BROWN
Computer Science

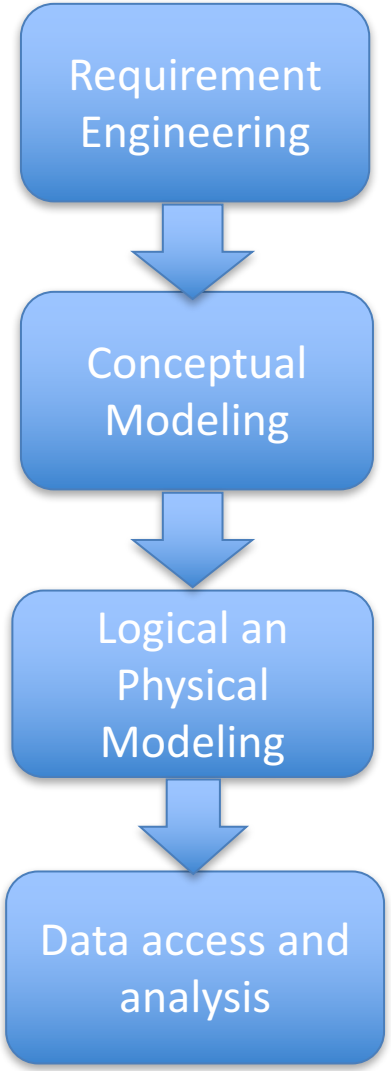
CS1951A: Data Science

Lecture 4: SQL Queries

Lorenzo De Stefani

Spring 2022

Databases for Data Scientists



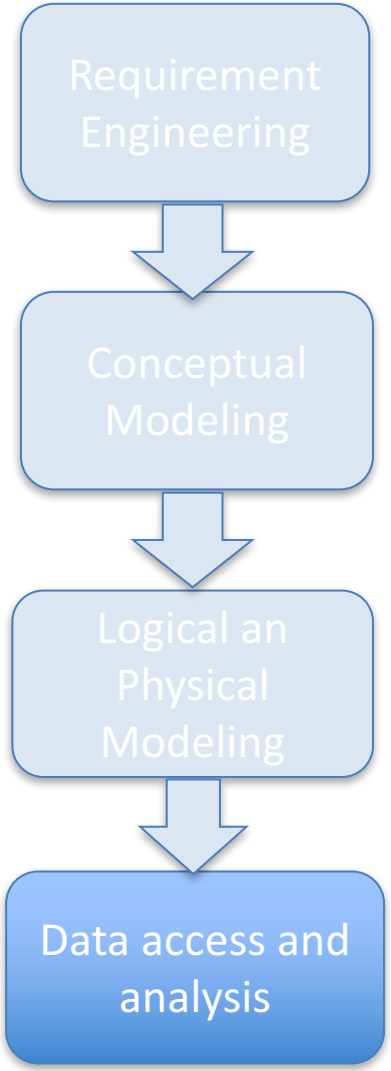
- "Book of duty"
- Understand and model the "world" of interest

- Conceptual DB design
- Entity Relations (ER) method

- Logical design (schema, table names, data types)
- Physical design (index, hints, memory organization)

- Asking and answering questions (queries)
- Extract information form the DBMS (views)
- SQL and relational algebra

Databases for Data Scientists



- "Book of duty"
- Understand and model the "world" of interest

- Conceptual DB design
- Entity Relations (ER) method

- Logical design (schema, table names, data types)
- Physical design (index, hints, memory organization)

- Asking and answering questions (queries)
- Extract information form the DBMS (views)
- SQL and relational algebra

Querying the DBMS

TWEET

ID	Timestamp	Author	Text	Mentions
389472	1/1/19 12:34	Bob	hey	NULL
123794	1/1/19 12:32	Maria	lol	{Bob}
596208	1/2/19 1:04	Yu	:-D	NULL

- Queries retrieve a set of records among the state of a relation
  - Q1: "Find all tweets authored by Bob"
    - {(389472,1/1/19 12:34, Bob, hey,NULL)}
  - Q2: "Find all tweets that mention Joe"
    - ∅

Invalid queries

TWEET

ID	Timestamp	Author	Text	Mentions
389472	1/1/19 12:34	Bob	hey	NULL
123794	1/1/19 12:32	Maria	lol	{Bob}
596208	1/2/19 1:04	Yu	:-D	NULL

- Q3: “Find all tweets with more than 10k likes”
  - ❌
- The query is invalid, as the criteria specified to select records is not part of the relational scheme!

