CS 217 Data Management and Information Processing

Other Topics in Database

Outline

- Division Query
- Indexing
- SQL on a remote server

“Division” Query

Division

Division is typically required when you want to find out entities that are interacting with **all entities** of a set of different type entities.

Table1

Supplier	Component
Apple	CPU
Intel	CPU
Intel	SSD
Samsung	CPU
Samsung	SSD
Samsung	memory
Samsung	GPU
Samsung	monitor
…	

Table2

Component
CPU
SSD
memory
monitor
motherboard

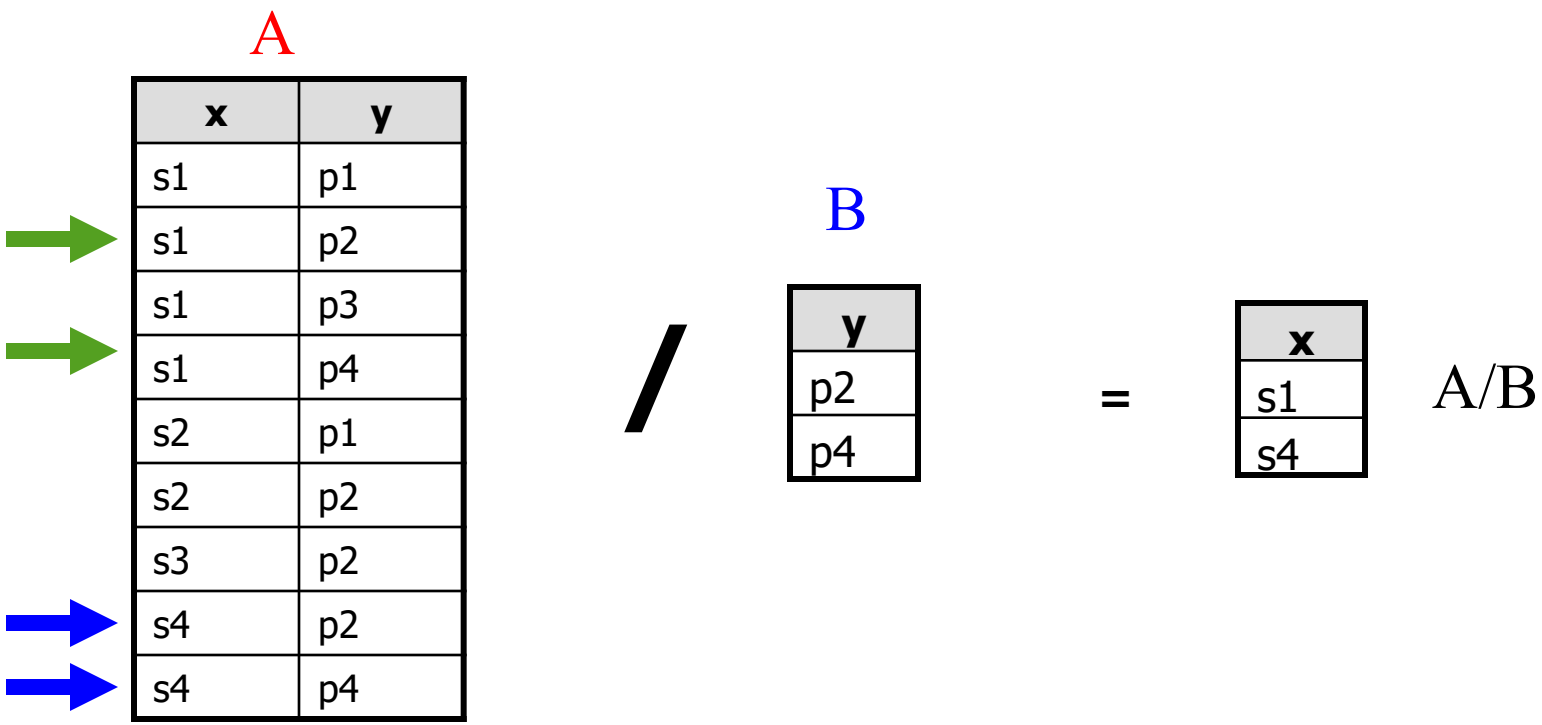
Find out the List of Suppliers that Can Provide All Components in Table2

Division

Let A have two fields x and y

Let B have one field y

A/B contains all x, such that for **every** y in B there is a (x, y) tuple in A



Steps to Compute Division

1. Compute all possible combinations of the first column of A and B.
2. Remove those rows that exist in A
3. Keep only the first column of the result. These are the ***disqualified*** values
4. A/B is the first column of A **except** the disqualified values

