DEALING WITH DATA

SPRING 2019: INFO-GB.2346.30

PROFESSOR GUTHRIE COLLIN

TEACHING FELLOW AJINKYA WALIMBE

CLASS 6: SQL

MARCH 28, 2019

CONTENTS

Group By: Aggregation Queries

Group by

count(*), sum(*), avg(*), min, max:
Applied to groups!!!!

SELECT A_1, Aggregation Function
FROM $T_1, T_2, \ldots T_m$
WHERE condition
Group By A_1

Note: Whatever attribute you select (in this case A_1) **must** appear in the group by clause.

Basic aggregation functions

Operator	*Description*
count(*)	Counts the number of rows in the group
count(attr)	Counts the number of rows in the group with non-null values for the attribute
count(DISTINCT attr)	Counts the number of distinct, non-null values for the attribute in the group
max(attr)	Row with maximum attribute value in the group
min(attr)	Row with minimum value in the group
sum(attr)	Sums values of selected rows in the group
avg(attr)	Estimates the average the attribute in the group

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT    Student_id, count(*)
FROM      Table1
GROUP BY  Student_id
```

Student_id	*Count*
1	4
2	3
3	1

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT    Student_id, avg(Grade)
FROM      Table1
GROUP BY  Student_id
```

Student_id	*Avg(grade)*
1	(19+17+19+14)/4
2	(16+18+13)/3
3	20

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT    Class, avg(Grade)
FROM      Table1
GROUP BY  Class
```

Class	*Avg(grade)*
Algrebra	(19+16+20)/3
Analysis	(18+17)/2
History	14
Physics	(19+13)/2

Aggregation practice query #1

Find the most popular Category for NYC filming permits

Aggregation practice query #1 - ANSWER

Find the most popular Category for NYC filming permits

```sql
SELECT
    Category,
    count(distinct EventID) as film_permits
FROM nyc_film_permits
GROUP BY
    Category
ORDER BY
    film_permits DESC;
```

Aggregation practice query #2

Find the most popular Borough by Category for NYC filming permits

Aggregation practice query #2 - ANSWER

Find the most popular Borough by Category for NYC filming permits

```sql
SELECT
    Category,
    Borough,
    count(distinct EventID) as film_permits
FROM nyc_film_permits
GROUP BY
    Category,
    Borough
ORDER BY
    Category ASC,
    film_permits DESC;
```

Aggregation practice query #3

Find the year and zip code with the most tax returns in NYC

Aggregation practice query #3 - ANSWER

Find the year and zip code with the most tax returns in NYC

```sql
SELECT
    zipcode,
    year,
    SUM(return_count) as total_returns
FROM irs_nyc_tax_returns
GROUP BY
    zipcode,
    year
ORDER BY
    total_returns DESC;
```

Having

SELECT A_1, Aggregation Function
FROM $T_1, T_2, \ldots T_m$
WHERE condition
GROUP BY A_1

HAVING Aggregation Function **Condition**

Differences of WHERE and HAVING

- WHERE applies to rows, **before** computing the aggregate
- HAVING applies to aggregate value only

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT    Class, avg(Grade)
FROM      Table1
GROUP BY  Class
HAVING    avg(Grade) > 15
```

Class	*Avg(grade)*
Algrebra	(19+16+20)/3
Analysis	(18+17)/2
Physics	(19+13)/2

History class is *not* included in the result because its average is 14 (less than 15)

Group by Toy Example

Table1

Student_id	*Class*	*Grade*
1	Algebra	19
2	Algebra	16
3	Algebra	20
2	Analysis	18
2	Physics	13
1	Analysis	17
1	Physics	19
1	History	14

```
SELECT    Student_id, count(*)
FROM      Table1
GROUP BY  Student_id
HAVING        count(*) > 2
```

Student_id	*count*
1	4
2	3

Student with id=3 is not included in the results because he/she is taking only one class

HAVING practice query #1

Find the Subcategories with more than 5000 NYC filming permits in the database

HAVING practice query #1 - ANSWER

Find the Subcategories with more than 5000 NYC filming permits in the database

```sql
SELECT
    Category,
    SubCategoryName,
    COUNT(DISTINCT EventID) as permit_count
FROM nyc_film_permits
GROUP BY
    Category,
    SubCategoryName
HAVING
    permit_count > 5000
ORDER BY
    permit_count DESC;
```

HAVING practice query #2

Find the Year where “Commercials” had less than 700 filming permits

HAVING practice query #2 - ANSWER

Find the Year where “Commercials” had less than 700 filming permits

```sql
SELECT
    STRFTIME('%Y',StartDateTime) as permit_year,
    COUNT(DISTINCT EventID) as permit_count
FROM nyc_film_permits
WHERE LOWER(Category) = "commercial"
GROUP BY
    permit_year
HAVING
    permit_count < 700
ORDER BY
    permit_count DESC;
```

Conditional Construct: CASE

Conditional Construct: CASE

Syntax

```
CASE
        WHEN condition THEN result
        [WHEN condition THEN result] ...
        [ELSE result]
END
```

Example: Send targeted ads to Single Females. Do not show ads to Males. Others TBD

```
SELECT     P.ProfileID, P.Name, P.Sex, R.Status,
        CASE
        WHEN P.Sex = 'Female' AND R.Status = 'Single' THEN `TargetAd'
        WHEN P.Sex = 'Male' THEN 'NoAd'
        ELSE 'TBD'
        END AS AdTargeting
FROM      Profiles P JOIN Relationship R ON R.ProfileID = P.ProfileID
```

CASE Practice Query #1

Map each zip code and year's return counts to a "return count" bucket

```sql
SELECT
    year,
    zipcode,
    CASE
        WHEN SUM(return_count) < 5000 THEN "under 5k returns"
        WHEN SUM(return_count) < 10000 THEN "5k to 10k returns"
        WHEN SUM(return_count) < 25000 THEN "10k to 25k returns"
        WHEN SUM(return_count) < 50000 THEN "25k to 50k returns"
        WHEN SUM(return_count) < 100000 THEN "50k to 100k returns"
        ELSE "over 100k returns"
    END AS return_count_bucket
FROM irs_nyc_tax_returns
GROUP BY
    year,
    zipcode;
```