SQL for extracting information

- Data Definition Language (DDL)
  - Define data types (domains) and Relation Schemas (intensions!)
- Data Manipulation and Query Language (DML):
  - Populating/updating data bases (extensions!)
  - Querying DBMSs

Basic Template

```
SELECT <attribute list>
FROM <table list>
[ WHERE <condition> ]
[ GROUP BY <attribute list> ]
[ HAVING <condition> ]
[ ORDER BY <attribute list> ];
```

The output of such a query is a table constructed according to the specifications

- SELECT columns from <attribute list>
- Tuples selected FROM tables in the <table list> ...
- Satisfying (HAVING) the <condition>
- Records can be GROUPED according to the values of selected attributes (<attribute list >) satisfying some <condition>
- ... and ORDERED according to the values of selected attributes

Queries formalism with SQL

TWEET

ID	Timestamp	Author	Text	Mentions
389472	1/1/19 12:34	Bob	hey	NULL
123794	1/1/19 12:32	Maria	lol	{Bob}
596208	1/2/19 1:04	Yu	:-D	NULL

- * allows to select all attributes from the selected table
- Q1: "Find all tweets authored by Bob"
  - SELECT * FROM TWEET WHERE Author IS Bob
- Q2: "Find all tweets that mention Joe"
  - SELECT * FROM TWEET WHERE Joe ∈ Mentions

Queries formalism with SQL

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>;
```

Queries formalism with SQL

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

```
SELECT <attribute list>
FROM <table list>
WHERE <condition>;
```

Queries from multiple tables

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

```
SELECT ID, Text
FROM Tweet,
     Author
WHERE ID = Tweet
```

AUTHOR

Person	Tweet
s	389472
d	123794
j	596208
d	782138
d	127890
j	173902
s	893110

Cross condition for values in different tables

Attributes can be selected from different tables if uniquely identifiable

```
SELECT TWEET.ID, AUTHOR.Text
FROM Tweet,
     Author
WHERE ID = Tweet
```

Queries from multiple tables

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

AUTHOR

Person	Tweet
s	389472
d	123794
j	596208
d	782138
d	127890
j	173902
s	893110

```
SELECT ID, Person
FROM Tweet,
     Author
WHERE ID = Tweet
```

ID	Time	Text	Person	Tweet
389472	2019-01-01	hey	s	389472
123794	2019-01-01	lol	d	123794
596208	2019-01-02	:-D	j	596208
782138	2019-01-04	1951A 4 lyfe	d	782138
127890	2019-01-04	hey	d	127890
173902	2019-01-05	i <3 1951A	j	173902
893110	2019-01-06	i <3 1951A	s	893110

Joining Tables

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

```
SELECT ID, Person
FROM Tweet,
     Author
WHERE ID = Tweet
```

JOIN CONDITION

AUTHOR

Person	Tweet
s	389472
d	123794
j	596208
d	782138
d	127890
j	173902
s	893110

ID	Time	Text	Person	Tweet
389472	2019-01-01	hey	s	389472
123794	2019-01-01	lol	d	123794
596208	2019-01-02	:-D	j	596208
782138	2019-01-04	1951A 4 lyfe	d	782138
127890	2019-01-04	hey	d	127890
173902	2019-01-05	i <3 1951A	j	173902
893110	2019-01-06	i <3 1951A	s	893110

The JOIN condition, determines the way information from multiple table is combined

Joining Tables with unspecified condition

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

AUTHOR

Person	Tweet
s	389472
d	123794
j	596208
d	782138
d	127890
j	173902
s	893110