Division Example: Step 1

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

x
s1
s2
s3
s4

×

y
p2
p4

=

x	y
s1	p2
s1	p4
s2	p2
s2	p4
s3	p2
s3	p4
s4	p2
s4	p4

A

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

/

B

y
p2
p4

(SELECT DISTINCT x FROM A) CROSS JOIN B

1. Compute all possible combinations of the first column of A and B.
2. Remove those rows that exist in A
3. Keep only the first column of the result. These are the *disqualified* values
4. A/B is the first column of A except the disqualified values

Division Example : Step 2

x	y
s1	p2
s1	p4
s2	p2
s2	p4
s3	p2
s3	p4
s4	p2
s4	p4

−

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

=

x	y
s2	p4
s3	p4

A

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

/

B

y
p2
p4

```
A EXCEPT (
        (SELECT DISTINCT x FROM A) CROSS JOIN B
)
```

1. Compute all possible combinations of the first column of A and B.
2. Remove those rows that exist in A
3. Keep only the first column of the result. These are the *disqualified* values
4. A/B is the first column of A except the disqualified values

Division Example : Step 3

A

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

/

B

y
p2
p4

x	y
s2	p4
s3	p4

x
s2
s3

```
SELECT DISTINCT x FROM (
        A EXCEPT  (
                (SELECT DISTINCT x FROM A) CROSS JOIN B
        )
)
```

1. Compute all possible combinations of the first column of A and B.
2. Remove those rows that exist in A
3. Keep only the first column of the result. These are the *disqualified* values
4. A/B is the first column of A except the disqualified values

Division Example : Step 4

x
s1
s2
s3
s4

−

x
s2
s3

=

X
S1
S4

A

x	y
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

/

B

y
p2
p4

```
SELECT x from A EXCEPT (
    SELECT DISTINCT x FROM (
        A EXCEPT (
            (SELECT DISTINCT x FROM A) CROSS JOIN B
        )
    )
)
```

1. Compute all possible combinations of the first column of A and B.
2. Remove those rows that exist in A
3. Keep only the first column of the result. These are the *disqualified* values
4. A/B is the first column of A except the disqualified values

Division query

Find the Employment numbers of the pilots who can fly **all** MD planes

Employment

Emp_No	Model_No
1001	B727
1001	B747
1001	DC10
1002	A320
1002	A340
1002	B757
1002	DC9
1003	A310
1003	DC9
1003	DC10

Model

Maker	Model_No
Airbus	A310
Airbus	A320
Airbus	A330
Airbus	A340
Boeing	B727
Boeing	B747
Boeing	B757
MD	DC10
MD	DC9

Emp_No
1003

Indexing

Indexing

- When working with large amounts of data it can be a challenge to find an item of interest.
- We don’t want to request every storage address to find what we’re looking for.
- ***Sorting*** the data can help tremendously, because it allows *binary search*.