CASE, alternatives

Syntax

```
CASE value
      WHEN compare_value THEN result
      [WHEN compare_value THEN result] ...
      [ELSE result]
END
```

Example: Is the student at Stern or not?

```
SELECT    P.ProfileID, P.Name, P.Sex, C.Concentration,
      CASE C.Concentration
      WHEN 'Finance' THEN 'Sternie'
      WHEN 'Accounting' THEN 'Sternie'
      WHEN 'Marketing' THEN 'Sternie'
      ELSE 'Not a Sternie'
      END AS SternieOrNot
FROM      Profiles P JOIN Concentration C ON C.ProfileID = P.ProfileID
```

CASE Practice Query #2

Make the results more human friendly by giving a short name for the agi_map_id field results.

```sql
SELECT
    year,
    zipcode,
    CASE agi_map_id
        WHEN 1 THEN "under $25K"
        WHEN 2 THEN "$25K to $50K"
        WHEN 3 THEN "$50K to $75K"
        WHEN 4 THEN "$75K to $100K"
        WHEN 5 THEN "$100K to $200K"
        ELSE "over $200K"
    END AS income_level,
    return_count
FROM irs_nyc_tax_returns
WHERE
    zipcode = 10128
    AND year = 2013;
```

CASE Practice Query #3

Flatten the IRS database by putting the return_count in a separate column for each agi_map_id

```sql
SELECT
    year,
    zipcode,
    SUM(CASE WHEN agi_map_id = 1 THEN return_count ELSE NULL END) AS agi_under25k,
    SUM(CASE WHEN agi_map_id = 2 THEN return_count ELSE NULL END) AS agi_25k_to_50k,
    SUM(CASE WHEN agi_map_id = 3 THEN return_count ELSE NULL END) AS agi_50k_to_75k,
    SUM(CASE WHEN agi_map_id = 4 THEN return_count ELSE NULL END) AS agi_75k_to_100k,
    SUM(CASE WHEN agi_map_id = 5 THEN return_count ELSE NULL END) AS agi_100K_to_200k,
    SUM(CASE WHEN agi_map_id = 6 THEN return_count ELSE NULL END) AS agi_over200k
FROM irs_nyc_tax_returns
GROUP BY
    year,
    zipcode;
```

BASICS OF SQL JOINS

- Enables an analyst to combine rows from two or more data tables

- Requires a common data field found in both tables to enable the combination
  - Data field comparison does NOT require the same column name
  - Data fields MUST be the same data type to enable comparison

- Syntax:

```
SELECT ...

FROM LeftTable

{INNER|RIGHT|LEFT|FULL OUTER} JOIN RightTable
ON LeftTable.common_field = RightTable.common_field
```

EXAMPLE: APPLES & ORCHARDS

VARIETIES

Apple_ID	Name
1	Golden Delicious
2	Red Delicious
3	Granny Smith
4	Fuji
5	McIntosh

PRODUCTION

Orchard_ID	Apple_ID	Quantity
100	1	100
101	3	50
101	5	100
102	3	50
103	4	100

EXAMPLE: APPLES & ORCHARDS

VARIETIES

Apple_ID	Name
1	Golden Delicious
2	Red Delicious
3	Granny Smith
4	Fuji
5	McIntosh

PRODUCTION

Orchard_ID	Apple_ID	Quantity
100	1	100
101	3	50
101	5	100
102	3	50
103	4	100

WHAT WAS APPLE PRODUCTION BY ORCHARD AND VARIETY NAME?

WHAT WAS APPLE PRODUCTION BY ORCHARD AND VARIETY NAME?

VARIETIES

Apple_ID	Name
1	Golden Delicious
2	Red Delicious
3	Granny Smith
4	Fuji
5	McIntosh

```
SELECT
     production.orchard_id
     ,varieties.name
     ,production.quantity
FROM      varieties
     JOIN production
     ON varieties.apple_id =
     production.apple_id;
```

PRODUCTION

Orchard_ID	Apple_ID	Quantity
100	1	100
101	3	50
101	5	100
102	3	50
103	4	100

Orchard_ID	Name	Quantity
100	Golden Delicious	100
101	Granny Smith	50
101	McIntosh	100
102	Granny Smith	50
103	Fuji	100

(SAME EXAMPLE WITH DIFFERENT COLUMN NAME)

VARIETIES

Variety_ID	Name
1	Golden Delicious
2	Red Delicious
3	Granny Smith
4	Fuji
5	McIntosh

```
SELECT
     production.orchard_id
     ,varieties.name
     ,production.quantity
FROM      varieties
     JOIN production
     ON varieties.variety_id =
     production.apple_id;
```

PRODUCTION

Orchard_ID	Apple_ID	Quantity
100	1	100
101	3	50
101	5	100
102	3	50
103	4	100

Orchard_ID	Name	Quantity
100	Golden Delicious	100
101	Granny Smith	50
101	McIntosh	100
102	Granny Smith	50
103	Fuji	100

SQL JOIN TYPES

- **INNER JOIN** (aka "JOIN"): Returns all rows when comparison fields (in "ON" statement) are matching in BOTH tables.

- **LEFT JOIN**: Returns all rows from left table with matching rows (based on comparison fields) from right table.

- **RIGHT JOIN**: Return all rows from right table with matching rows from left table.

- **FULL JOIN**: Return all rows from both tables when there is a match in at least ONE of the tables.

- **CROSS JOIN**: Returns all possible combinations of rows from each table.

