

Digital Career Institute

Databases - Basic Performance



Goal of the Submodule

The goal of this submodule is to help the learners understand how to make a database perform better. By the end of this submodule, the learners will be able to:

- Identify the potential database performance bottlenecks.
- Understand how to increase the database throughput using hardware and software strategies.
- Understand how to reduce the database workload using more efficient SQL queries.
- Use SQL to analyze the performance of specific query statements.

Topics

- Database throughput and workload.
- Optimize throughput using hardware:
 - Hard disk and memory.
 - Database clustering.
- Optimize throughput using software:
 - Sequential vs. indexed scanning.
 - PostgreSQL default index creation.
 - Defining custom indexes.
- Optimize workload:
 - Explaining queries.
 - Populating tables.
 - Subqueries.
 - Application optimizations.

Database Performance

How good a database performs depends on two main factors:

Throughput

Offer

The **throughput** is the amount of operations per unit of time that the system can *offer*.

Workload

Demand

The **workload** is the amount of operations per unit of time that the applications may *demand*.

Database Throughput

Database Throughput

It is the overall capability of the hardware and software to process data.

Hardware

A better hardware will often translate into a better throughput.

Software

The throughput can also be optimized using software strategies.

The most relevant hardware parts for the database are:

Hard Drive

Data is stored on the file system in the hard drive. A faster hard drive will produce faster results.

Memory

Some data is often also stored in memory. A faster memory module will produce faster results.

Multiple hardware can be used to increase the throughput.

Clusters

A database cluster is a set of database instances that work together to provide a single point of entry solution.

Database clusters usually act as routers to deliver each request to the instance with the lowest demand at each moment. This is called **load balancing**.

Optimizations may be made on two main areas:

Accessing Data

Accessing the data takes time, more so if the data is in the file system. RDBMS store some data in memory to improve the throughput.

Searching Data

If the data is stored with a more specific structure, searching may require less operations and less reading may be required.

Accessing Data: Hard Drive vs. RAM

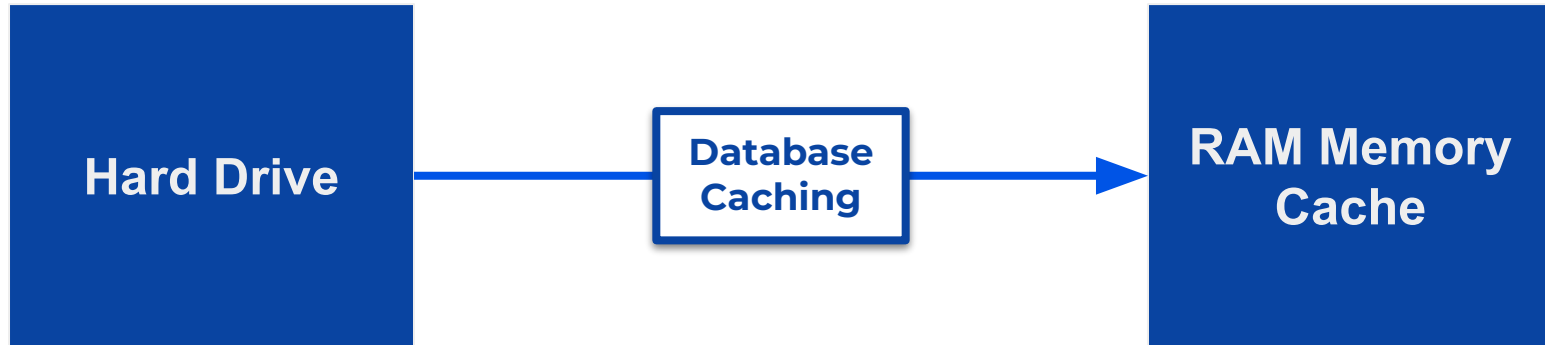
Accessing the hard drive consumes more time and resources than accessing the memory.

	Hard Drive	RAM
Speed	750MB/sec.	12.800MB/sec.

The durable ACID property requires the data to be stored on the hard drive. Every time we read, update or create contents on the database, the hard drive needs to be accessed. But some operations can be used only in-memory and some data can be stored in memory.

