Optimizing Database Performance

Database Design

Department of Computer Engineering

Sharif University of Technology

Maryam Ramezani maryam.ramezani@sharif.edu

Introduction

Motivation

- ❑ DBMS stores vast quantities of data
- ❑ Data is stored on external storage devices and fetched into main memory as needed for processing
- ❑ Page is unit of information read from or written to disk. (in DBMS, page may have size 8KB or more).
- ❑ Data on external storage devices:
  - ○ Disks: Can retrieve random page at fixed cost
    But reading several consecutive pages is much cheaper than reading them in random order
  - ○ Tapes: Can read pages only in sequence
    Cheaper than disks; used for archival storage
- ❑ Cost of page I/O dominates cost of typical database operations

Files and indices

- **File organization:**
  - Method of arranging a file of records on external storage.
  - Record id (rid) is sufficient to physically locate a record
- **Indexes:**
  - Indexes are data structures that allow us to find the record ids of records with given values in index search key fields

Alternative File Organizations

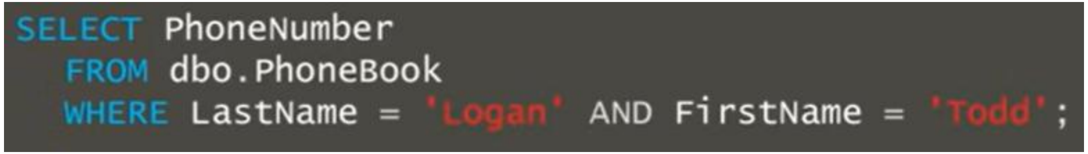
Many alternatives exist, each ideal for some situations, and not so good in others:

- Heap (random order) files: Suitable when typical access is a file scan retrieving all records.
- Sorted Files: Best if records must be retrieved in some order, or only a `range' of records is needed.
- Indexes: Data structures to organize records via trees or hashing.
  - Like sorted files, they speed up searches for a subset of records, based on values in certain ("search key") fields
  - Updates are much faster than in sorted files.

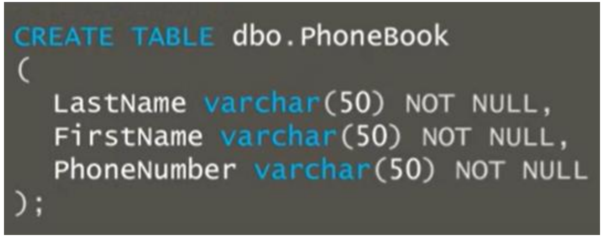
Indexing

Introduction

- **Scan Search**

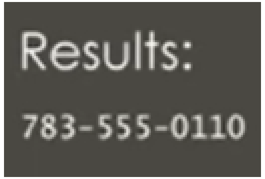


```
SELECT PhoneNumber
  FROM dbo.PhoneBook
  WHERE LastName = 'Logan' AND FirstName = 'Todd';
```



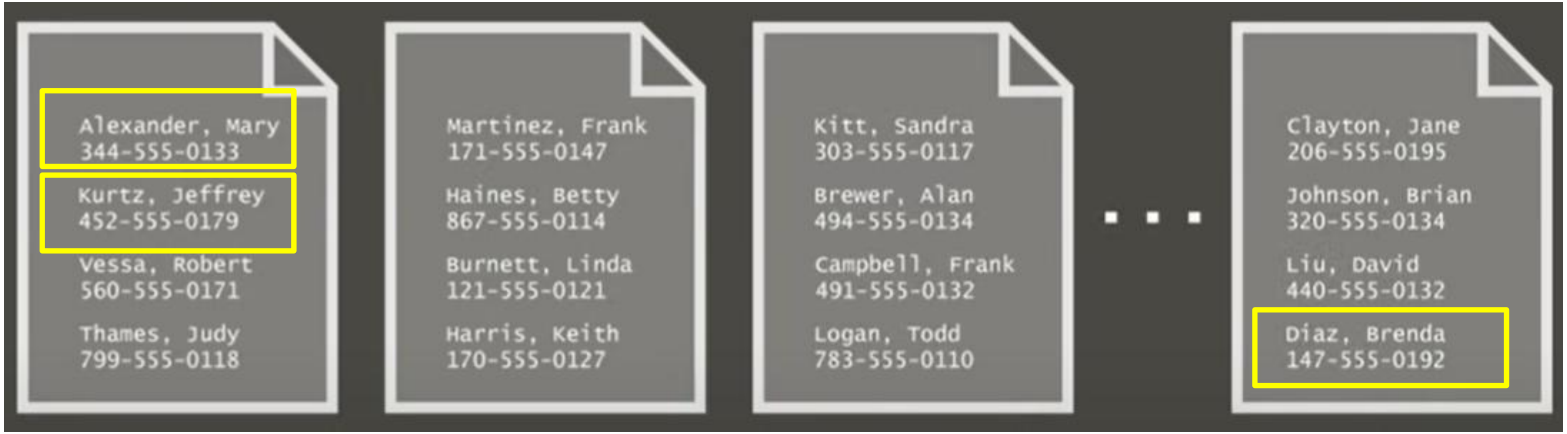
```
CREATE TABLE dbo.PhoneBook
(
   LastName varchar(50) NOT NULL,
   FirstName varchar(50) NOT NULL,
   PhoneNumber varchar(50) NOT NULL
);
```

It is insufficient!!!



Results:

783-555-0110



Alexander, Mary
344-555-0133

Kurtz, Jeffrey
452-555-0179

Vessa, Robert
560-555-0171

Thames, Judy
799-555-0118

Martinez, Frank
171-555-0147

Haines, Betty
867-555-0114

Burnett, Linda
121-555-0121

Harris, Keith
170-555-0127

Kitt, Sandra
303-555-0117

Brewer, Alan
494-555-0134

Campbell, Frank
491-555-0132

Logan, Todd
783-555-0110

. . .