CROSS JOIN EXAMPLE

VARIETIES

Apple_ID	Name	Color_ID
1	Golden Delicious	200
2	Red Delicious	201
5	McIntosh	201

```
SELECT *
FROM    varieties,
colors;
```

COLORS

Color_ID	Color
200	Yellow
201	Red

Apple_ID	Name	Varieties.Color_ID	Colors.Color_ID	Color
1	Golden Delicious	200	200	Yellow
1	Golden Delicious	200	201	Red
2	Red Delicious	201	200	Yellow
2	Red Delicious	201	201	Red
5	McIntosh	201	200	Yellow
5	McIntosh	201	201	Red

CROSS JOIN

VARIETIES

Apple_ID	Name	Color_ID
1	Golden Delicious	200
2	Red Delicious	201
5	McIntosh	201

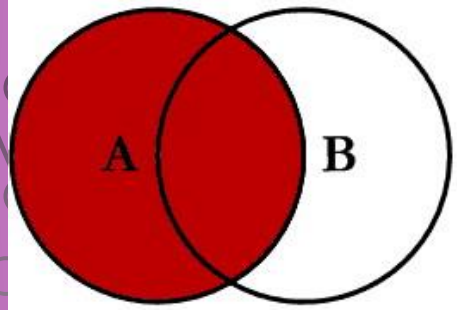
```
SELECT * *
FROM    varieties,
colors;
```

COLORS

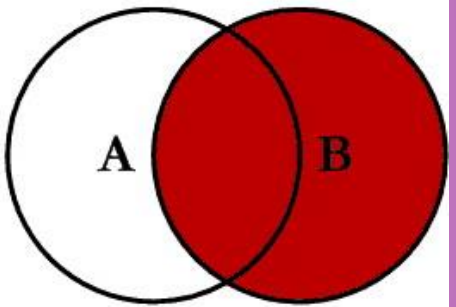
Color_ID	Color
200	Yellow
201	Red

Apple_ID	Name	Varieties.Color_ID	Colors.Color_ID	Color
1	Golden Delicious	200	200	Yellow
1	Golden Delicious	200	201	Red
2	Red Delicious	201	200	Yellow
2	Red Delicious	201	201	Red
5	McIntosh	201	200	Yellow
5	McIntosh	201	201	Red