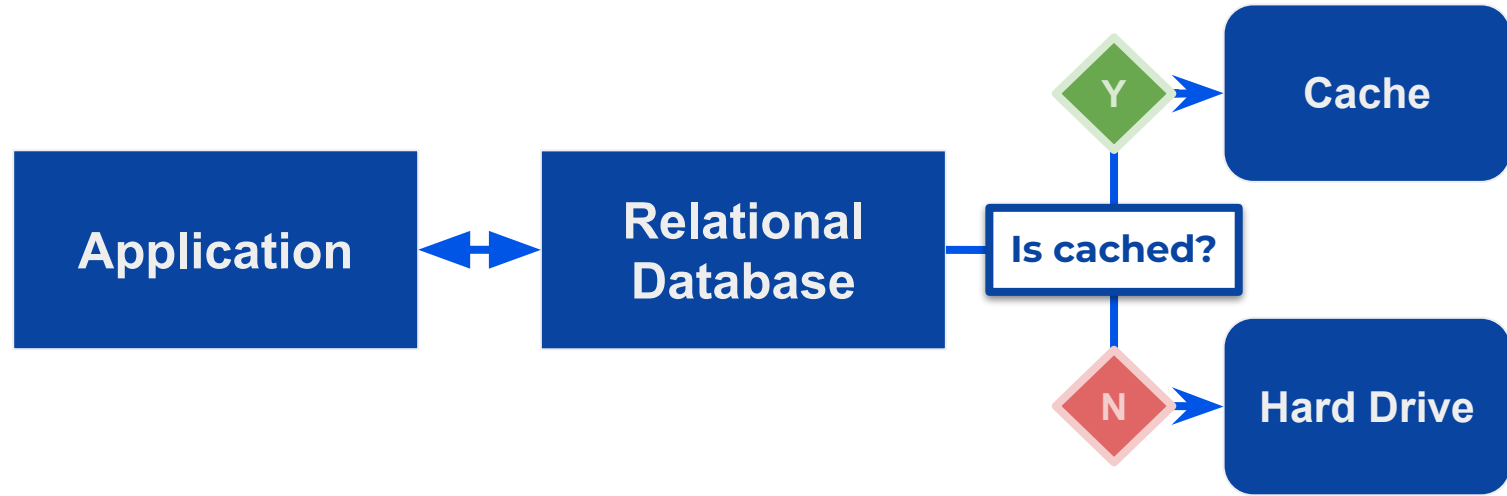
Accessing Data: Caching

The process of storing a copy of the most common data to improve access times is called **caching**.