```
SELECT ID, Person
FROM Tweet,
     Author
```

JOIN CONDITION (WHERE) MISSING
The returned table is the Cartesian product of the records of the tables

Even when the JOIN CONDITION is specified

- First the system produces the Cartesian product table
- Then it selects records to keep based on the JOIN CONDITION

Aliasing

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

AUTHOR

Person	Tweet
s	389472
d	123794
j	596208
d	782138
d	127890
j	173902
s	893110

```
SELECT ID, Text
FROM Tweet AS t,
     Author AS a
WHERE t.ID = a.Tweet
    AND a.Person = "d"
```

JOIN CONDITION can be composed of multiple logical clauses by means of logical operators AND, OR

Aliasing is meant to avoid ambiguities!

Aliasing

TWEET

ID	Time	Text
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
596208	2019-01-02 3:14:15	:-D
782138	2019-01-04 15:04:57	1951A 4 lyfe
127890	2019-01-04 17:30:07	hey
173902	2019-01-05 3:34:18	i <3 1951A
893110	2019-01-06 12:21:53	i <3 1951A

```
SELECT ID, Text
FROM Tweet AS t,
     Author AS a
WHERE t.ID = a.Tweet
    AND a.Person = "d"
```

AUTHOR

Person	Tweet
s	389472
d	123794
j	596208
d	782138
d	127890
j	173902
s	893110

ID	Time	Text	Person	Tweet
123794	2019-01-01 12:34:57	lol	d	123794
782138	2019-01-04 15:04:57	1951A 4 lyfe	d	782138
127890	2019-01-04 17:30:07	hey	d	127890

Quiz 1

PERSON

Handle	Name
s	Sol
d	Diane
j	Josh

RETWEET

Person	Tweet
s	1
s	2
d	1

Design a SQL query to find the names of the people who retweeted

Quiz 1

PERSON

Handle	Name
s	Sol
d	Diane
j	Josh

RETWEET

Person	Tweet
s	1
s	2
d	1

(a)
```
SELECT Name
FROM PERSON AS p,
    RETWEET AS r
WHERE r.Person = p.Name
```

(b)
```
SELECT *
FROM PERSON AS p,
    RETWEET AS r
WHERE r.Person = p.Handle
```

(c)
```
SELECT Name
FROM PERSON AS p,
    RETWEET AS r
WHERE r.Person = p.Handle
```

Joins

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

```
SELECT ID, Text
FROM (TWEET JOIN AUTHOR
        ON ID = Tweet)
```

ID	Text
389472	hey
596208	:-D
173902	i <3 1951A
893110	i <3 1951A

MATCHING RECORDS
(I.e., ID=Tweet)

MISMATCHED RECORDS
No records in Tweet matching those in AUTHOR

Inner Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

```
SELECT ID, Text
FROM (TWEET JOIN AUTHOR
        ON ID = Tweet)
```

Person	Tweet	ID	Text
s	389472	389472	hey
j	596208	596208	:-D
j	173902	173902	i <3 1951A
s	893110	893110	i <3 1951A
~~d~~	~~672109~~		
		~~782138~~	~~1951A 4 lyfe~~

Left Outer Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

"left" table

```
SELECT ID, Text
FROM (TWEET LEFT OUTER JOIN AUTHOR
        ON ID = Tweet)
```

"right" table

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

Missing attribute values are set to NULL

Left Outer Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

"left" table

```
SELECT ID, Text
FROM (TWEET LEFT OUTER JOIN AUTHOR
        ON ID = Tweet)
```

"right" table

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

ID	Text
389472	hey
596208	:-D
173902	i <3 1951A
893110	i <3 1951A
782138	1951A 4 lyfe

Think of it as saying: Keep incomplete records from the left table (Tweet) and not matched by the right table (Author) and fill the missing values with NULL

Right Outer Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

```
SELECT ID, Text
FROM (TWEET RIGHT OUTER JOIN AUTHOR
        ON ID = Tweet)
```

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

ID	Text
389472	hey
596208	:-D
173902	i <3 1951A
893110	i <3 1951A
NULL	NULL

Full Outer Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