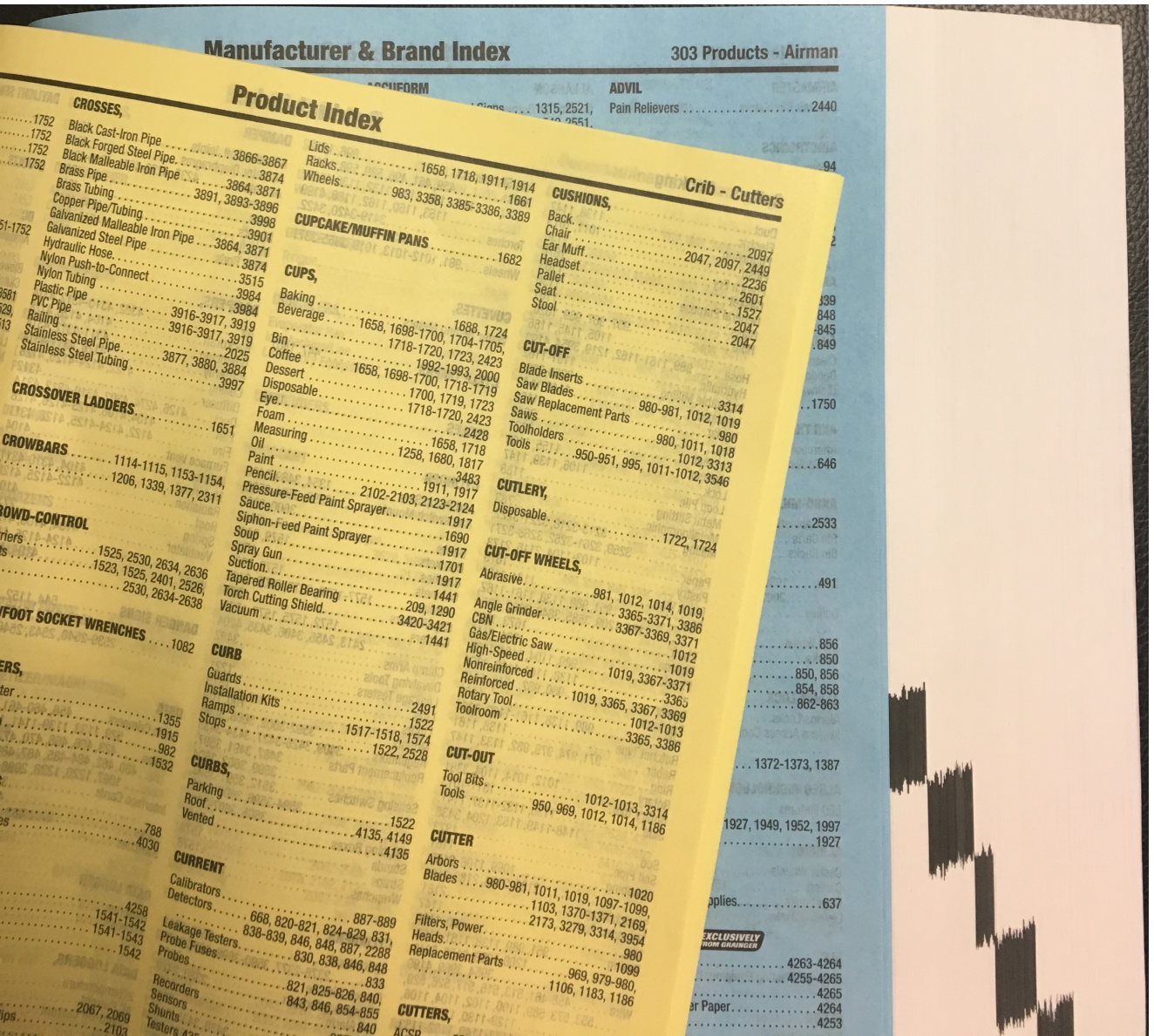
Sorting and Binary Search

- We know it's easy to find data if it's in a ***sorted*** list.
  - That's why printed dictionaries and phone books are alphabetical.
- **Binary Search** is how computers find entries in a sorted list.
  - Let's say you're looking for the word "key" in a list of 10,000 words
    1. Compare "key" to the word in the middle position (5,000th word).
    2. If you're lucky and that middle word is ***equal to*** "key", then you're done!
    3. If the middle word is ***greater than*** "key" then go back to step 1, but refine your search to just the left half of the list (words 0 through 4,999).
    4. If the middle word is ***less than*** "key" then go back to step 1, but refine your search to just the right half of the list (words 5,001 through 10,000).
  - At most will take $\log_2 N$ steps to find the entry, where N is the list size.
    - Eg., 32 steps for binary search in a list of 4 billion entries (because $2^{32} \cong 4$ billion)

Why sorting is not enough

- You can't sort in **multiple dimensions**
  - Let's say you want to find a product quickly according to either it's name, manufacturer, or price. You can only sort by one of the three columns.
- Can't **insert new data** without *shifting* everything over to make room.
  - Shifting data in storage would require rewriting about half of it (on average).
  - That's incredibly amount of work to accommodate just one tiny addition.
- Sorting doesn't take advantage of the hardware's storage hierarchy.
  - The binary search will have to access the disk in every step because the index is distributed over the full data set.
  - It would be better to put all the index data close together (spatial locality).