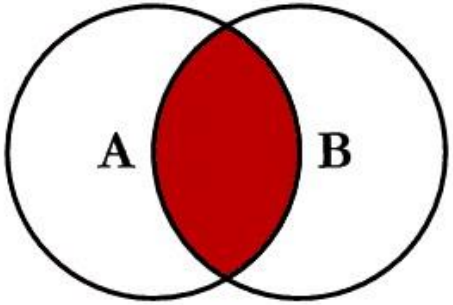
SQL JOINS



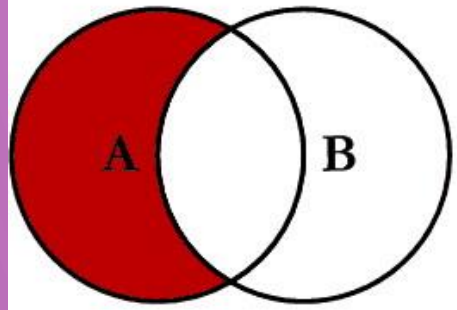

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



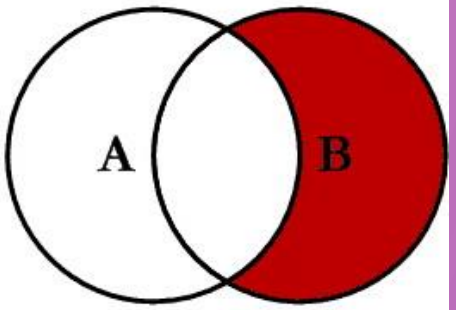

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



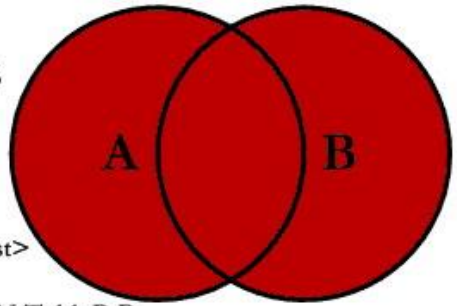

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



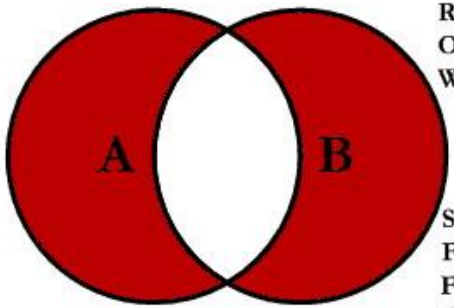

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

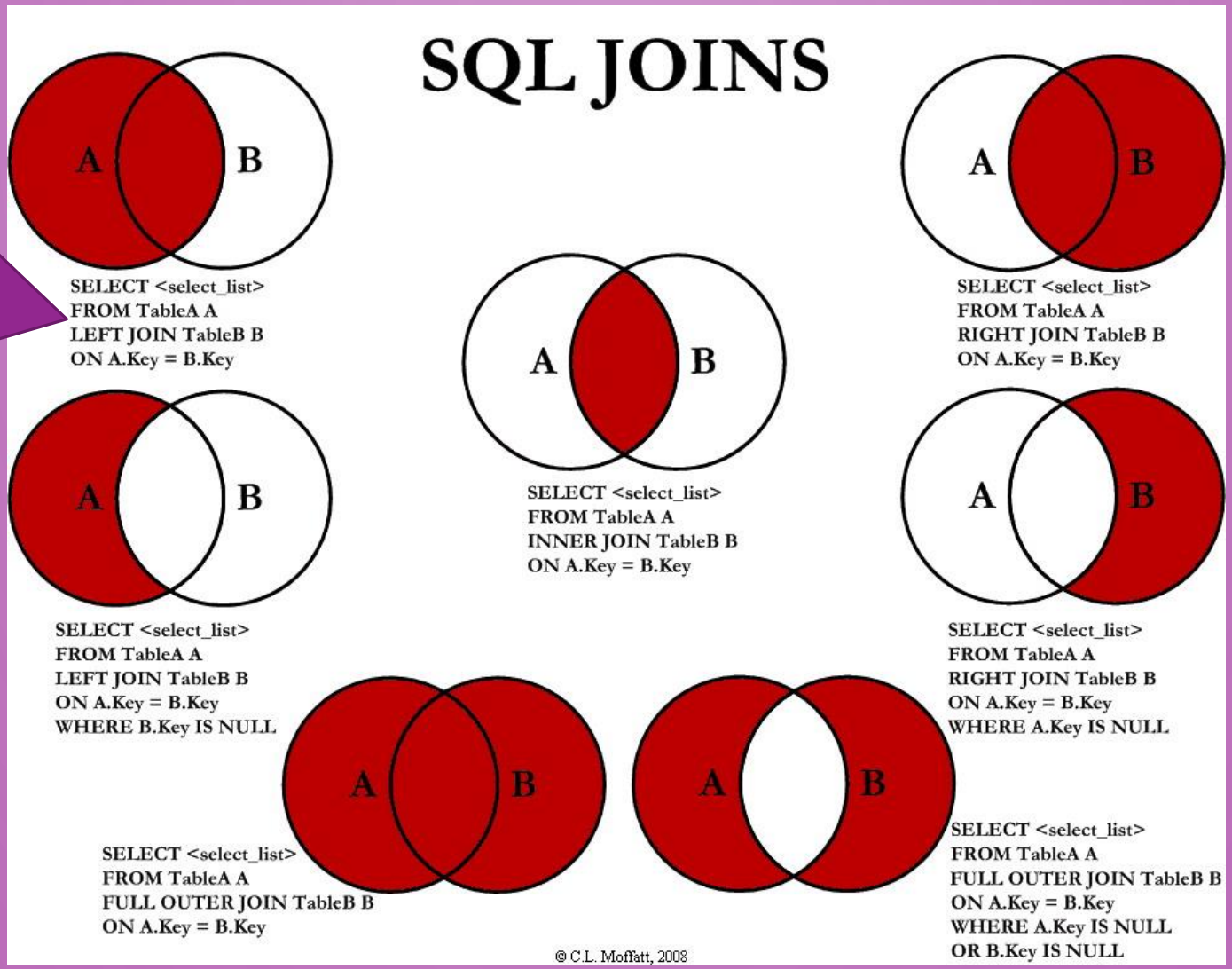


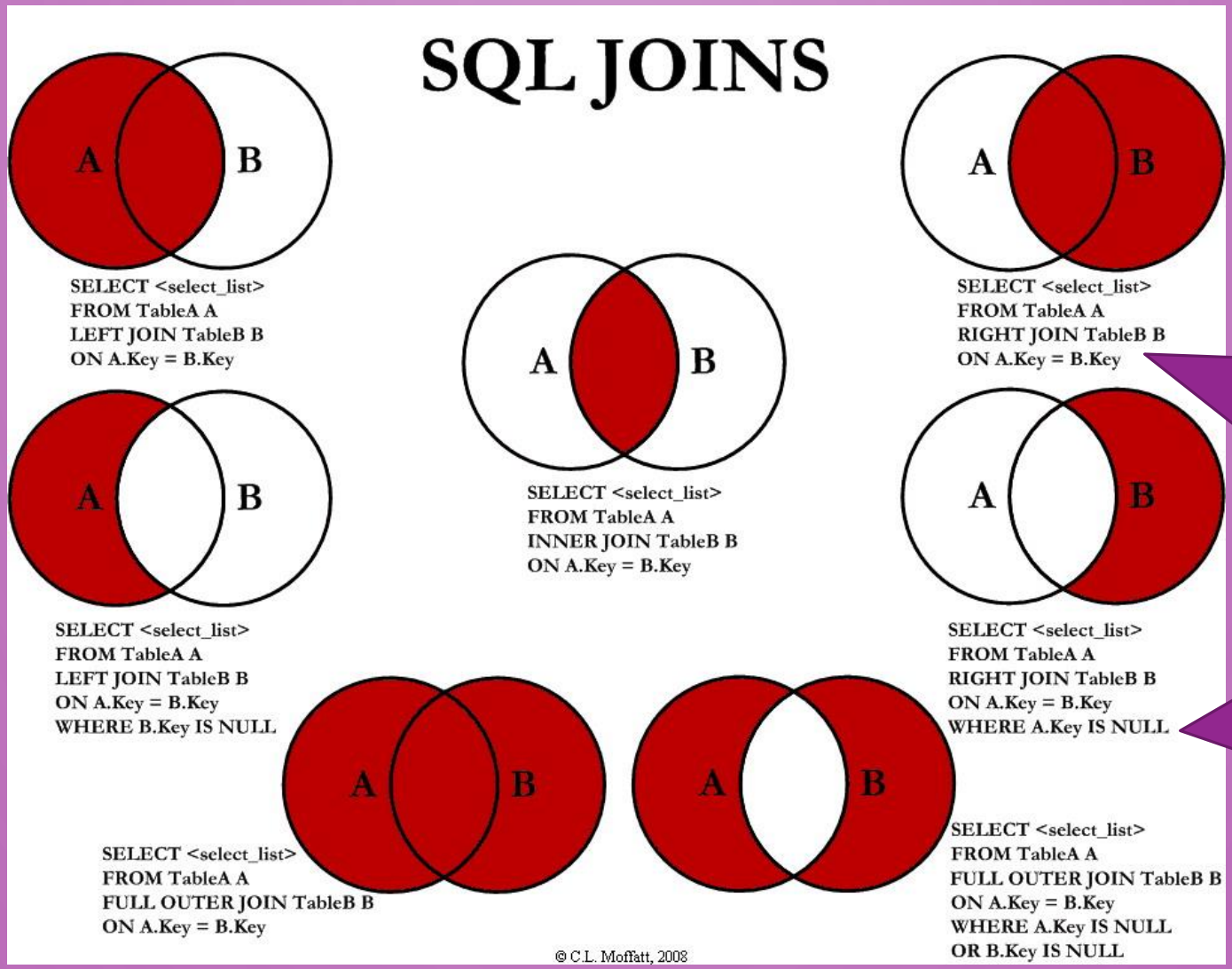

```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

"LEFT" table and "RIGHT" table are determined by order after the FROM command





Uses "A.Key=B.Key " to return results from right table plus matching rows on left table.

Uses "WHERE A.Key IS NULL" to return only rows from RIGHT table.

JOIN PRACTICE QUERY #1 - INNER

Combine IRS tax returns with the human-readable form of the income buckets

JOIN PRACTICE QUERY #1 - INNER

Combine IRS tax returns with the human-readable form of the income buckets