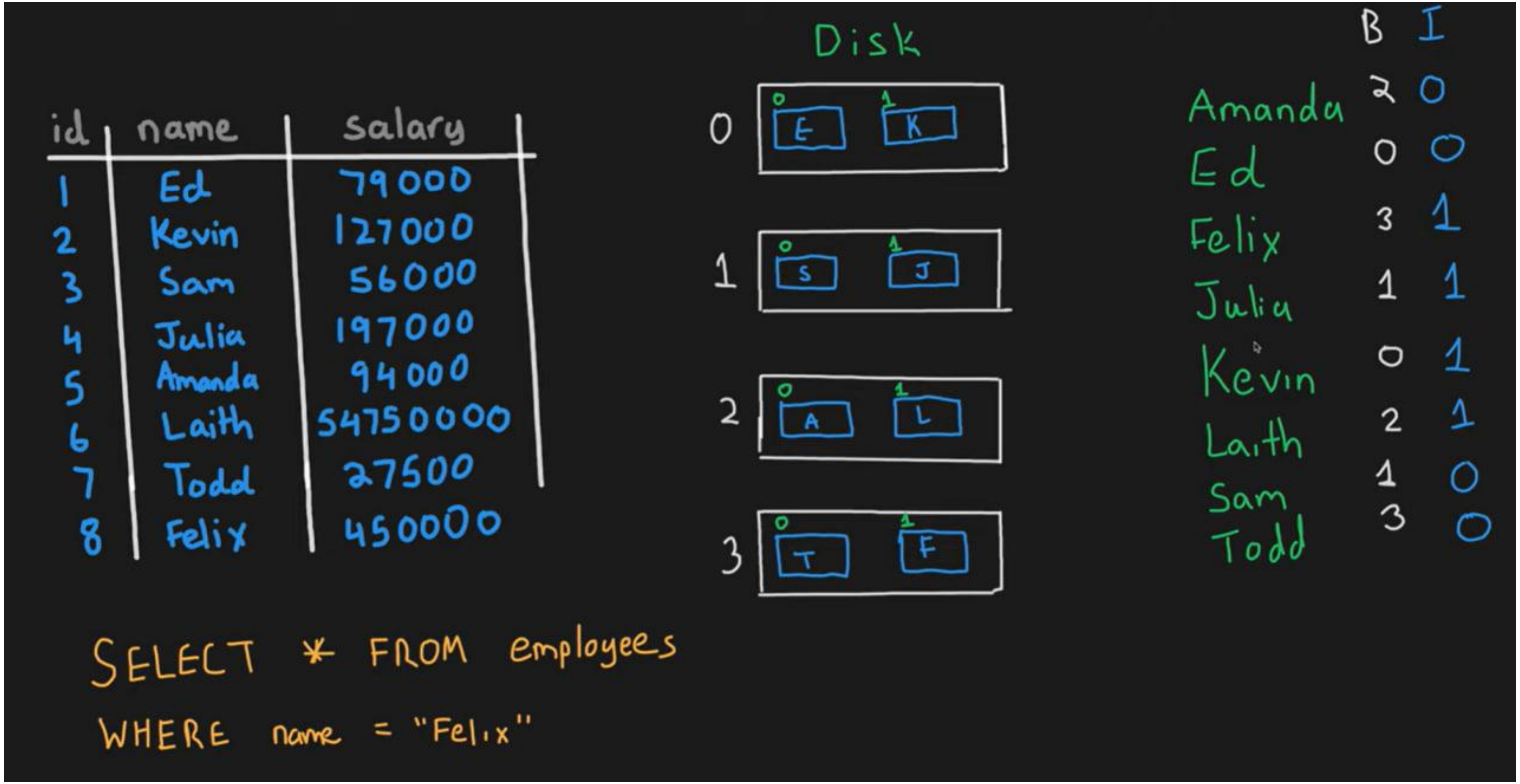
Clayton, Jane
206-555-0195

Johnson, Brian
320-555-0134

Liu, David
440-555-0132

Diaz, Brenda
147-555-0192

Introduction



id	name	salary
1	Ed	79000
2	Kevin	127000
3	Sam	56000
4	Julia	197000
5	Amanda	94000
6	Laith	54750000
7	Todd	27500
8	Felix	450000

	B	I
Amanda	2	0
Ed	0	0
Felix	3	1
Julia	1	1
Kevin	0	1
Laith	2	1
Sam	1	0
Todd	3	0

```
SELECT * FROM employees
WHERE name = "Felix"
```

Indexes

- An *index* on a file speeds up selections on the *search key fields* for the index.
  - Any subset of the fields of a relation can be the search key for an index on the relation (e.g., age or colour).
  - *Search key* is not the same as *key* (minimal set of fields that uniquely identify a record in a relation).
- An index contains a collection of *data entries*, and supports efficient retrieval of all data entries **k*** with a given key value **k**.

Indexes

- In Internal schema of Three-Schema Architecture!

- An index for an attribute (or attributes) of a relation is a data structure used to speed access to tuples of a relation, given values of the attribute(s).
- In a DBMS it is a balanced search tree with giant nodes (a full disk page) called a B-tree.
- Can make query answering and joins involving the attribute much faster.
- On the other hand, modifications are more complex and take longer.

Declaring Indexes

- No standard!
- Typical syntax:

```
CREATE INDEX foodInd ON foods(nationality);
CREATE INDEX SellInd ON Sells(resturant, food);
```

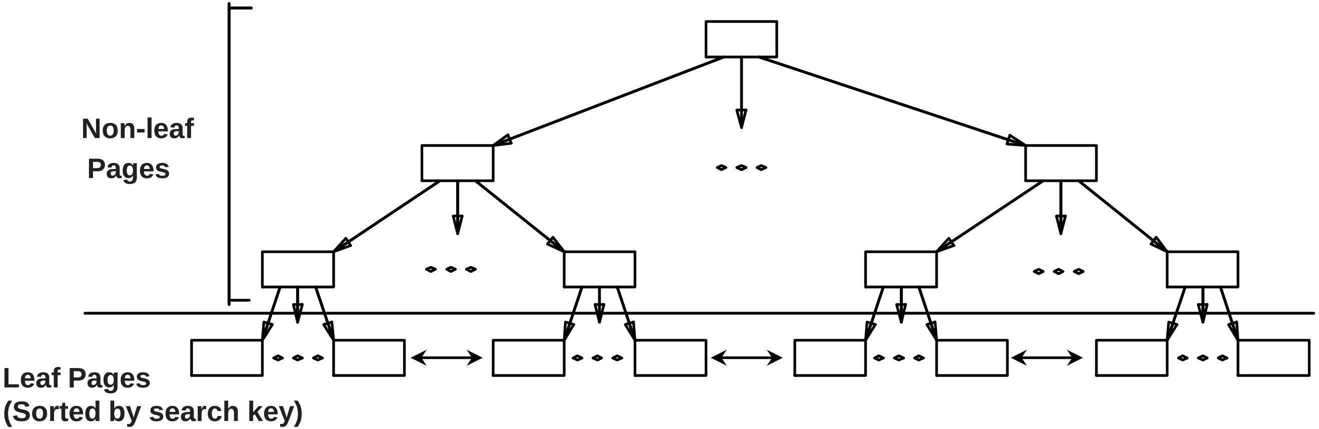
Using Indexes

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use foodInd and SellInd to find the prices of foods which nationality is Iranian and sold by Joe. (next slide)
- With the indices, just retrieve tuples satisfying these conditions
  - Clearly, can result in huge savings (vs. retrieving all tuples from the mentioned relations)

```
SELECT price
FROM foods, Sells
WHERE nationality = ‘Iranian’ AND
    foods.name = Sells.food AND
    resturant = ‘Joe’’s resturant’;
```

1. Use foodInd to get all the foods which Iranian nationality.
2. Then use SellInd to get prices of those foods, with resturant = ’Joe’’s resturant’

E.g., Tree index



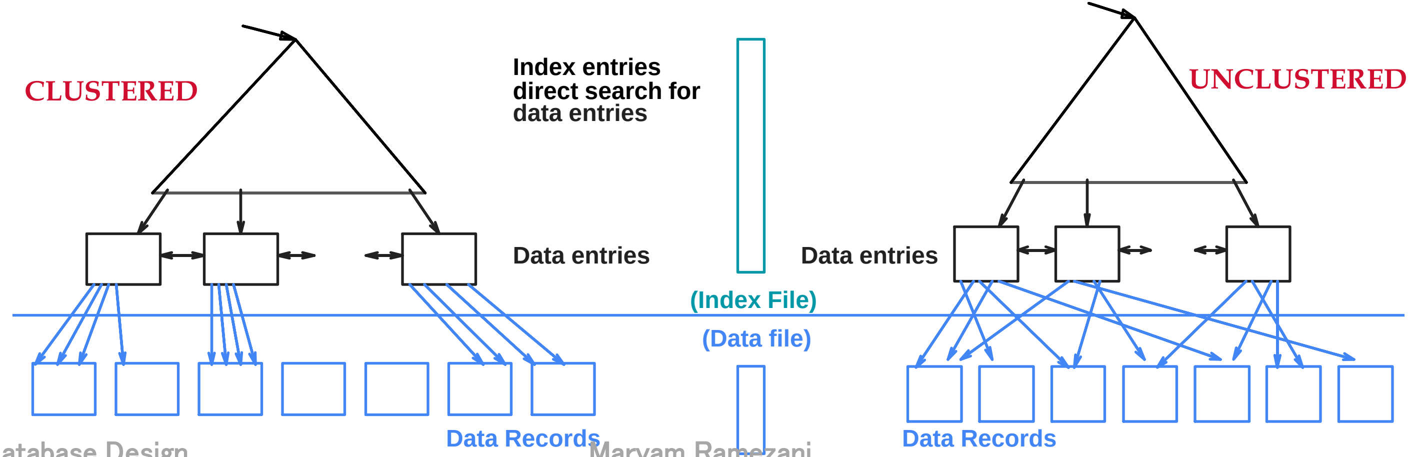
- Leaf pages contain *data entries*
- Non-leaf pages have *index entries*; used only to direct searches:

Alternatives for Data Entry k* in Index

- Three alternatives:
  - Data record with key value **k**
  - ⟨**k**, rid of data record with search key value **k**⟩
  - ⟨**k**, list of rids of data records with search key **k**⟩
- Alternative 3 more compact than Alternative 2, but leads to variable sized data entries even if search keys are of fixed length.
- Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value k
  - Examples of indexing techniques: B+ tree, hash based structures
  - Typically, index contains auxiliary information that directs searches to the desired data entries
- Clustered vs. unclustered: If order of data records is the same as, or \`close to’, order of data entries, then called clustered index.

