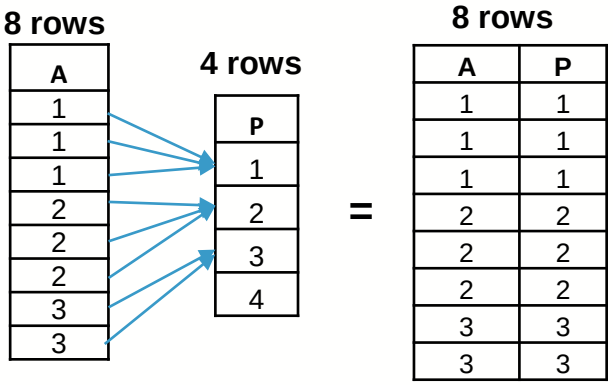
“Stable” Joins vs. “ Cartesian ” Joins

1:n (and **n:1**) joins can be considered “Stable” because the result set of this join will have the same number of rows of as “the join input table with the most rows”

“**Stable**” Join (n:1)

```
select count(*)
from allergies as a
inner join patients as p
    on a.patient = p.id --this is an n:1 level join
```

count(*) ⇔ $\sum_1^N a_i * 1$ **≤ N** rows



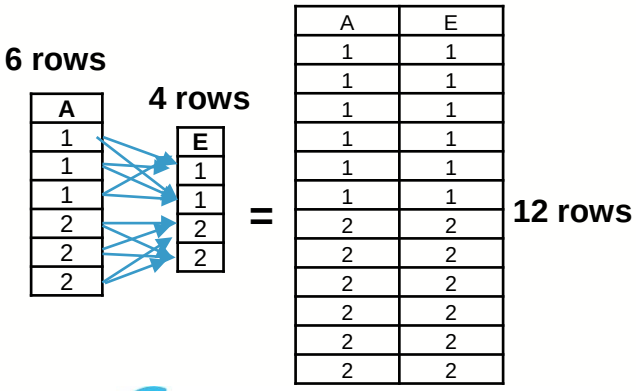
“**Cartesian**” Join (n:m)

n:m (where n>1 & m>1) joins are referred to as **Cartesian** Joins.

The final result set of a Cartesian join will have a total row count less than or equal to the “product of the number of rows in both input tables

```
select count(*)
from allergies as a
inner join encounters as b
    on a.patient = b.patient --this is an n:m level join
```

count(*) ⇔ $\sum_1^N a_i * b(i)$ **≤ N*M** rows



Children's Hospital of Philadelphia®

Looking at Data in the ADR

We can actually take a look at this ourselves (Provided we have the necessary CITI training completed) from the ARCUS application web portal (only available from CHOP network):

Lets check out the site! https://app.arcus.chop.edu

Follow Along with me:

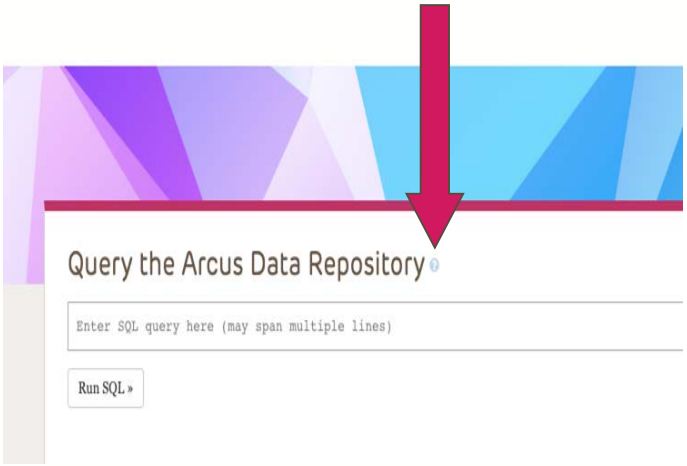
- Log in (button in upper right corner of the page)
- Sign the terms of use
- Go to the “Data Repository Query” tab

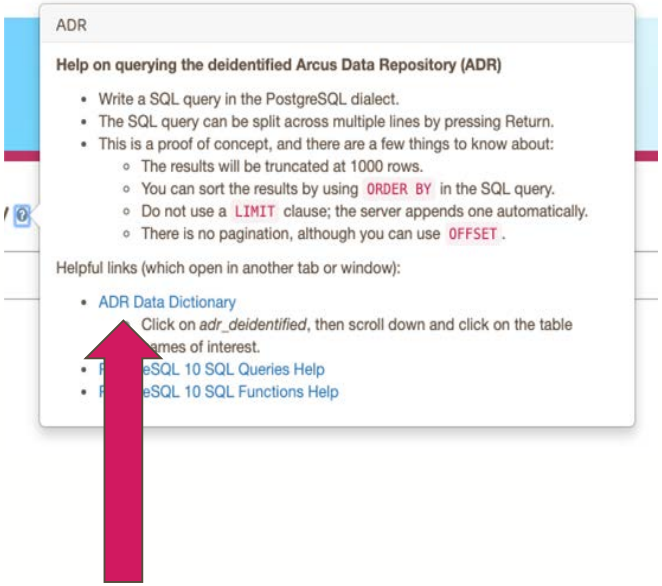
Children's Hospital of Philadelphia®

Looking at Data in the ADR

You can preview the data schema (the shape of the data) by following the “question mark” path in the same tab.

Hover over that question mark icon, then click on the link for the “ADR Data Dictionary”.





Children's Hospital of Philadelphia®

A Realistic Join Query on **ARCUS Data**

The below example query returns a count of all patients with active ”Complex Chronic Conditions”

```
--example #1 : CCC Patients Query Template
select
    count(distinct patient.pat_id) ccc_patient_count
from patient
inner join problem_list
    on patient.pat_id = problem_list.pat_id
inner join master_diagnosis
    on problem_list.dx_id = master_diagnosis.dx_id
where
    problem_list.prob_resolved_year is null
    and problem_list.chronic_yn = 'Y'
```

Children's Hospital of Philadelphia®

A Realistic Join Query on ARCUS Data

The example ADR query below returns results for all labs ordered within the CHOP system:

```
--example #2 : Lab Query Template
select distinct
        proc_order.proc_ord_id
        ,proc_order.pat_id
        ,proc_order.encounter_id
        ,proc_result.result_year
        ,proc_order.proc_ord_desc
        ,proc_result.line
        ,proc_result.result_component_name
        ,proc_result.value_number
        ,proc_result.ref_low
        ,proc_result.ref_high
        ,proc_result.ref_range
        ,proc_result.ref_unit
        ,proc_result.abnormal_status_name
        ,proc_result.loinc_code
from adr_coded.procedure_order as proc_order
inner join adr_coded.procedure_order_result as proc_result
        on proc_order.proc_ord_id = proc_result.proc_ord_id
```

Children's Hospital of Philadelphia®

Any Questions?

Arcus Education Homepage:

https://education.arcus.chop.edu/

Recordings Archive (recording for this Lecture to be added soon!)

https://education.arcus.chop.edu/recorded-webinars/

Stay Tuned for Updates on the SQL 201 & 202 Lecture Series!

Children's Hospital of Philadelphia®