```sql
SELECT
    map.agi_descrp,
    SUM(return_count) AS total_returns
FROM irs_nyc_tax_returns irs
    INNER JOIN irs_agi_map map
        ON irs.agi_map_id = map.agi_map_id
GROUP BY
    map.agi_descrp
ORDER BY
    irs.year ASC;
```

JOIN PRACTICE QUERY #2 - INNER

Find out if there's a potential relationship between total filming permits and annual average tax returns for the top 10 zip codes for filming permits.

JOIN PRACTICE QUERY #2 - INNER

Find out if there's a potential relationship between total filming permits and annual average tax returns for the top 10 zip codes for filming permits.

```sql
SELECT
    irs.zipcode,
    ROUND(AVG(irs.return_count),0) as tax_returns,
    COUNT(DISTINCT nfp.EventId) as permit_count
FROM nyc_film_permits nfp
    INNER JOIN irs_nyc_tax_returns irs
        ON (irs.zipcode = nfp.zipcode AND strftime('%Y',nfp.EndDateTime) = irs.year)
GROUP BY
    irs.zipcode
ORDER BY
    permit_count DESC
LIMIT 10;
```

JOIN PRACTICE QUERY #3 - LEFT

Find the bottom 20 zip codes for filming permits.

JOIN PRACTICE QUERY #3 - LEFT

Find the bottom 20 zip codes for filming permits.

```sql
SELECT
    irs.zipcode,
    COUNT(DISTINCT nfp.EventId) as permit_count
FROM irs_nyc_tax_returns irs
    LEFT OUTER JOIN nyc_film_permits nfp
        ON (irs.zipcode = nfp.zipcode AND strftime('%Y',nfp.EndDateTime) = irs.year)
GROUP BY
    irs.zipcode
ORDER BY
    permit_count ASC
LIMIT 20;
```

JOIN PRACTICE QUERY #4 - LEFT

Find the percentage of children living in the top 5 filming permit zip codes over all years in the database

JOIN PRACTICE QUERY #4 - LEFT

Find the percentage of children living in the top 5 filming permit zip codes over all years in the database

```sql
SELECT
    cen.zipcode,
    100 * (SUM(Age_under5) + SUM(Age_5to9) + SUM(Age_10to14) + SUM(Age_15to19)) / SUM(cen.age_all)
    COUNT(DISTINCT nfp.EventId) as permit_count
FROM nyc_census_data cen
    LEFT JOIN nyc_film_permits nfp
        ON cen.zipcode = nfp.zipcode
GROUP BY
    cen.zipcode
ORDER BY
    permit_count DESC
LIMIT 5;
```

SUBQUERIES

- Subqueries are temporary tables created with nested SELECT statements where a table should be, typically used in JOINS
- Subqueries can enable deeper analysis with SQL
- Subqueries (and nesting them) is fast, easy to code-up once, but can be slow, hard to maintain, and can come with a cost (in query performance)
- Two ways to improve these: (1) Add Aliases; (2) Use WITH clause

NESTED *subqueries* AND *subqueries* **AS** *aliases*

```
SELECT
     ...
FROM Table1
LEFT JOIN (
          SELECT ...
               FROM (
                    SELECT ...
                    FROM Table3
                    RIGHT JOIN ( ... )
                    )
          )
WHERE Column IN (SELECT … FROM …)
```

```
SELECT
     ...
FROM Table1
LEFT JOIN (
     SELECT ...
     FROM Table2
) AS t2 ON Table1.x = t2.x
INNER JOIN (
     SELECT ...
     FROM Table3
) AS t3 ON Table1.y = t3.y
```

ALIAS

WITH CLAUSE

- The WITH clause creates a Common Table Expression and enables sub-query empowered analyses with easier SQL codebase

- Syntax:

```
WITH
     temp_table1 AS (
          SELECT ... FROM ...
     ),
     temp_table2 AS (
          SELECT ... FROM ...
     )

SELECT ...
FROM temp_table1
     INNER JOIN temp_table2
     ON temp_table1.x = temp_table2.x
```

4. DID EACH COLOR GENERATE TRAFFIC ON THE FIRST DAY?

SOLUTION: SQL, REVISITED USING **WITH**

WITH *alias* **AS** *(query)*

```
WITH dt AS (
     SELECT *
     FROM daily_traffic
     WHERE daily_traffic.date =
TO_DATE('01/01/2018', 'MM/DD/YYYY')
)

SELECT
     colors.product_id
     ,colors.color
     ,dt.daily_page_view_count
FROM colors
LEFT JOIN dt
     ON colors.product_id =
dt.product_id;
```

Nested *(subquery)* **AS** *alias*

```
SELECT
     colors.color
     ,dt.daily_page_view_count
FROM colors
LEFT JOIN (
     SELECT *
     FROM daily_traffic
     WHERE daily_traffic.date =
TO_DATE('01/01/2018', 'MM/DD/YYYY')
     ) AS dt
     ON colors.product_id =
dt.product_id;
```

SUBQUERY PRACTICE #1 & 2 – NESTED VS WITH CTE

Find the top 10 zipcodes with the largest percentage of persons of Latino/Hispanic descent, and calculate the percentage of adults and percentage of six-figure income earners in these zip codes

SUBQUERY PRACTICE #1 – NESTED