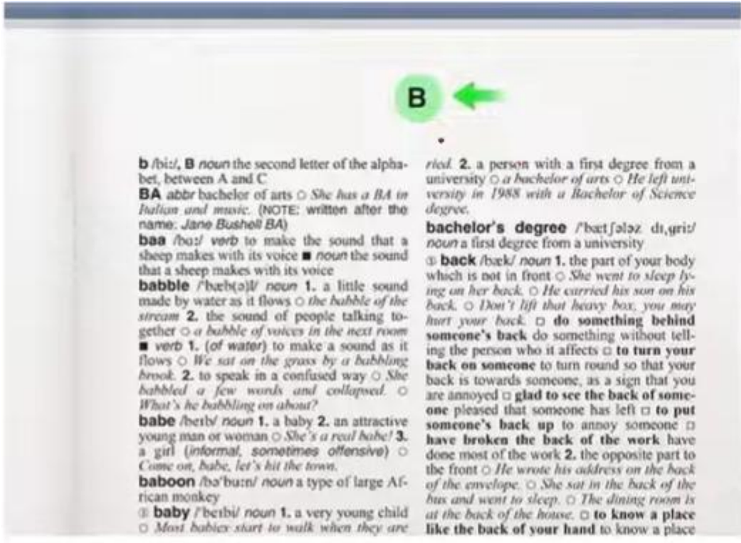
Clustered vs. Unclustered Index

- Suppose that Alternative (2) is used for data entries, and that the data records are stored in a Heap file.
  - To build clustered index, first sort the Heap file (with some free space on each page for future inserts).
  - Overflow pages may be needed for inserts. (Thus, order of data recs is `close to', but not identical to, the sort order.)



Clustered Index

- ❑ A cluster index defined the order in which data is physically stored in a table.
    - ○ For example Dictionary.
- ❑ A table can only have one cluster index.
- ❑ If you configure a PRIMARY KEY, Database Engine automatically creates a clustered index, unless a clustered index already exists.



B

b /biː/, **B** *noun* the second letter of the alphabet, between A and C

BA *abbr* bachelor of arts ○ *She has a BA in Italian and music.* (NOTE: written after the name: *Jane Bushell BA*)

baa /bɑː/ *verb* to make the sound that a sheep makes with its voice ■ *noun* the sound that a sheep makes with its voice

babble /ˈbæb(ə)l/ *noun* **1.** a little sound made by water as it flows ○ *the babble of the stream* **2.** the sound of people talking together ○ *a babble of voices in the next room* ■ *verb* **1.** (*of water*) to make a sound as it flows ○ *We sat on the grass by a babbling brook.* **2.** to speak in a confused way ○ *She babbled a few words and collapsed.* ○ *What's he babbling on about?*

babe /beɪb/ *noun* **1.** a baby **2.** an attractive young man or woman ○ *She's a real babe!* **3.** a girl (*informal, sometimes offensive*) ○ *Come on, babe, let's hit the town.*

baboon /bəˈbuːn/ *noun* a type of large African monkey

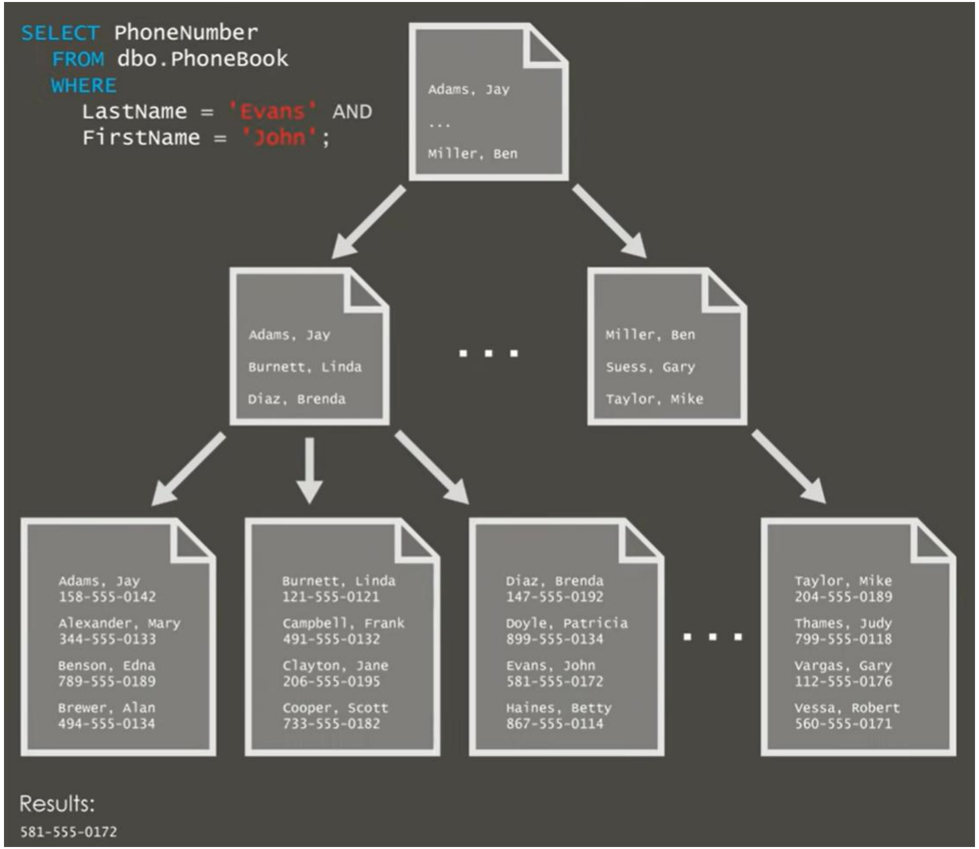
baby /ˈbeɪbi/ *noun* **1.** a very young child ○ *Most babies start to walk when they are*

ried. **2.** a person with a first degree from a university ○ *a bachelor of arts* ○ *He left university in 1988 with a Bachelor of Science degree.*

bachelor's degree /ˈbætʃələz dɪˌgriː/ *noun* a first degree from a university

back /bæk/ *noun* **1.** the part of your body which is not in front ○ *She went to sleep lying on her back.* ○ *He carried his son on his back.* ○ *Don't lift that heavy box, you may hurt your back.* □ **do something behind someone's back** do something without telling the person who it affects □ **to turn your back on someone** to turn round so that your back is towards someone, as a sign that you are annoyed □ **glad to see the back of someone** pleased that someone has left □ **to put someone's back up** to annoy someone □ **have broken the back of the work** have done most of the work **2.** the opposite part to the front ○ *He wrote his address on the back of the envelope.* ○ *She sat in the back of the bus and went to sleep.* ○ *The dining room is at the back of the house.* □ **to know a place like the back of your hand** to know a place