```
SELECT ID, Text
FROM (TWEET FULL OUTER JOIN AUTHOR
        ON ID = Tweet)
```

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

ID	Text
389472	hey
596208	:-D
173902	i <3 1951A
893110	i <3 1951A
NULL	NULL
782138	1951A 4 lyfe

Natural Join Condition

TWEET

TweetID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

```
SELECT ID, Text
FROM (TWEET JOIN AUTHOR)
```

AUTHOR

Person	TweetID
s	389472
j	596208
j	173902
s	893110
d	672109

- Assumes condition is ALL PAIRS of attributes with matching names
- Inner Join!

Natural Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

```
SELECT ID, Text
FROM (TWEET AS t(tweetid, text) JOIN
      AUTHOR AS a(person, tweetid)
```

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

person	tweetid	tweetid	text
s	389472	389472	hey
j	596208	596208	:-D
j	173902	173902	i <3 1951A
s	893110	893110	i <3 1951A

- Aliasing is useful to assign matching names to attributes with different names from different tables
  - Alias name and “original” name can be used interchangeably

Natural Join

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

```
SELECT ID, Text
FROM (TWEET AS t(tweetid, text) JOIN
      AUTHOR AS a(person, tweetid)
```

ID	Text
389472	hey
596208	:-D
173902	i <3 1951A
893110	i <3 1951A

Quiz 2

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

```
SELECT ID, Text
FROM (TWEET AS t(tweetid, foo) JOIN
      AUTHOR AS a(foo, tweetid)
```

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

- What would happen in this case?

Quiz 2

TWEET

ID	Text
389472	hey
596208	:-D
782138	1951A 4 lyfe
173902	i <3 1951A
893110	i <3 1951A

```
SELECT ID, Text
FROM (TWEET AS t(tweetid, foo) JOIN
      AUTHOR AS a(foo, tweetid)
```

foo	tweetid

AUTHOR

Person	Tweet
s	389472
j	596208
j	173902
s	893110
d	672109

- No matches are found!

Quiz 3

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

```
SELECT Name, Course
FROM (STUDENT LEFT OUTER JOIN GRADES ON ID = Student)
```

Name	Course
Diane	32
Sol	1951A
NULL	32

(a)

Name	Course
Diane	32
Sol	1951A
Josh	NULL
Karlly	NULL
Mounika	NULL

(b)

ORDERED keyword

TWEET

ID	Time	Text
782138	2019-01-04 15:04:57	1951A 4 lyfe
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
127890	2019-01-04 17:30:07	hey
893110	2019-01-06 12:21:53	i <3 1951A
596208	2019-01-02 3:14:15	:-D
173902	2019-01-05 3:34:18	i <3 1951A

```
SELECT Text
FROM Tweet
ORDER BY Time
```

Records are sorted in increasing order according to the selected attribute values

ORDERED keyword

TWEET

ID	Time	Text
782138	2019-01-04 15:04:57	1951A 4 lyfe
389472	2019-01-01 12:34:56	hey
123794	2019-01-01 12:34:57	lol
127890	2019-01-04 17:30:07	hey
893110	2019-01-06 12:21:53	i <3 1951A
596208	2019-01-02 3:14:15	:-D
173902	2019-01-05 3:34:18	i <3 1951A

```
SELECT Text
FROM Tweet
ORDER BY ID
```

Different data types have different ordering criteria:

- Natural ordering for numeric types
- Alphanumeric ordering for string types
- Timestamps and other types have their total ordering

GROUP BY

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text,
Count(*), AVG(Likes)
FROM Tweet
GROUP BY Text
```

Text	Count(*)	AVG(Likes)
lol	1	100
hey	2	5
i <3 1951A	2	504,000,000
:-D	1	1
1951A 4 lyfe	1	1,000

Used to group records according to the value of chosen attributes

- Count(*) yields the number of records grouped together
- Can generate values obtained by combining the records being grouped