```sql
SELECT
    cen.zipcode,
    100 * (SUM(Age_all) - (SUM(cen.Age_under5) + SUM(cen.Age_5to9) + SUM(cen.Age_10to14) + SUM(cen.Age_15to19) )) / SUM
    (Age_all) AS adult_pct,
    100 * SUM(cen.Ethnicity_All_HispancLatino_Descent) / SUM(cen.Age_all) AS latino_pct,
    100 * (SUM(irs.agi_over200k) + SUM(irs.agi_100K_to_200K)) / SUM(total_returns) AS six_figure_income_pct
FROM nyc_census_data cen
    INNER JOIN (
        SELECT
            year,
            zipcode,
            SUM(CASE WHEN agi_map_id = 1 THEN return_count ELSE NULL END) AS agi_under25k,
            SUM(CASE WHEN agi_map_id = 2 THEN return_count ELSE NULL END) AS agi_25k_to_50k,
            SUM(CASE WHEN agi_map_id = 3 THEN return_count ELSE NULL END) AS agi_50k_to_75k,
            SUM(CASE WHEN agi_map_id = 4 THEN return_count ELSE NULL END) AS agi_75k_to_100k,
            SUM(CASE WHEN agi_map_id = 5 THEN return_count ELSE NULL END) AS agi_100K_to_200k,
            SUM(CASE WHEN agi_map_id = 6 THEN return_count ELSE NULL END) AS agi_over200k,
            SUM(return_count) AS total_returns
        FROM irs_nyc_tax_returns
        GROUP BY
            year,
            zipcode
            ) irs
        ON cen.zipcode = irs.zipcode
WHERE
    irs.year = 2012
GROUP BY
    cen.zipcode
ORDER BY
    latino_pct DESC
LIMIT 10;
```

SUBQUERY PRACTICE #2 – WITH *CTE*

```sql
WITH flat_irs AS (
    SELECT
        year,
        zipcode,
        SUM(CASE WHEN agi_map_id = 1 THEN return_count ELSE NULL END) AS agi_under25k,
        SUM(CASE WHEN agi_map_id = 2 THEN return_count ELSE NULL END) AS agi_25k_to_50k,
        SUM(CASE WHEN agi_map_id = 3 THEN return_count ELSE NULL END) AS agi_50k_to_75k,
        SUM(CASE WHEN agi_map_id = 4 THEN return_count ELSE NULL END) AS agi_75k_to_100k,
        SUM(CASE WHEN agi_map_id = 5 THEN return_count ELSE NULL END) AS agi_100K_to_200k,
        SUM(CASE WHEN agi_map_id = 6 THEN return_count ELSE NULL END) AS agi_over200k,
        SUM(return_count) AS total_returns
    FROM irs_nyc_tax_returns
    GROUP BY
        year,
        zipcode
)

SELECT
    cen.zipcode,
    100 * (SUM(Age_all) - (SUM(cen.Age_under5) + SUM(cen.Age_5to9) + SUM(cen.Age_10to14) + SUM(cen.Age_15to19) )) / SUM
    (Age_all) AS adult_pct,
    100 * SUM(cen.Ethnicity_All_HispancLatino_Descent) / SUM(cen.Age_all) AS latino_pct,
    100 * (SUM(irs.agi_over200k) + SUM(irs.agi_100K_to_200K)) / SUM(total_returns) AS six_figure_income_pct
FROM nyc_census_data cen
    INNER JOIN flat_irs irs
        ON cen.zipcode = irs.zipcode
WHERE
    irs.year = 2012
GROUP BY
    cen.zipcode
ORDER BY
    latino_pct DESC
LIMIT 10;
```

SUBQUERY PRACTICE #3 – WITH *CTE*

CLASS PROJECT ➔ we want to compare the demographic makeup of NYC versus the demographic makeup of the "NYC on video media (films/TV)". To do this, we first need to understand the percentage of NYC citizens in each major demographic bucket to compare versus the films.

SUBQUERY PRACTICE #3 – WITH *CTE, PART 1 ➔ CTE FOR CENSUS DATA*

```sql
WITH
/* Get US Census demographic totals for major categories for each zip code */
nyc_cen AS (
    SELECT
        cen.zipcode,
    /* Gender */
        SUM(cen.Gender_Male) AS gen_male_cnt,
        SUM(cen.Gender_Female) AS gen_female_cnt,
    /* ages */
        SUM(cen.Age_under5) + SUM(cen.Age_5to9) + SUM(cen.Age_10to14) + SUM(cen.Age_15to19) AS age_under19_cnt,
        SUM(cen.Age_over84) + SUM(cen.Age_80to84) + SUM(cen.Age_75to79) + SUM(cen.Age_70to74) AS age_over70_cnt,
        SUM(cen.Age_20to24) + SUM(cen.Age_25to29) + SUM(cen.Age_30to34) AS age_20to34_cnt,
        SUM(cen.Age_35to39) + SUM(cen.Age_40to44) + SUM(cen.Age_45to49) + SUM(cen.Age_50to54) AS age_25to54_cnt,
        SUM(Age_all)
            - ( --child count
              SUM(cen.Age_under5) + SUM(cen.Age_5to9) + SUM(cen.Age_10to14) + SUM(cen.Age_15to19)
              )
            - ( --senior citizen count
              SUM(cen.Age_over84) + SUM(cen.Age_80to84) + SUM(cen.Age_75to79) + SUM(cen.Age_70to74)
              )
            - ( --young adult count
              SUM(cen.Age_20to24) + SUM(cen.Age_25to29) + SUM(cen.Age_30to34)
              )
            - ( --prime earning years count
              SUM(cen.Age_35to39) + SUM(cen.Age_40to44) + SUM(cen.Age_45to49) + SUM(cen.Age_50to54)
              )
         AS age_55to70_cnt,
    /* ethnicity */
         SUM(cen.Ethnicity_White) AS eth_euro_cnt,
         SUM(cen.Ethnicity_AfricanAmerican) AS eth_african_cnt,
         SUM(cen.Ethnicity_Asian) + SUM(cen.Ethnicity_PacificIslander) AS eth_asiapac_cnt,
         SUM(cen.Ethnicity_NativeAmerican) + SUM(cen.Ethnicity_Other) AS eth_other_cnt,
         SUM(cen.Ethnicity_All_HispancLatino_Descent) AS eth_hislat_descent_cnt,
    /* total pop */
        SUM(cen.Age_all) as total_pop_cnt
    FROM
        nyc_census_data cen
    GROUP BY
        cen.zipcode
    ORDER BY
        cen.zipcode ASC
),
```

SUBQUERY PRACTICE #3 – WITH *CTE, PART 2 ➔ CTE FOR IRS DATA*