Clustered Index

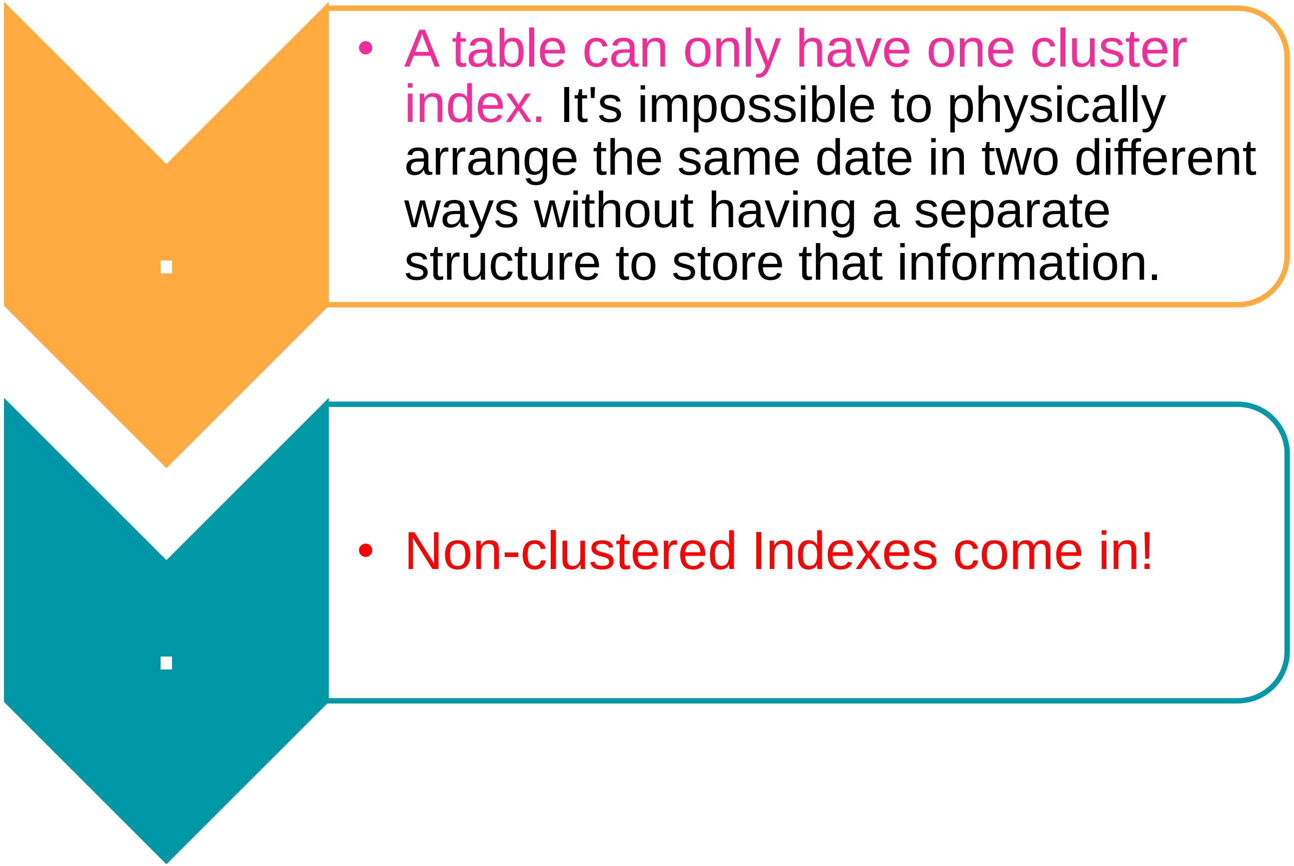


```
SELECT PhoneNumber
  FROM dbo.PhoneBook
  WHERE
    LastName = 'Evans' AND
    FirstName = 'John';
```

Results:

581-555-0172

Clustered Index



- A table can only have one cluster index. It's impossible to physically arrange the same date in two different ways without having a separate structure to store that information.

- Non-clustered Indexes come in!

Non-Clustered Index

- ❑ A non-clustered index is stored at one place and table data is stored in another place. For example Book Index.
- ❑ Instead of having base table at the leaf of tree, we have a set of pointers or references back to the base data.
- ❑ A table can have multiple non-clustered index.
- ❑ Non-clustered index is slower than clustered index.
- ❑ If the index is non-unique, a uniquified value is adds internally to make it unique, and it carries through into reference values. RIDs are always unique.