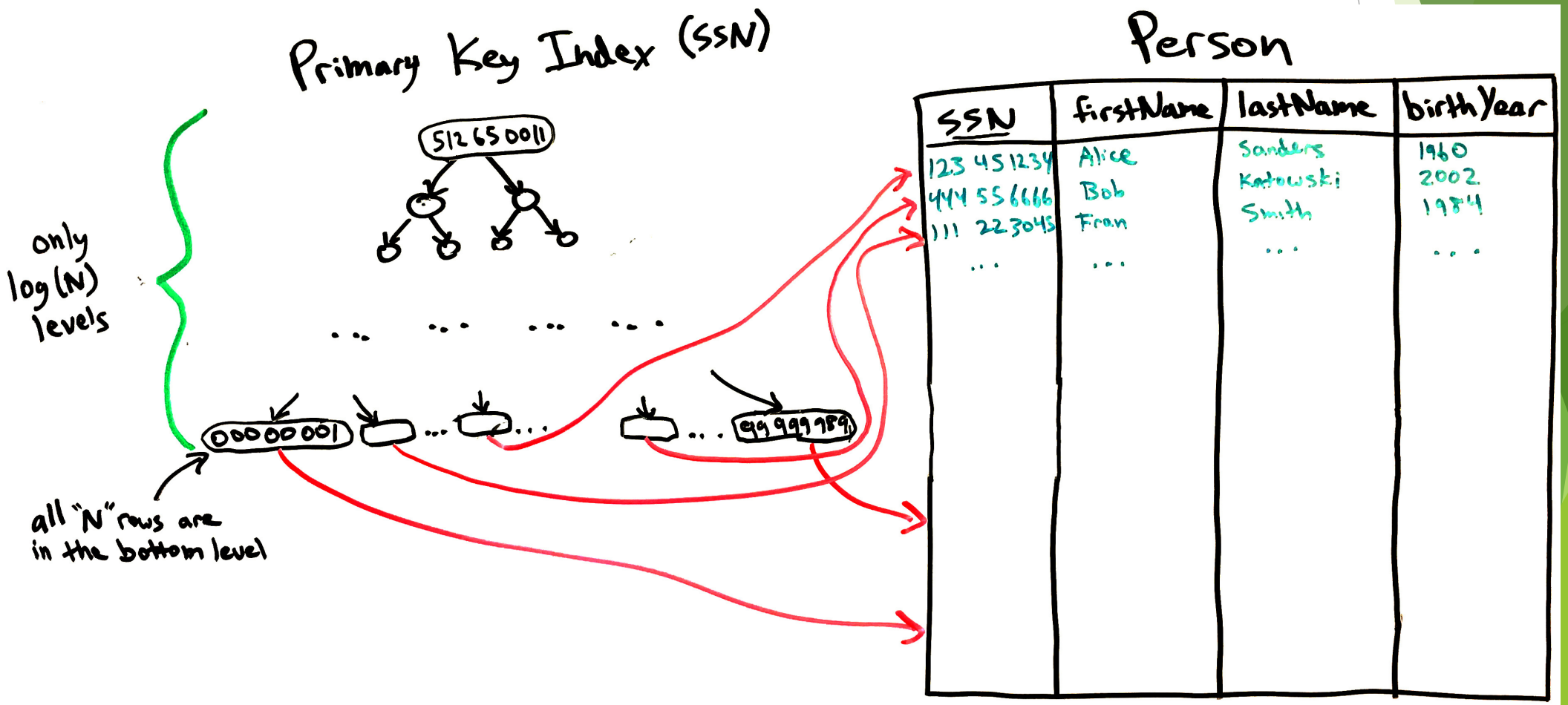
A printed catalog can add multiple indexes



- Grainger catalog is sorted according to high-level *product categories*.
- It has both yellow and blue index pages.
- These allow efficient lookup by:
  - *product type names*
  - *manufacturer names*
- In total, products can be efficiently found in three ways.
- Simple sorted lists are effective here because data is never added.