```
1   WITH
2   /* Get US Census demographic totals for major categories for each zip code */
3   nyc_cen AS (…
42  ),
43
44  /* Get IRS income category totals for each zip code */
45  nyc_irs AS (
46      SELECT
47          zipcode,
48          SUM(CASE WHEN agi_map_id IN (1,2) THEN return_count ELSE NULL END) AS agi_under50k_return_cnt,
49          SUM(CASE WHEN agi_map_id IN (3,4) THEN return_count ELSE NULL END) AS agi_50k_to_100k_return_cnt,
50          SUM(CASE WHEN agi_map_id IN (5,6) THEN return_count ELSE NULL END) AS agi_over100K_return_cnt,
51          SUM(return_count) AS total_return_cnt
52      FROM irs_nyc_tax_returns
53      WHERE year >= 2012
54          AND year <= 2015
55      GROUP BY
56          zipcode
57      ORDER BY
58          zipcode ASC
59  ),
```

SUBQUERY PRACTICE #3 – WITH *CTE, PART 3 ➔ CTE FOR COMBINED DATA*

```
1   WITH
2   /* Get US Census demographic totals for major categories for each zip code */
3   nyc_cen AS (…
42  ),
43
44  /* Get IRS income category totals for each zip code */
45  nyc_irs AS (…
59  ),
60
61  /* Get Totals for NYC zips */
62  nyc_total_cnt_by_zip AS (
63      SELECT
64          cen.*,
65          irs.agi_under50K_return_cnt,
66          irs.agi_50k_to_100k_return_cnt,
67          irs.agi_over100K_return_cnt,
68          irs.total_return_cnt
69      FROM
70          nyc_cen cen
71          INNER JOIN
72              nyc_irs irs
73                  ON (cen.zipcode = irs.zipcode)
74      ORDER BY
75          cen.zipcode
76  )
```

SUBQUERY PRACTICE #3 – WITH *CTE, PART 4 ➔ FINAL QUERY*

```sql
WITH
/* Get US Census demographic totals for major categories for each zip code */
nyc_cen AS (…
),

/* Get IRS income category totals for each zip code */
nyc_irs AS (…
),

/* Get Totals for NYC zips */
nyc_total_cnt_by_zip AS (…
)