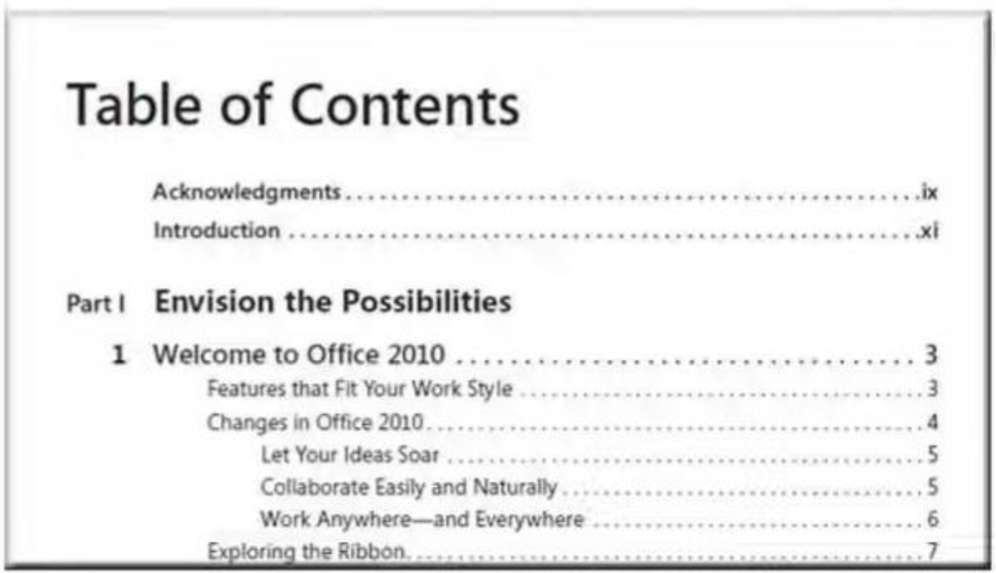


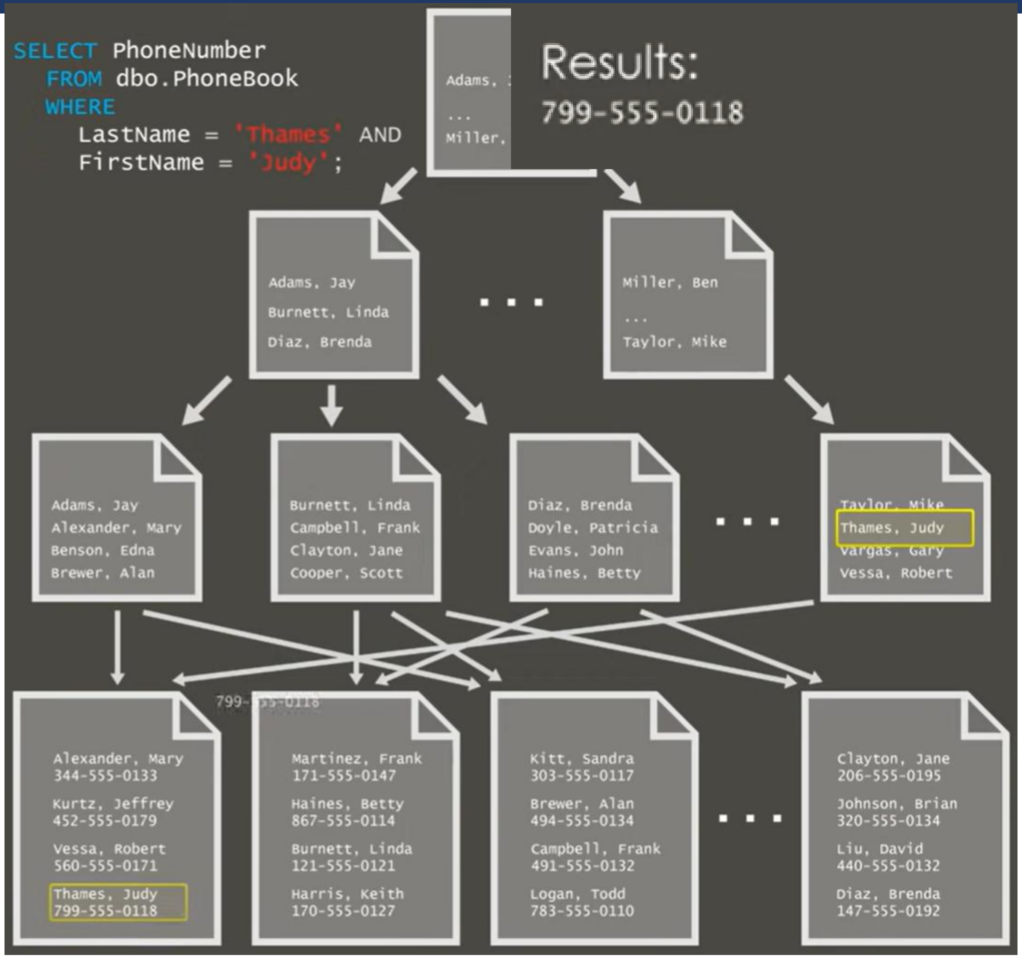
Table of Contents

Acknowledgments ix
Introduction xi

Part I Envision the Possibilities

1 Welcome to Office 2010 3
Features that Fit Your Work Style 3
Changes in Office 2010 4
Let Your Ideas Soar 5
Collaborate Easily and Naturally 5
Work Anywhere—and Everywhere 6
Exploring the Ribbon 7

Non-Clustered Index



```
SELECT PhoneNumber
  FROM dbo.PhoneBook
  WHERE
    LastName = 'Thames' AND
    FirstName = 'Judy';
```

Results:

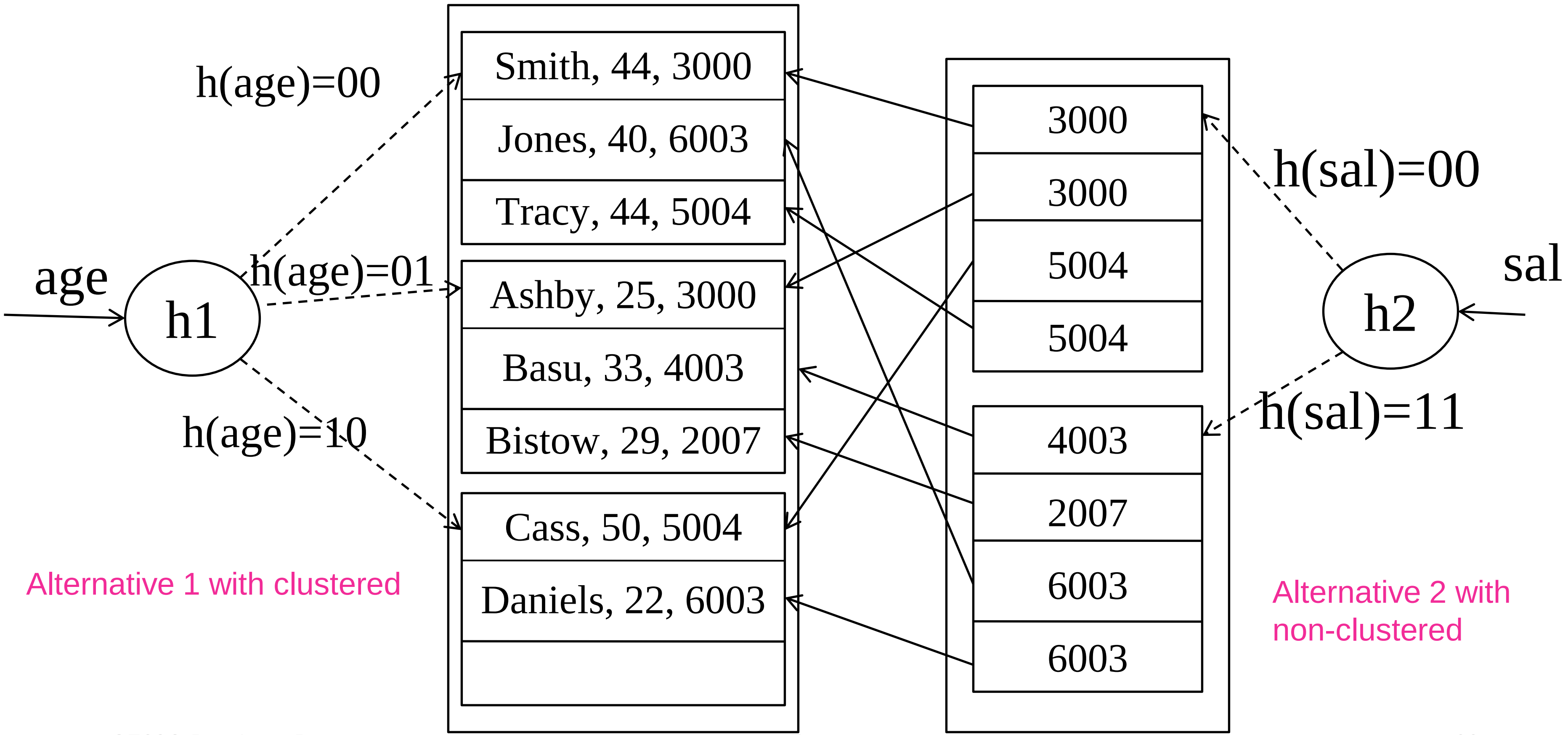
799-555-0118

RID=Row Identifier= physical location of the rows in the table.

Hash-Based Indexes

- Good for equality selections.
    - Index is a collection of *buckets*. Bucket = *primary* page plus zero or more *overflow* pages.
    - *Hashing function* **h**: **h**(*r*) = bucket in which record *r* belongs. **h** looks at the *search key* fields of *r.*
- If Alternative (1) is used, the buckets contain the data records; otherwise, they contain <key, rid> or <key, rid-list> pairs.

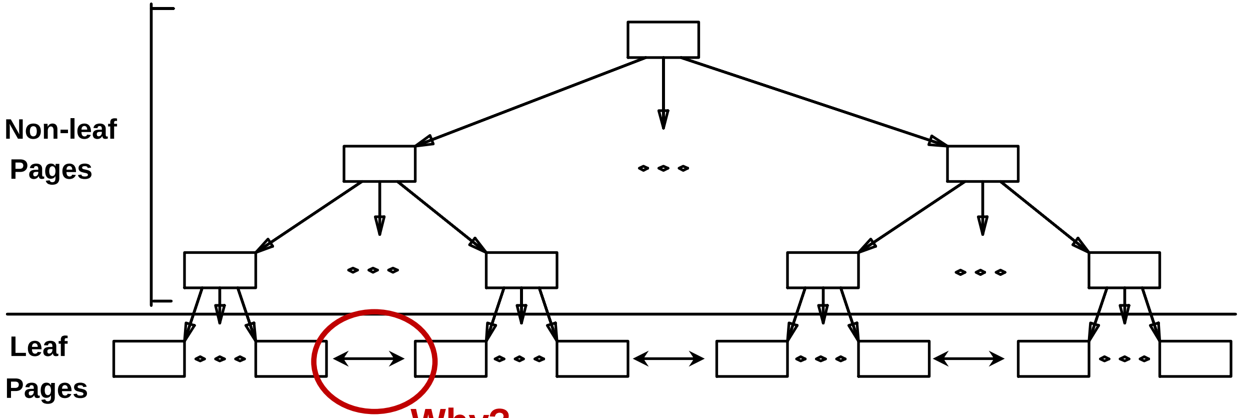
Index-organized file hashed on age, with auxiliary index on sal



Alternative 1 with clustered

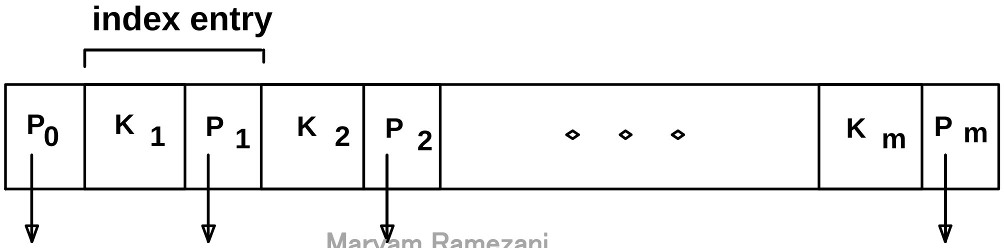
Alternative 2 with non-clustered

B+ Tree Indexes



Why?