GROUP BY

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text,
MAX(Likes),MIN(Likes)
FROM Tweet
GROUP BY Text
```

Text	MAX(Likes)	MIN(LIKES)
lol	100	100
hey	10	0
i <3 1951A	1000000000	8000000
:-D	1	1
1951A 4 lyfe	1,000	1000

- Possible aggregation criteria: Count(*), SUM, MAX, MIN, AVG... useable on numeric fields

HAVING

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text,
Count(*), AVG(Likes)
FROM Tweet
GROUP BY Text
HAVING COUNT(*) > 1
```

Text	Count(*)	AVG(Likes)
hey	2	5
i <3 1951A	2	504,000,000

- Similar behavior to "WHERE", but only used with aggregations/GROUP BY
- Filters groups based on the specified property

LIKE

TWEET

ID	Likes	Text
782138	1,000	1951A 4 lyfe
389472	10	hey
123794	100	lol
127890	0	hey
893110	8,000,000	i <3 1951A
596208	1	:-D
173902	1,000,000,000	i <3 1951A

```
SELECT Text, Count(*),
AVG(Likes)
FROM Tweet
WHERE Text LIKE '%1951A%'
GROUP BY Text
```

- LIKE is used in a WHERE condition to formulate a requirement in which one wants to detect a "pattern"
- Two possible wildcards "%" and "_":
  - "_" any character
  - "%" any string of characters including the empty one
- Is it possible to compose arbitrarily many conditions using AND, OR operators

LIKE

- Examples:
  - WHERE CustomerName LIKE 'a%': Finds any values that start with "a"
  - WHERE CustomerName LIKE '%a': Finds any values that end with "a"
  - WHERE CustomerName LIKE '%or%':
    - Finds any values that have "or" in any position
  - WHERE CustomerName LIKE '_r%': Finds any values that have "r" in the second position
  - WHERE CustomerName LIKE 'a_%': Finds any values that start with "a" and are at least 2 characters in length
  - WHERE CustomerName LIKE 'a__%':
    - Finds any values that start with "a" and are at least 3 characters in length
  - WHERE ContactName LIKE 'a%o':
    - Finds any values that start with "a" and ends with "o"

IN

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

```
SELECT Name
FROM STUDENT
WHERE ID IN
   (SELECT Student
    FROM GRADES
    WHERE Course = 1951A
    )
```

- IN allows to compare and handle sets of records
- E.g., the proposed query can be interpreted as "Find Names of students which completed 1951A"
- IN allows to formulate a sub-query which selects records
- In the example the operation can be seen as the intersection (based on the ID) of records in STUDENT and those in GRADES which attended 1951A

IN

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	32	A
2	1951A	A
6	32	A

```
SELECT Name
FROM STUDENT
WHERE ID IN
   (SELECT Student
    FROM GRADES
    WHERE Course = 1951A
    )
```

Name
Sol

ALL

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = "1951A"
  AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

- Similar to IN
- ALL allows to formulate a sub-query which selects a bag of records and then evaluate the selected parameter (E.g., grade) against ALL records selected by the sub-query
- In the example selects the grades of students that took 1951A and received the maximum grade among all such students

ALL

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Student
FROM GRADES
WHERE Course = "1951A"
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

Student
1
2

ANY

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = "1951A"
   AND Grade >= ALL
  (SELECT Grade
   FROM GRADES
   WHERE Course = 1951A
   )
```

ANY

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT Grade
FROM GRADES
WHERE Course = "1951A"
   AND Grade > ANY
   (SELECT Grade
    FROM GRADES
    WHERE Course = 1951A
    )
```

- ANY allows to formulate a sub-query which selects a bag of records and then evaluate the selected parameter (E.g., grade) against ANY of records selected by the sub-query
- In the example selects the grades of students that took 1951A and received a grade higher than at least one (i.e., not the minimum grade)

DISTINCT

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT DISTINCT Grade
FROM GRADES
WHERE Course = "1951A"
    AND Grade >= ALL
   (SELECT Grade
    FROM GRADES
    WHERE Course = 1951A
    )
```

- Removes duplicates with respect to the selected attribute
- Set operations (Union, Intersection, etc.) remove duplicates by default.

EXISTS

STUDENT

ID	Name
1	Diane
2	Sol
3	Josh
4	Karlly
5	Mounika

GRADES

Student	Course	Grade
1	1951A	3.5
2	1951A	3.5
6	1951A	2.8