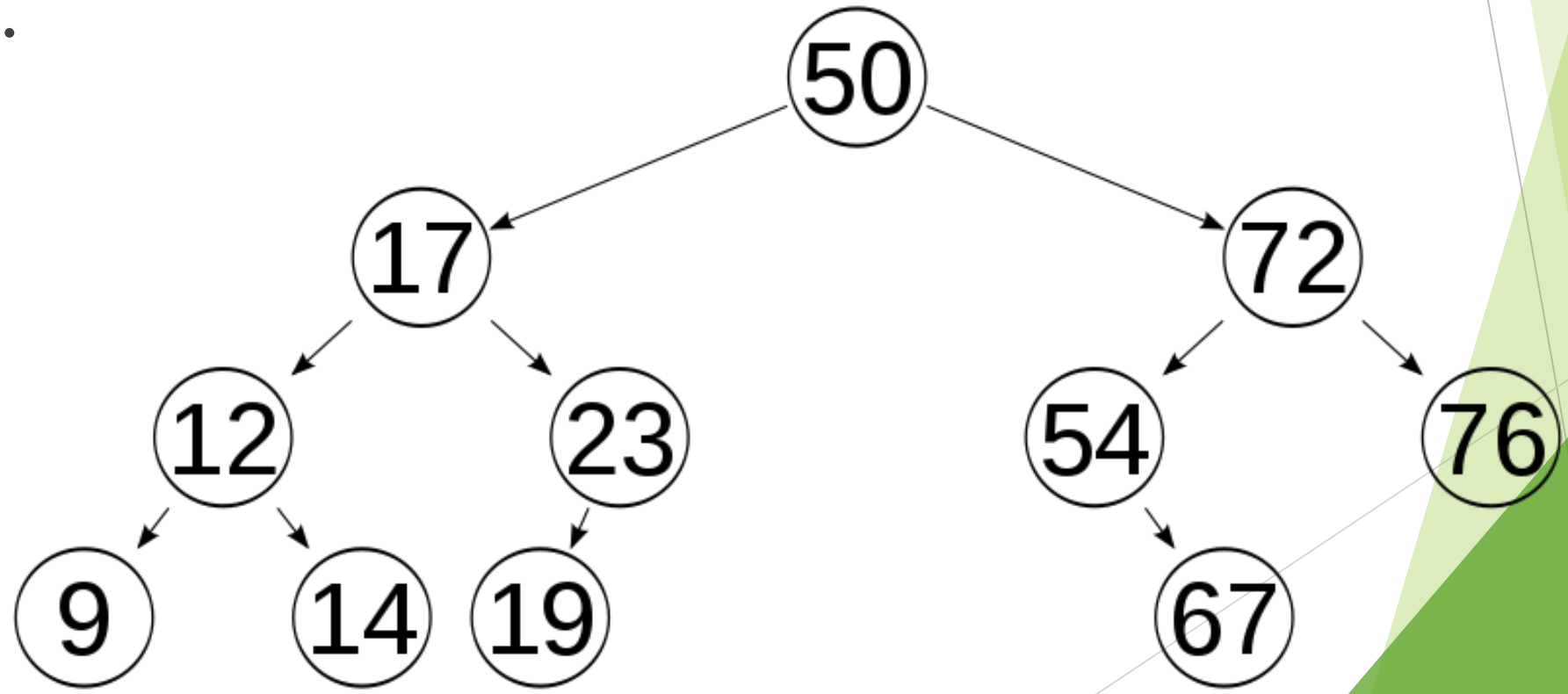
DB Indexes Use Trees

- Self-balanced binary trees give the *log(N)* speed of a binary search, while also allowing entries to be quickly added and deleted.
- The details are beyond scope of this class (covered in CS-214 Data Structures).



Balanced binary search tree

- Finding an element is very similar to binary search of a sorted list.
- Start from the root. Move to the **left subtree** if the value you're looking for is smaller, otherwise move to the **right subtree**.
- Repeat.



Creating indexes/keys

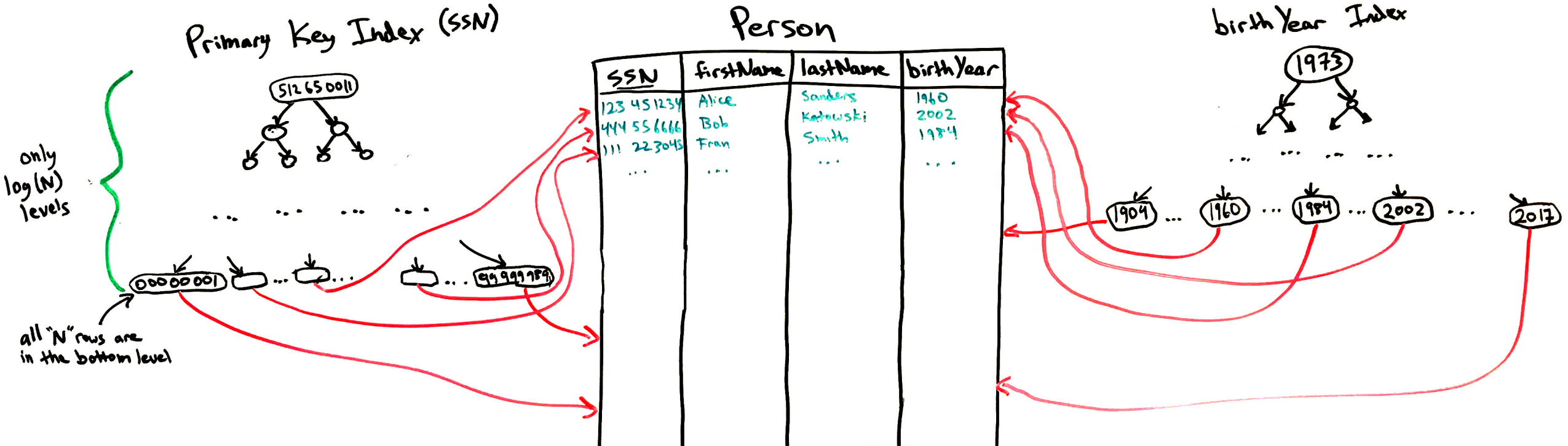
- Indexes are usually defined when the table is created
  - ***Primary key*** must be unique for each row.
  - We must be able to quickly check that new value does not already exist.
  - Thus, unique/primary keys are indexed.
- But you may later realize that certain queries are too slow
  - Without proper indexes, DBMS will have to examine every row in the table to find the relevant rows.
  - Adding one or more indexes may dramatically speed up a query.

Basic syntax:

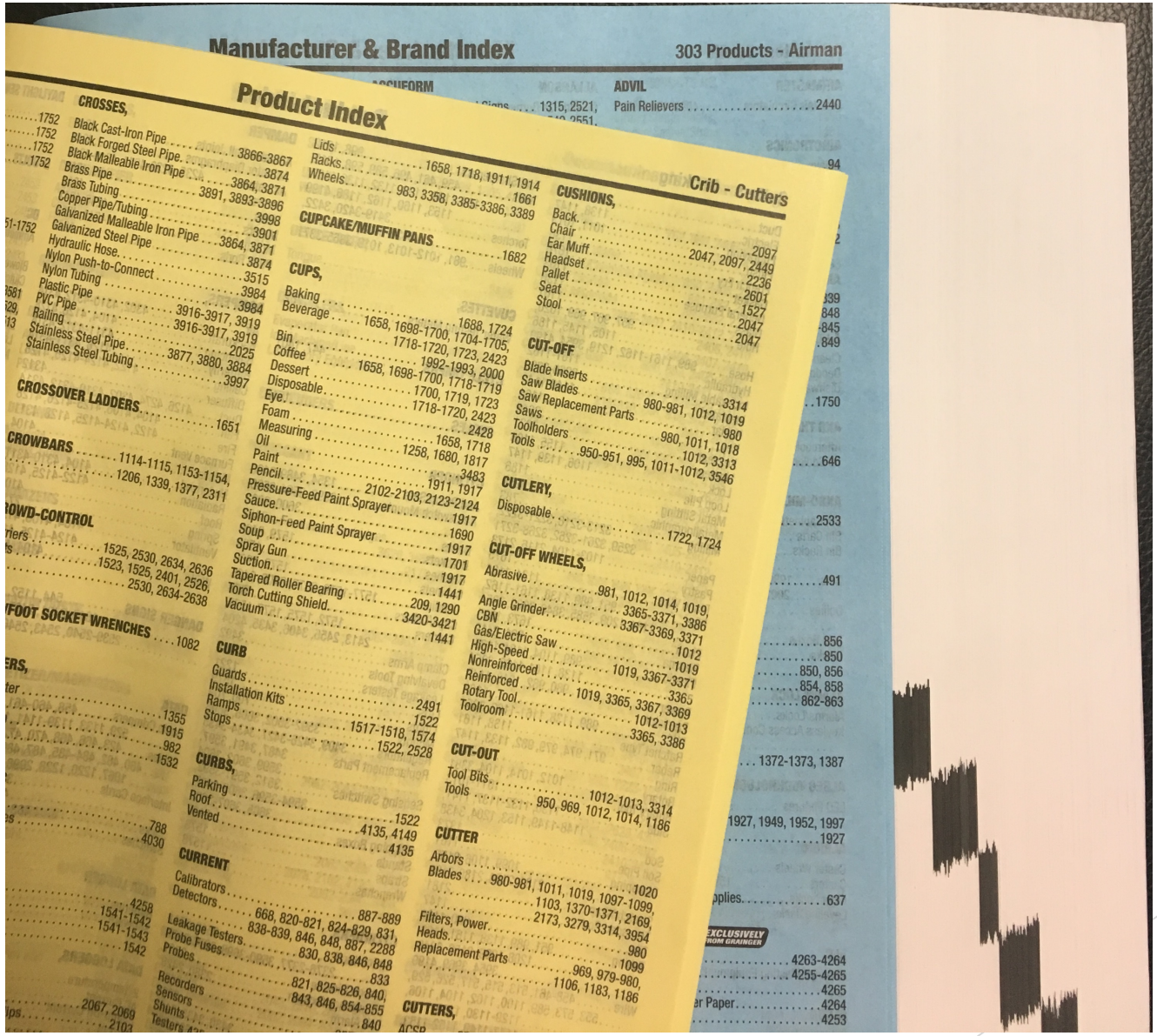
```
CREATE INDEX index_name ON table_name (column_name)
```

Multiple indexes in one table are possible

- Allow finding rows quickly based on multiple criteria
- Need two indexes to quickly get results for both:
  - `SELECT * FROM Person` **`WHERE SSN=543230921`**
  - `SELECT * FROM Person`
    `WHERE birthYear BETWEEN 1979 AND 1983`