- Leaf pages contain *data entries*, and are chained (prev & next)
- Non-leaf pages contain *index entries*; they direct searches:

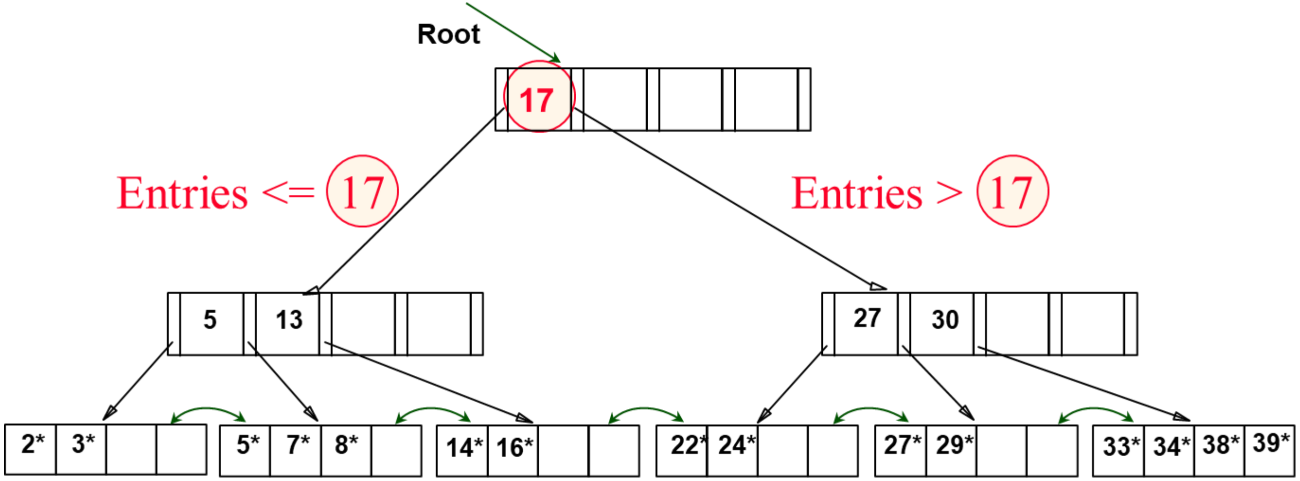


B+ Tree Indexes

- ❑ Faster than binary search.
- ❑ Lots of pointer while the height o tree is at most 3 or 4!
- ❑ Pages at leaves are linked for interval search!
- ❑ Example
  - ○ Number of pointers: 100 with height:4 will be 100^4 leaves.
  - ○ Order of tree is 4 but binary search is log(10^4)

Example B+ Tree

- ❑ Find 28*?
- ❑ Find 29*?
- ❑ Find All > 17* and < 30*
- ❑ Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - ○ And change sometimes bubbles up the tree



Lets test on Postgres

- explain analyze select * from athlete a where sport_id=1

- explain analyze select athlete_id from athlete a where athlete_id =15

- explain analyze select * from athlete a where athlete_id =15
-

- explain analyze select * from athlete a where a.athlete_name ='browntoni’

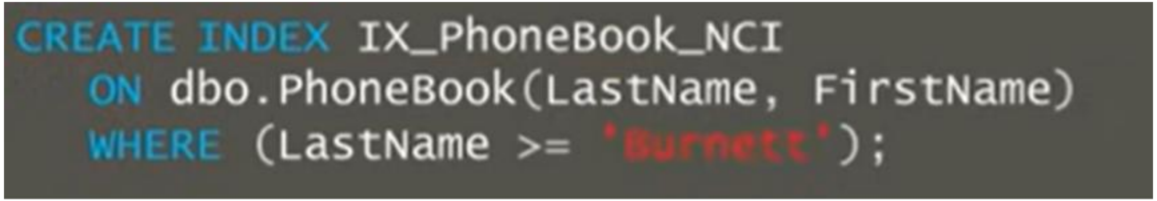
- explain analyze select * from athlete a where a.athlete_name like ‘%b%'

Using Both Clustered & Non-Clustered

- Since a non-clustered index is separate from the base data, the base data could exist instead as clustered index. So the references in leaf of non-clustered index are not RID, but instead are the clustered index key values.

Filtered Indexes

- Filtered indexes only contain rows that meet a user-defined predicate, by adding WHERE clause to the index definition. (In Postgres its name **Partial Index**)



```
CREATE INDEX IX_PhoneBook_NCI
    ON dbo.PhoneBook(LastName, FirstName)
    WHERE (LastName >= 'Burnett');
```

- A clustered index can’t be filtered because it has to contain all the data in the table.

Database Tuning

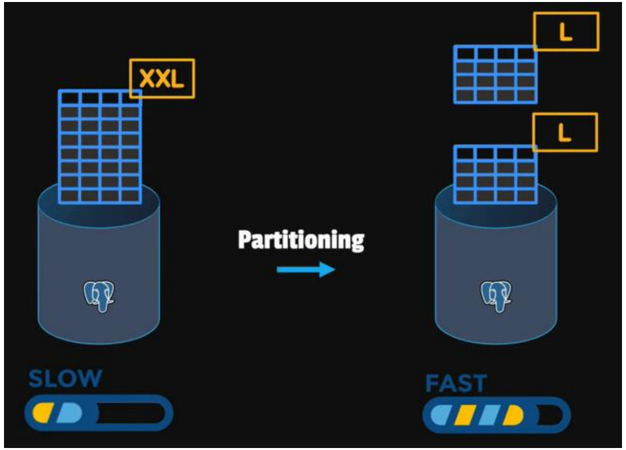
- A major problem in making a database run fast is deciding which indexes to create.
- Recall:
  - Pro: An index speeds up queries that can use it.
  - Con: An index slows down modifications on its relation because the index must be modified too.
- The key for a relation is usually the most useful attribute to have an index on:
  - Queries in which a value for a key is specified are common.
  - For a given key value there is only one tuple. Thus the index returns at most one tuple, requiring just 1 page from the relation instance to be retrieved.

Partitioning

Partitioning

- ❑ When the table size grows over time, each operation cost on the table will increase as well.
- ❑ We can't increase the size of the table over 32GB in normal conditions. Before reaching this size performance issues may arise.

Good Solution: Partitioning

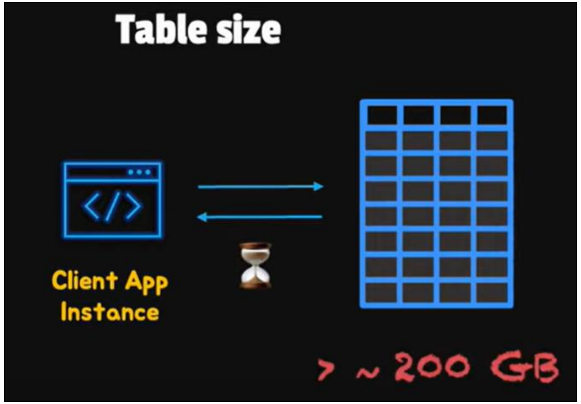


Add partitioning for a table?

- It shouldn’t be the first option to improve performance!!! Why?
  - It adds another level of complexity!!
  - Unlike other performance enhancing such as indexing, partitions are part of table definition so its difficult to change!!

Add partitioning for a table?