```
SELECT NAME
FROM STUDENT s
WHERE NOT EXISTS
   (SELECT *
    FROM GRADES
    WHERE Course = 1951A
    AND Student = s.ID
    )
```

- Used to realize set operation intersection (EXISTS), and set difference (NOT EXISTS)
- In the example, select from students from STUDENT of they did not take CS1951

The NULL value

- If an operand of an operation is NULL, the result is NULL:
  - NULL + 1 = NULL
  - NULL * 0 = NULL
- Comparisons: All comparisons that involve a null value, evaluate to unknown
  - NULL = NULL -> Unknown
  - NULL != NULL -> Unknown
  - NULL < 13 -> Unknown
  - NULL > NULL -> Unknown

Logical operations

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

Unknown and logical operations

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
TRUE	UNK	TRUE	UNK	UNK
FALSE	UNK	UNK	FALSE	UNK
UNK	TRUE	TRUE	UNK	UNK
UNK	FALSE	UNK	FALSE	UNK
UNK	UNK	UNK	UNK	UNK

NULL values and WHERE statements

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT COUNT(*)
FROM TWEET
WHERE Likes > 10
```

- Only tuples which evaluate to true are part of the query result.
- Unknown and false treated equivalently.

NULL values and GROUP BY statements

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT Text, COUNT(*)
FROM TWEET
GROUP BY Text
```

- If there are NULL values, group them up
- CAREFUL: This may seem in contradiction with what we said earlies about NULL = NULL being FALSE

NULL values in predicates

TWEET

ID	Text	Likes
389472	NULL	100
123794	NULL	3
596208	:-D	NULL
782138	1951A 4 lyfe	NULL
173902	i <3 1951A	19
893110	i <3 1951A	7539

```
SELECT Text ID
FROM TWEET
WHERE Text IS NULL
```

- Use “IS NULL” (or “IS NOT NULL”) rather than the standard “=”

NULL values and GROUP BY

- `COUNT(att)`: NULL is ignored
- `SUM(att)`: NULL is ignored
- `AVG(att)`: ration of results from `SUM` and `COUNT`
- `MIN(att)` and `MAX(att)`: NULL is ignored
- Exception! If NULL is the only value in the column, then `SUM`/`AVG`/`MIN`/`MAX` all return "NULL"

Quiz 5

RUNNERS

ID	Name
1	Diane
2	Sol
3	Josh
4	Jon

RACES

Event_ID	Event	Winner_ID
1	PVD Marathon	2
2	PVD Half	3
3	PVD 2 yard jump	2
4	Race4NULL: Raising Awareness	NULL

What is the result of this query?

```
SELECT COUNT(*)
FROM RUNNERS
WHERE ID NOT IN SELECT(Winner_ID FROM RACES)
```

(a)

Count(*)
0

(b)

Count(*)
1

(c)

Count(*)
2

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

What is the order of execution of the operations?

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

↓

ID	Text
389472	hey

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

What is the order of execution of the operations?

FROM

|

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

↓

ID	Text
389472	hey

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

ID	Text
389472	hey

What is the order of execution of the operations?

FROM

|

WHERE

|

Execution order

TWEET

ID	Time	Text
389472	12:34:5	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

What is the order of execution of the operations?

FROM
|
WHERE
|
SELECT

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

ID	Text
389472	hey

Execution order

TWEET

ID	Time	Text
389472	12:34:5	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

What is the order of execution of the operations?

FROM
|
WHERE
|
SELECT

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

ID	Text
389472	hey

Still other executions tree may be equivalent!

FROM
|
SELECT
|
WHERE

Execution order

TWEET

ID	Time	Text
389472	12:34:5	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

ID	Text
389472	hey

So which execution tree is better?

- `WHERE(SELECT(FROM))`
- `SELECT (WHERE(FROM))`
- They are equally as good

Execution order

TWEET

ID	Time	Text
389472	12:34:5	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Text
FROM TWEET
WHERE Text = "hey"
```

ID	Text
389472	hey

So which execution tree is better?