/* NYC demographic makeup using 2010 US Census and tax returns from 2012-2015 */
SELECT
-- label
        "all nyc" AS description,
--gender
        CAST(SUM(gen_male_cnt) AS Float) / SUM(total_pop_cnt) AS gen_male_pct,
        CAST(SUM(gen_female_cnt) AS Float) / SUM(total_pop_cnt) AS gen_female_pct,
--age
        CAST(SUM(age_under19_cnt) AS Float)/ SUM(total_pop_cnt) AS age_under19_pct,
        CAST(SUM(age_20to34_cnt) AS Float) / SUM(total_pop_cnt) AS age_20to34_pct,
        CAST(SUM(age_25to54_cnt) AS Float) / SUM(total_pop_cnt) AS age_25to54_pct,
        CAST(SUM(age_55to70_cnt) AS Float) / SUM(total_pop_cnt) AS age_55to70_pct,
        CAST(SUM(age_over70_cnt) AS Float) / SUM(total_pop_cnt) AS age_over70_pct,
--income
        CAST(SUM(agi_under50K_return_cnt) AS Float)/ SUM(total_return_cnt) AS agi_under50K_return_pct,
        CAST(SUM(agi_50k_to_100k_return_cnt) AS Float)/ SUM(total_return_cnt) AS agi_50k_to_100k_return_pct,
        CAST(SUM(agi_over100K_return_cnt) AS Float)/ SUM(total_return_cnt) AS agi_over100K_return_pct,
--ethnicity
        CAST(SUM(eth_euro_cnt) AS Float)/ SUM(total_pop_cnt) AS eth_euro_pct,
        CAST(SUM(eth_african_cnt) AS Float)/ SUM(total_pop_cnt) AS eth_african_pct,
        CAST(SUM(eth_asiapac_cnt) AS Float)/ SUM(total_pop_cnt) AS eth_asiapac_pct,
        CAST(SUM(eth_other_cnt) AS Float)/ SUM(total_pop_cnt) AS eth_other_pct,
        CAST(SUM(eth_hislat_descent_cnt) AS Float)/ SUM(total_pop_cnt) AS eth_hislat_descent_pct
FROM
    nyc_total_cnt_by_zip
```

SUBQUERY EXAMPLE #4 – WITH *CTE*

CLASS PROJECT ➔

- we want to compare the demographic makeup of NYC versus the demographic makeup of the "NYC on video media (films/TV)".
- In practice #3 – we found the makeup for all NYC.
- In example #4, we must do the same calculations but only for those zip codes that had filming shoots during 2012-2015.
- We also have to choose whether we weight up/down the zip codes by the number of filming shoots.

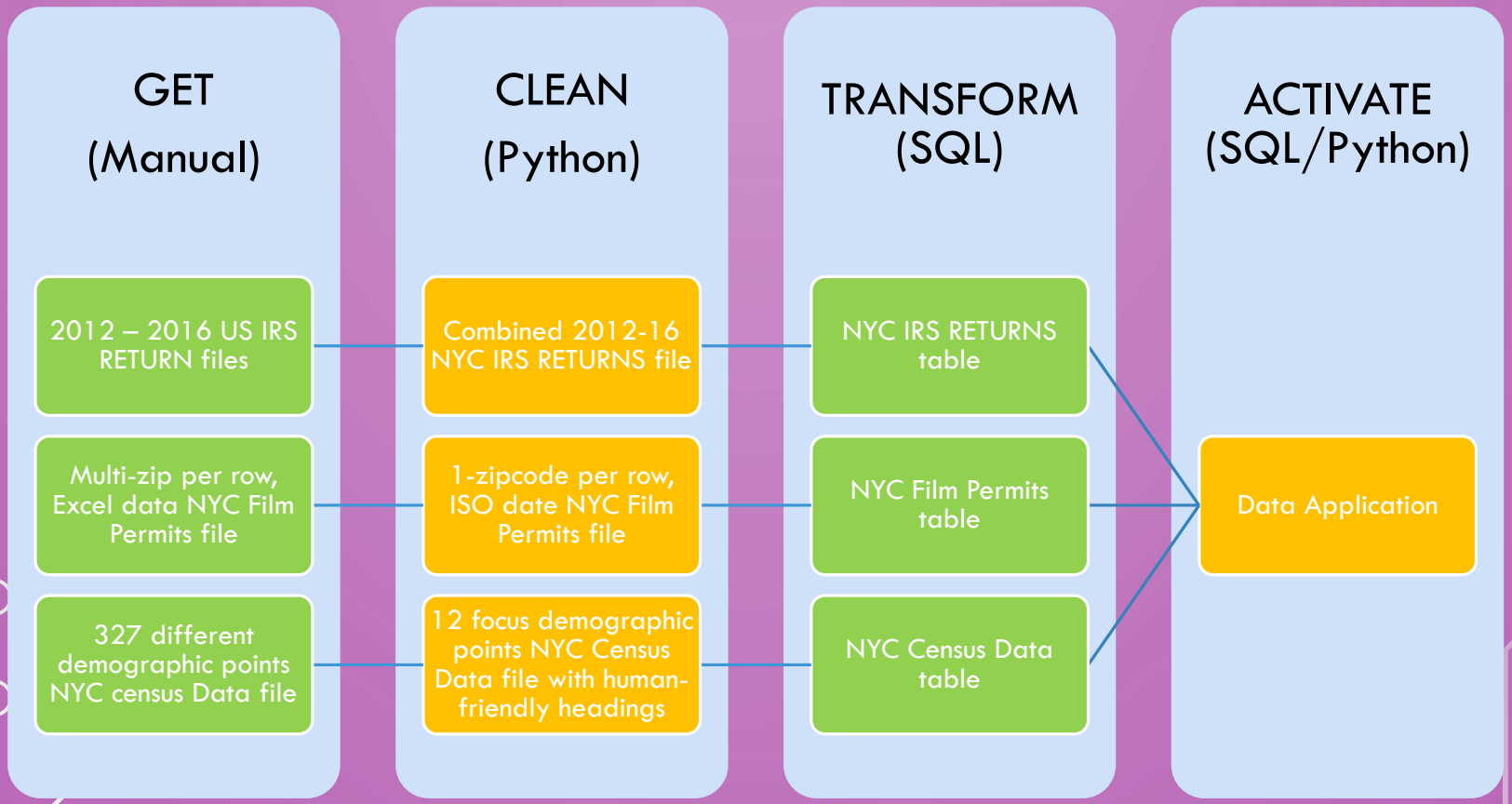
SUBQUERY EXAMPLE #4 – WITH *CTE*
UNWEIGHTED ZIP CODES

description	all nyc	nyc film permits, unweighted	films (-) all	Meaning
gen_male_pct	48%	48%	0%	
gen_female_pct	52%	52%	0%	
age_under19_pct	25%	24%	-1%	
age_20to34_pct	21%	25%	4%	Films portray a city with slightly more 20-something to early 30s
age_25to54_pct	28%	28%	0%	
age_55to70_pct	16%	15%	-1%	
age_over70_pct	10%	9%	-1%	
agi_under50K_return_pct	61%	66%	5%	Films portray a city slightly more under $50K earners
agi_50k_to_100k_return_pct	22%	20%	-2%	
agi_over100K_return_pct	17%	14%	-3%	
eth_euro_pct	68%	47%	-21%	Films portray a city with far less white citizens
eth_african_pct	17%	27%	10%	Films portray a city with more African-American citizens
eth_asiapac_pct	8%	14%	6%	Films portray a city with slightly more Asian-American citizens
eth_other_pct	10%	16%	6%	Films portray a city with slightly more diverse citizens beyond African-American and Asian-American
eth_hislat_descent_pct	18%	28%	10%	Films portray a city with more citizens of Hispanic or Latino descent

SUBQUERY EXAMPLE #4 – WITH *CTE WEIGHTED ZIP CODES*

description	all nyc	nyc films, weighted	films (-) all	Meaning
gen_male_pct	48%	48%	0%	
gen_female_pct	52%	52%	0%	
age_under19_pct	25%	19%	-6%	Films portray a city with slightly less children
age_20to34_pct	21%	31%	10%	Films portray a city with more 20-something to early 30s
age_25to54_pct	28%	28%	0%	
age_55to70_pct	16%	14%	-2%	
age_over70_pct	10%	8%	-2%	
agi_under50K_return_pct	61%	52%	-9%	Films portray a city with much less under $50K earners
agi_50k_to_100k_return_pct	22%	22%	0%	
agi_over100K_return_pct	17%	25%	8%	Films portray a city with more over $100K earners
eth_euro_pct	68%	63%	-5%	Films portray a city with slightly less white citizens
eth_african_pct	17%	15%	-2%	
eth_asiapac_pct	8%	14%	6%	Films portray a city with slightly more Asian-American citizens
eth_other_pct	10%	12%	2%	
eth_hislat_descent_pct	18%	22%	4%	Films portray a city with slightly more citizens of Hispanic or Latino descent

CLASS PROJECT PIPELINE: NYC MEDIA REPRESENTATION



Comparison Operators

Operator	*Description*
=	equals
<>	is not equal to
!=	is not equal to
<	less than
>	greater than
AND	logical and
OR	logical or
NOT	logical not

Other operators

SQL	*Description*
as	used to change the name of a column in the result
distinct	no duplicate rows
order by column(s)	sorts by column(s) in ascending order
order by .. desc	sorts by column(s) in descending order
*	select all columns
like ‘%pattern_’	$: any sequence of characters _: any single character
attribute is null	rows that have null values for the specific attribute
is not null	rows that have not null values for the specific attribute
between this and that	between **this** value and **that** value
in	set membership
limit n	fetches only the top n rows from the database