- Signs to check a table needs partitioning:
  1) Table Size: there is no rule! But encounter long responses time and table is larger than 200GB

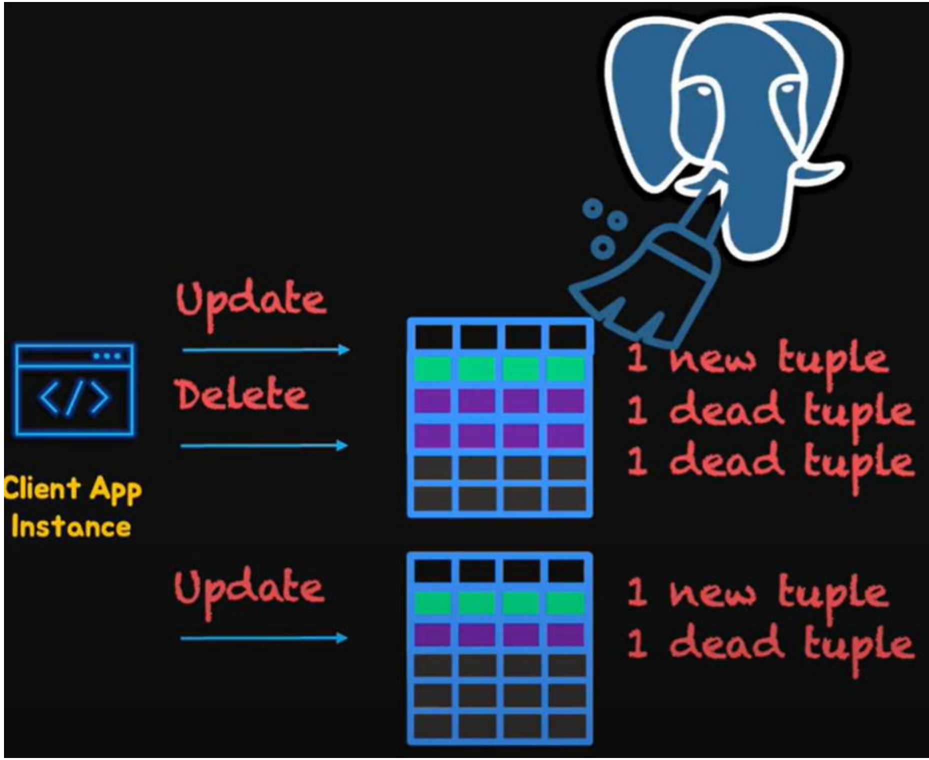


Add partitioning for a table?

2) Table Bloat: For a DELETE, it simply marks the row as unavailable for future transactions, and for UPDATE, under the hood it’s a combined INSERT then DELETE, where the previous version of the row is marked unavailable.

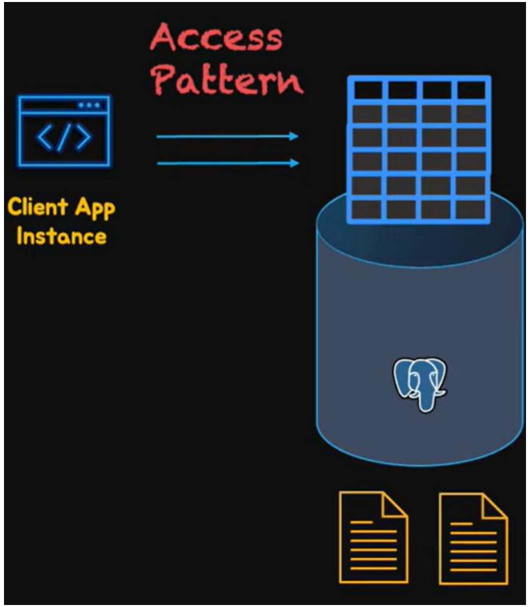
The space cannot be used. To then mark the space as available for use by the database, a vacuum process (manually or automatically) needs to come along behind the operations, and mark that space available for the database to use.

Vacuum process should scan all rows. If table is large vacuum process will take longer. Partitioning can help to make it faster with less CPU.

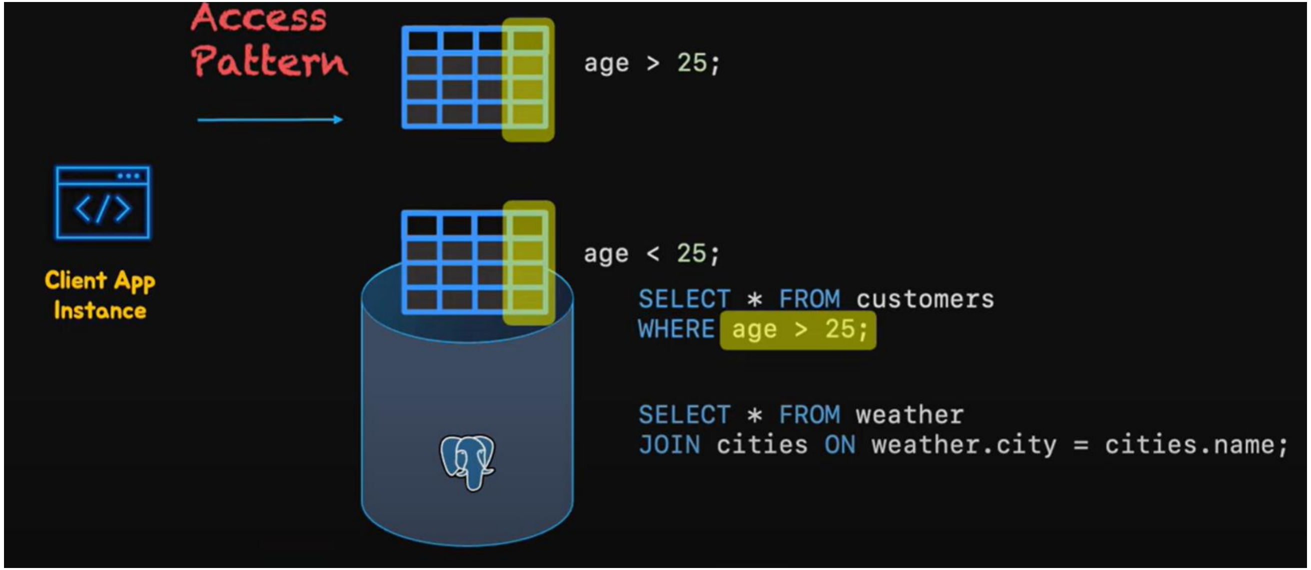


How should the Tables be partitioned?

- ❑ Partitioning can drastically improve performance on a table when done right, but when not needed o done wrong can make the performance worse or it can make the database unstable.
- ❑ First look for access patterns for splitting the tables:
  - ○ By knowing the applications that access the database.
  - ○ Monitoring the logs and generating reports.



How should the Tables be partitioned?



```
SELECT * FROM customers
WHERE age > 25;

SELECT * FROM weather
JOIN cities ON weather.city = cities.name;
```

We look for columns that are either in **where** or in **join** conditions.

These will be the partition keys.