- `WHERE(SELECT(FROM))`
- `SELECT (WHERE(FROM))`
- They are equally as good

But didn't we say earlier they are equivalent?

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

SQL

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Execution Tree

```
WHERE(SELECT(FROM))
```

```
SELECT(WHERE(FROM))
```

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Execution Tree

```
WHERE(SELECT(FROM))
```

```
SELECT(WHERE(FROM))
```

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Execution Tree

```
WHERE(SELECT(FROM))
```

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

ID	Time
389472	12:34:56
123794	12:34:57
596208	3:14:15
782138	15:04:57
173902	3:34:18
893110	12:21:53

Execution Tree

```
WHERE (SELECT (FROM) )
```

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

ID	Time
389472	12:34:56

Execution Tree

```
WHERE(SELECT(FROM))
```

The information required to applied the WHERE command was lost in the previous steps

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Execution Tree

```
SELECT(WHERE(FROM))
```

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

ID	Time	Text
389472	12:34:56	hey

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Execution Tree

```
SELECT(WHERE(FROM))
```

Execution order

TWEET

ID	Time	Text
389472	12:34:56	hey
123794	12:34:57	lol
596208	3:14:15	:-D
782138	15:04:57	1951A 4 lyfe
173902	3:34:18	i <3 1951A
893110	12:21:53	i <3 1951A

ID	Time
389472	12:34:56

```
SELECT ID, Time
FROM TWEET
WHERE Text = "hey"
```

Execution Tree

```
SELECT(WHERE(FROM))
```

Canonical Execution Order

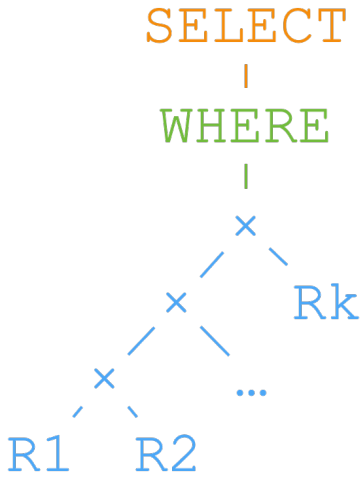
Execution order and memory requirement

Consider the following query over relations R1,R2,...,Rk each with m tuples:

```
SELECT A1…An
FROM R1…Rk
WHERE P
```

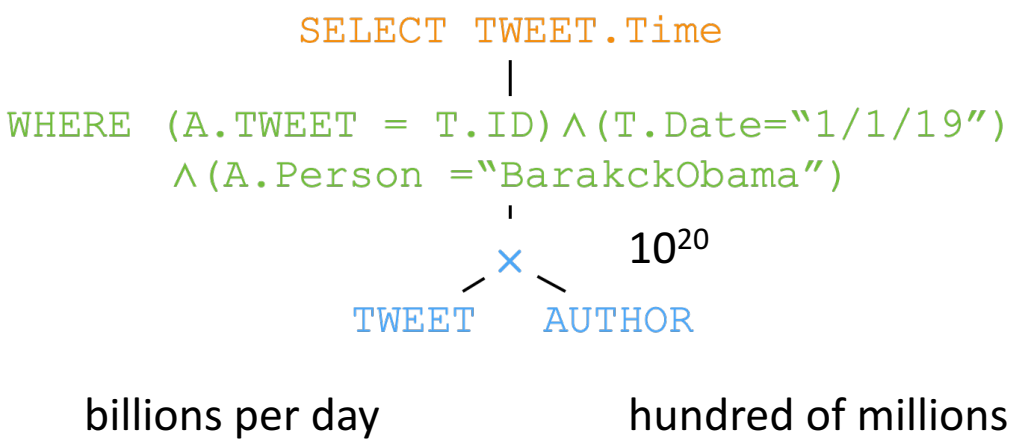
Assume each tuple in the each of the original k relations requires one memory location. How much overall memory is used by the query?