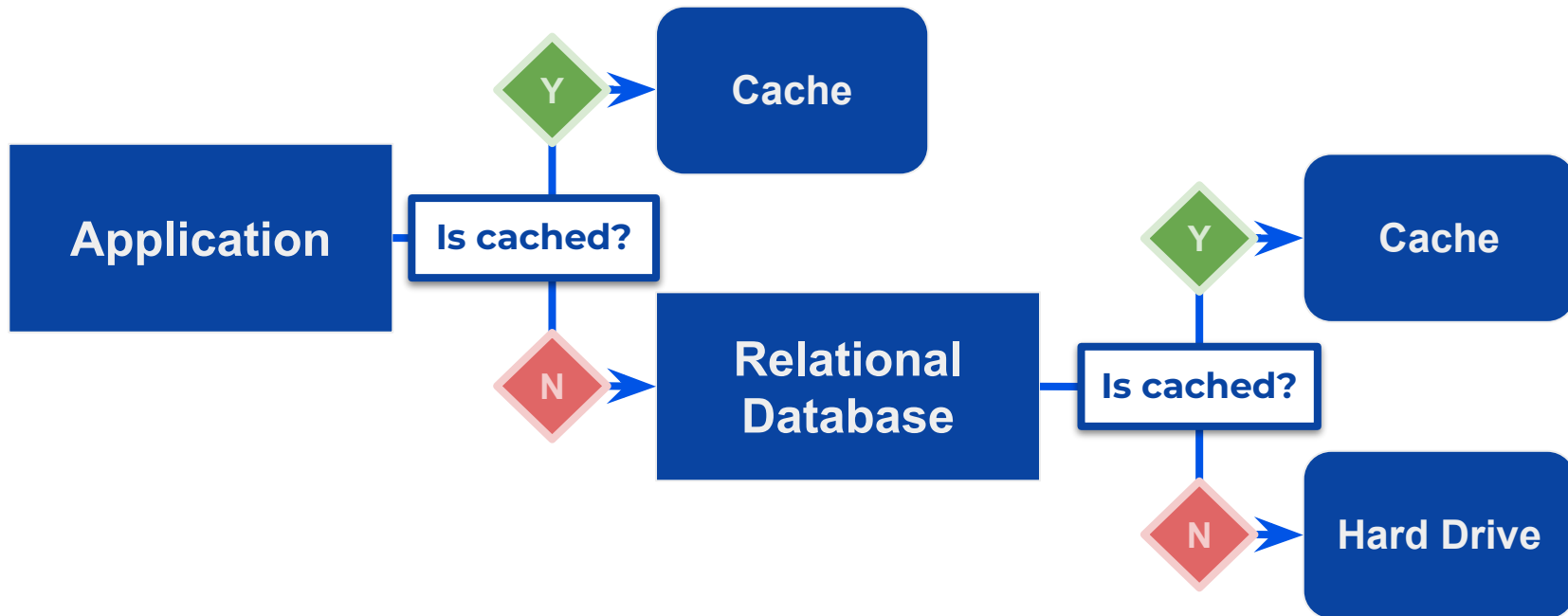
Accessing Data: Caching

If the required content has been cached, the RDBMS will access the in-memory copy. If not, it will access the hard disk.



Accessing Data: Caching

Caching may happen at different levels of the software stack.



Test 1.

Count the number of passes in the white team.

In the next slide, you will see a video of a white team and a black team of 3 players each, all mixed up and moving around.

Each team has a ball and passes the ball only between them.

Searching Data





Had you seen this picture before the video, you would probably have been able to notice the gorilla and, at the same time, count the correct number of passes.

Because nobody told you there would be a gorilla, you may have missed it. But you probably counted the passes right, because you were warned in advance.

An RBDMS also performs better when it knows what it will be expected to do.

Test 2.

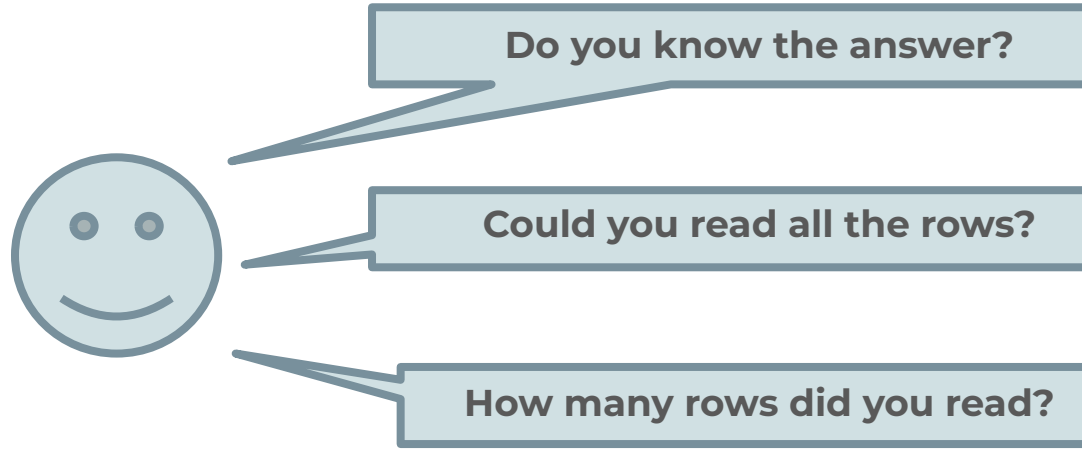
Count the number of gorillas in the list.