In a good design, we have a small subset of data rather than the whole

Partitioning Methods

Range Partition

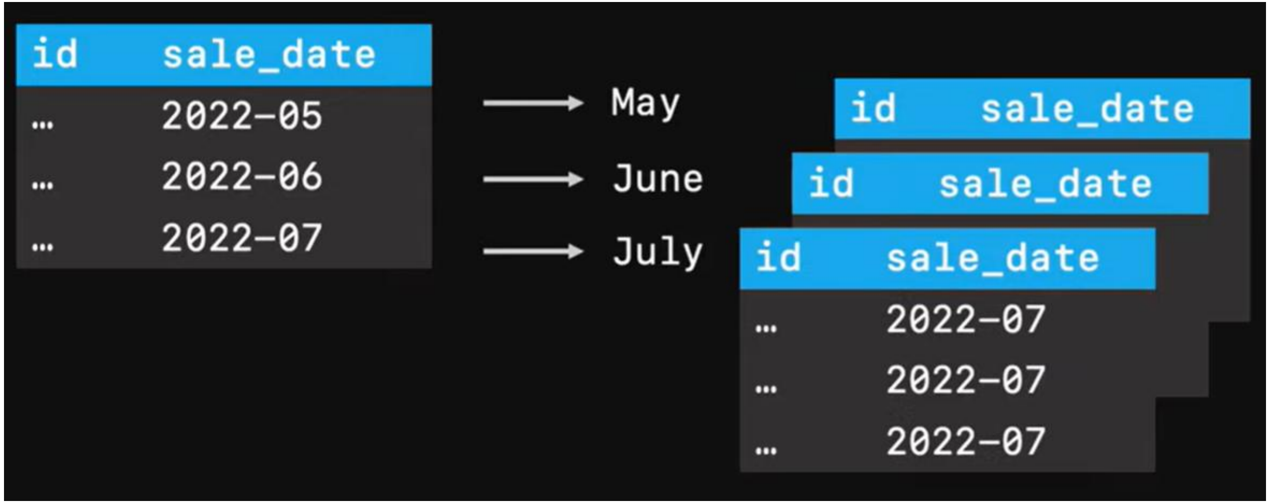
List Partition

Hash Partition

Composite Partition

Partitioning Methods

- Range partitioning maps data to partitions on the basis of ranges of partition key values for each partition.



id	sale_date
…	2022-05
…	2022-06
…	2022-07

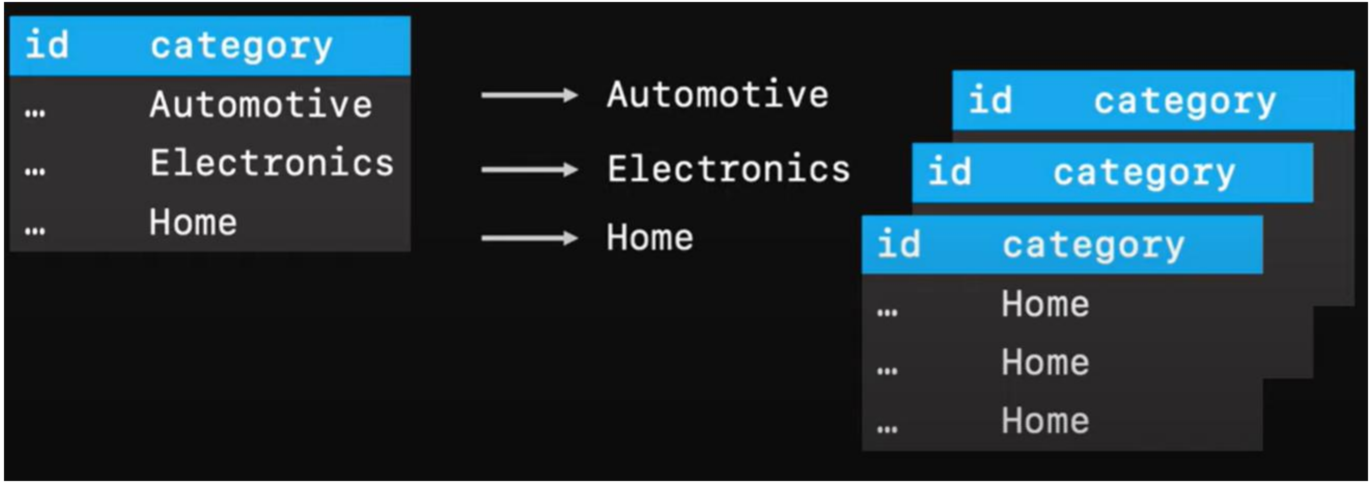
id	sale_date

id	sale_date

id	sale_date
…	2022-07
…	2022-07
…	2022-07

Partitioning Methods

- List partitioning maps rows to partitions by using a list of discrete values for the partitioning column.
  - Good when partition key is category value.



id	category
...	Automotive
...	Electronics
...	Home

→ Automotive

→ Electronics

→ Home

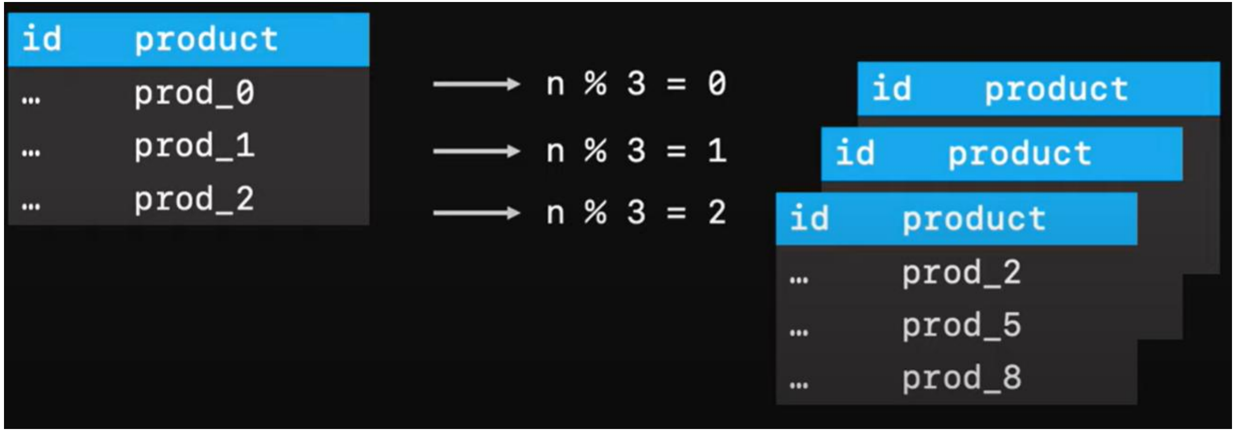
id	category

id	category

id	category
...	Home
...	Home
...	Home

Partitioning Methods

- Hash partitioning maps data to partitions by using a hashing algorithm applied to a partitioning key.
  - Especially useful when there is no obvious way of diving data into logical groups.



id	product
…	prod_0
…	prod_1
…	prod_2

→ n % 3 = 0

→ n % 3 = 1

→ n % 3 = 2

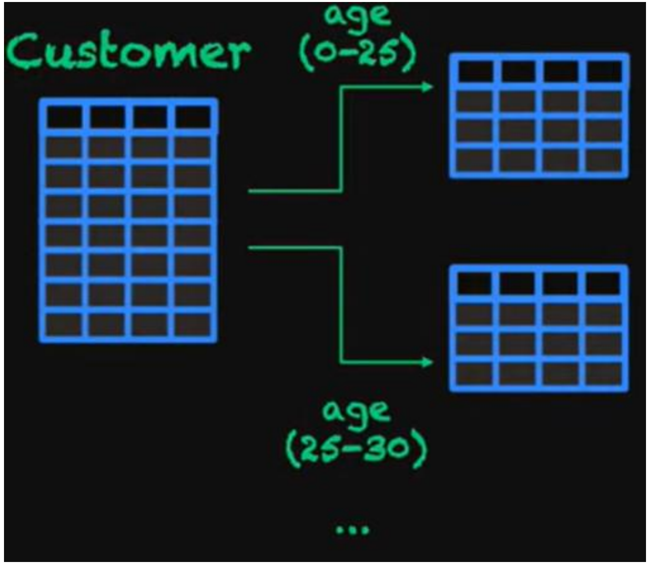
id	product
…	prod_2
…	prod_5
…	prod_8

Partitioning Methods

- Composite partitioning:
  - Range-Hash sub partitions the range partitions using a hashing algorithm.
  - Range-List sub partitions the range partitions using an explicit list.

Range Partition – Example

- Consider following table with not null age attribute:



Range Partition– Example

```
create table customers (id integer, name text, age numeric) partition by range(age)
create table cust_young partition of customers for values from (MINVALUE) to (25)
create table cust_medium partition of customers for values from (25) to (75)
create table cust_old partition of customers for values from (75) to (MAXVALUE)
insert into customers values (1,'Bob',20), (2,'Alice',20),(3,'Doe',38),(4,'Richard',80)
select * from customers c
select tableoid::regclass,* from customers c
```