A printed catalog can add multiple indexes

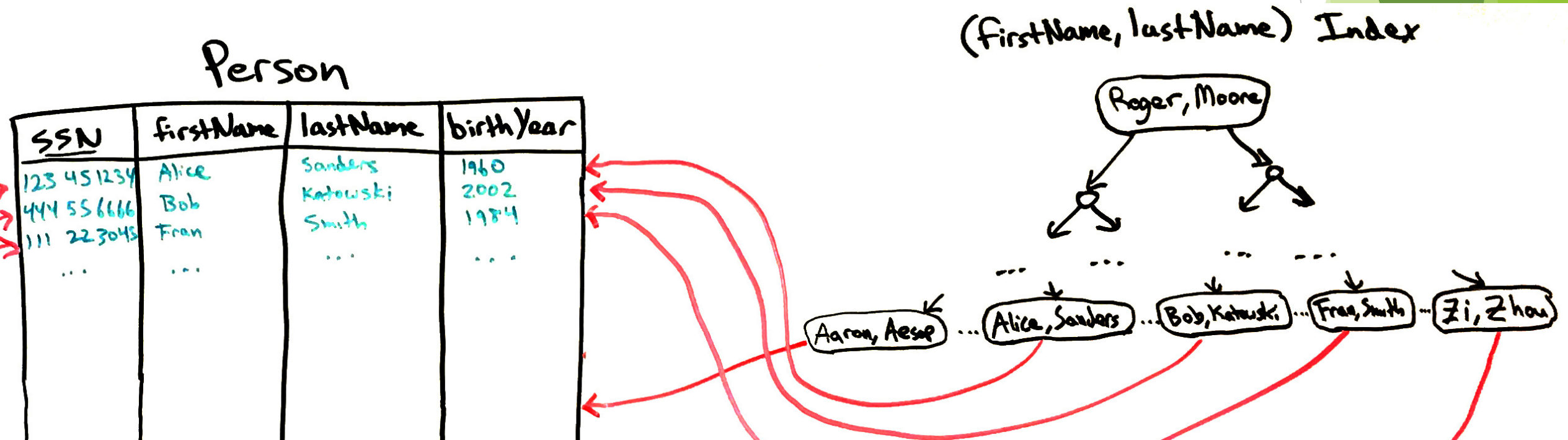


Composite indexes involve multiple columns

- Useful when WHERE clauses involves pairs of column values:

```
SELECT * FROM Person WHERE firstName = "Alice" AND lastName = "Sanders"
```

- Unlike two separate indexes, you can find the *matching pair* of values with one lookup.
- Otherwise, would have to first find results for `firstName = "Alice"` and scan through all the Alices checking for `lastName = "Sanders"`
- However, example below does not allow you to quickly find rows by lastName



When to index columns?

- When a query is slow!
- Generally, add an index if the column is:
  - Used in `WHERE` conditions, or
  - Used in `JOIN … ON` conditions, or
  - A foreign key refers to it.
- Also helpful if the column is:
  - In a `MIN` or `MAX` aggregation function

Indexes are not free!

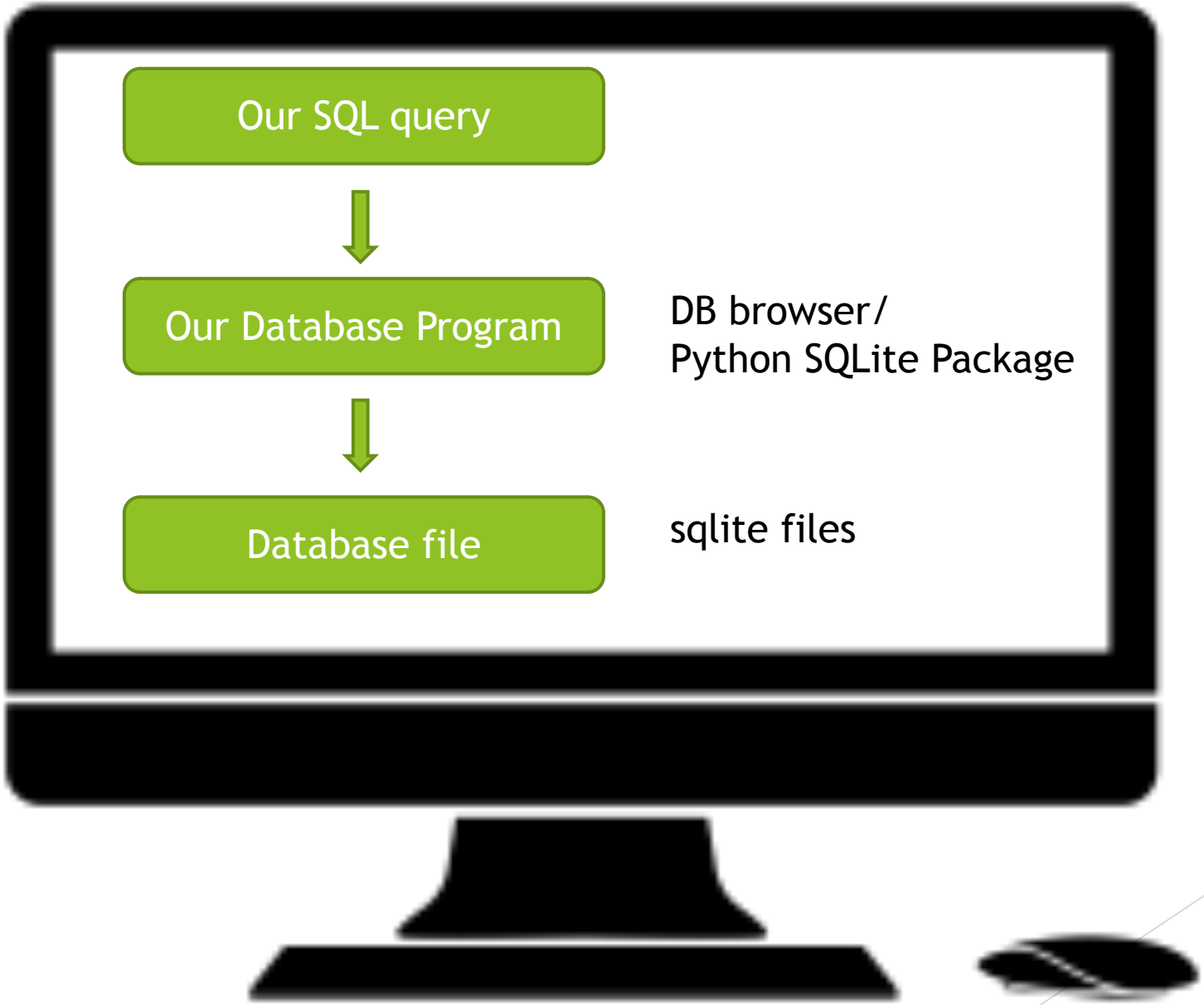
- Don’t add indexes unless you need them.
- Rookie mistake is to index every column “just in case.”
- Indexes consume storage space ***(storage overhead)***,
- Indexes must be updated when data is modified ***(performance overhead)***.