- $\approx m^k$
- $\approx m \times k$
- $\approx m + k$



Execution order and memory requirement

```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == '01/01/2019'
      and AUTHOR.Person = "BarackObama"
```



Extreme memory load!
Can we do anything to improve this?

Execution order and memory requirement

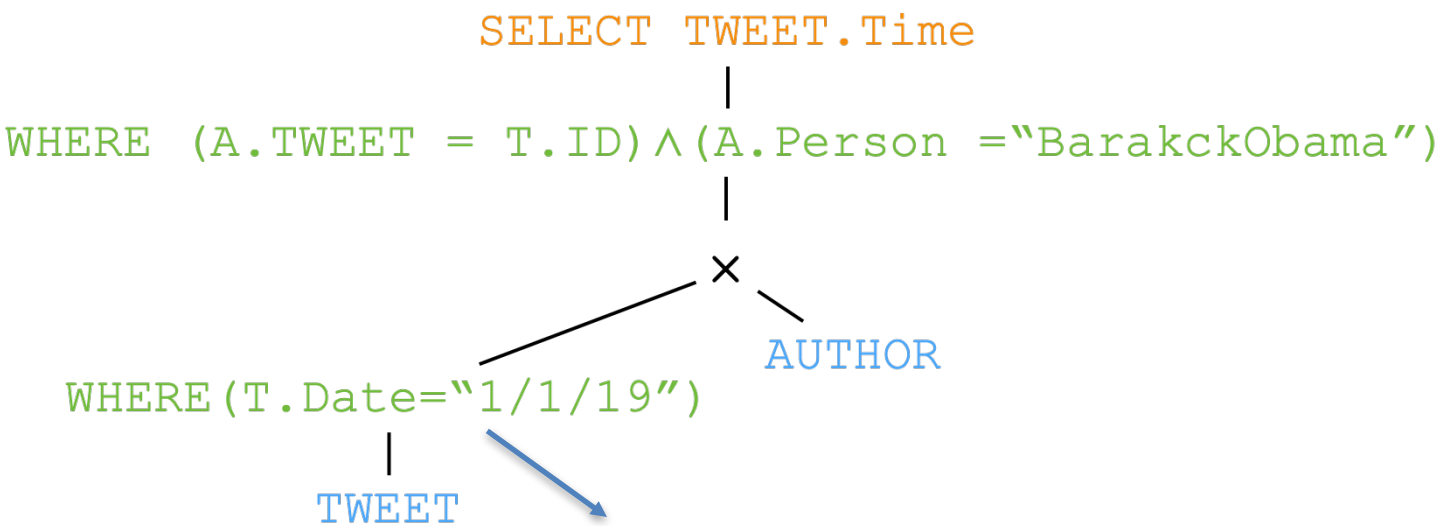
```
SELECT TWEET.Time
FROM TWEET, AUTHOR
WHERE AUTHOR.TWEET = TWEET.ID
      and TWEET.Date == '01/01/2019'
      and AUTHOR.Person = "BarackObama"
```

```
SELECT TWEET.Time
|
WHERE (A.TWEET = T.ID)∧(T.Date="1/1/19")
      ∧(A.Person ="BarakckObama")
|
×
TWEET   AUTHOR
```

much smaller subsets of original

Execution order and memory requirement

```
SELECT TWEET.Time
FROM AUTHOR,
(SELECT Tweet.ID
FROM TWEET
WHERE TWEET.Date == '01/01/2019') c
WHERE AUTHOR.TWEET = c.ID
      and AUTHOR.Person = "BarackObama"
```



By applying first intra-relation filters we can considerably reduce the memory requirements

Execution order and memory requirement

```
SELECT TWEET.Time
FROM
(SELECT Tweet.ID
FROM TWEET
WHERE TWEET.Date == '01/01/2019') c,
(SELECT AUTHOR.Tweet
FROM AUTHOR
WHERE AUTHOR.Person = "BarackObama") a
WHERE a.Tweet = c.ID
```