Key and Index terminology in SQL

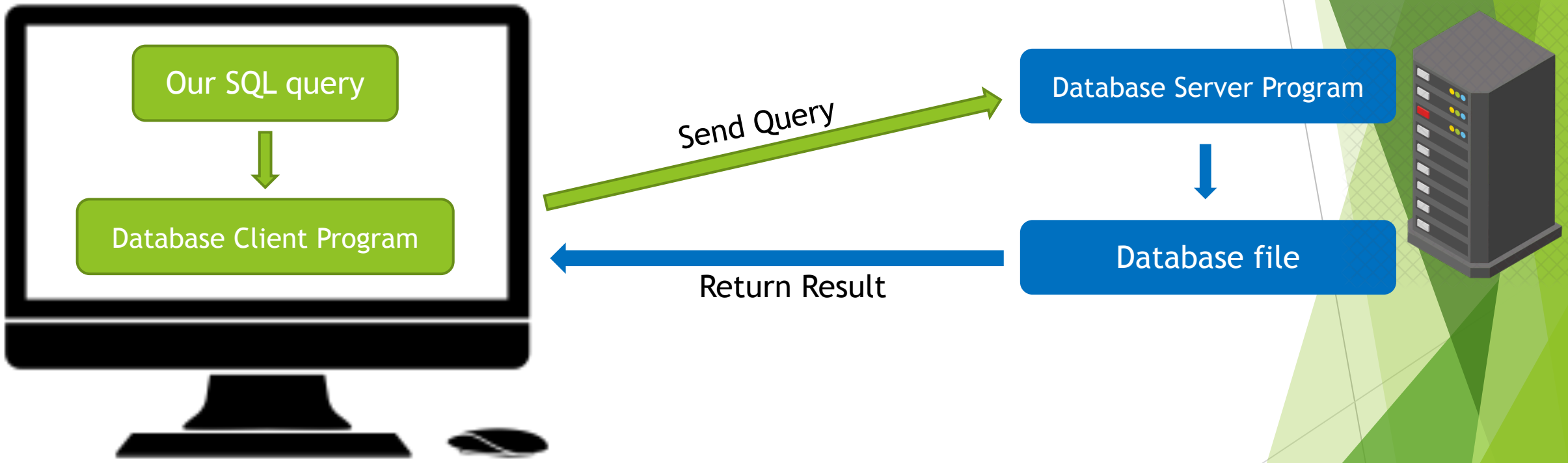
- Plain **key** or **index** is just a way to find rows quickly
  - Just creates a search tree.
- **Unique key** is an index that prevents duplicates
  - Bottom level of search tree has no repeated values
  - DBMS can use the tree to quickly search for existing rows with that value before allowing a row insertion (or column update) to proceed.
- **Primary key** is just a unique key, but there can only be one per table
  - We think of the primary key as the *most important* unique key in the table

SQL on a Remote Server

What We Covered in this Course



SQL Can be Executed Remotely Too!



Comparison

- Similarities
  - The SQL schema, SQL query are the same
  - You write the same SQL query and the result will be the same
- Differences
  - Usually you need to download a SQL client program used to connect to the SQL server
  - Authentication is needed to verify that you are authorized to access the data.

MySQL as an Example

IP address of the server

User name and password

Database name

```
import MySQLdb

# Open database connection
db = MySQLdb.connect("localhost","testuser","test123","TESTDB" )

# prepare a cursor object using cursor() method
cursor = db.cursor()

# execute SQL query using execute() method.
cursor.execute("SELECT VERSION()")

# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print "Database version : %s " % data

# disconnect from server
db.close()
```

SQL on a Server

- From the users' perspective, not much difference
- The underlying program can be more complicated than single-machine databases
  - Network
  - Authentication
  - Concurrent accesses
  - Access control policy
  - …