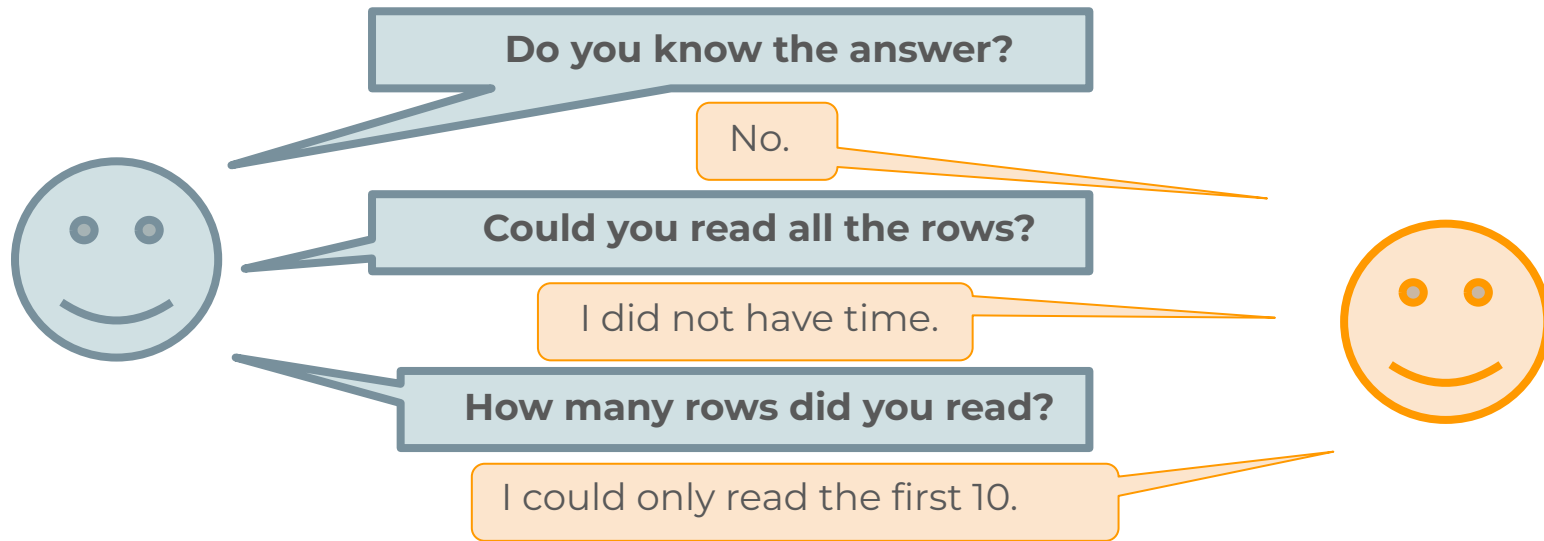
On the next slide, you will see a list of 17 players of three teams: White, Black and Gorilla.

The slide should only be visible for 2 seconds.

Searching Data

Name	Team
Allison	White
Alicia	Gorilla
Daniela	White
Charles	Gorilla
Johan	Black
Patrick	White
Julia	Gorilla
Carmen	Black
Ellen	White
George	Gorilla
Gemma	Black
Elliot	White
Sean	White
Samantha	Black
Michael	White
Jennifer	Black
Martha	White





Test 3.

Count the number of gorillas in the list.

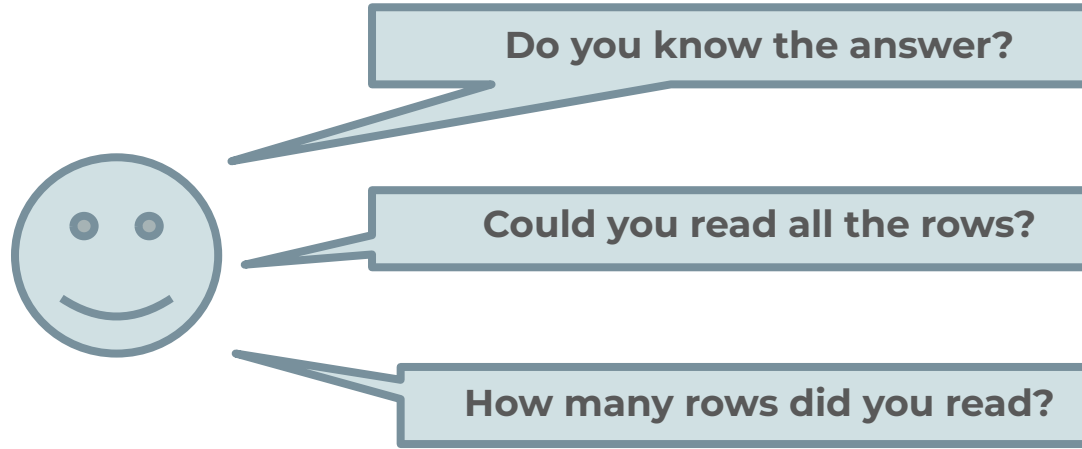
You will see the same list.

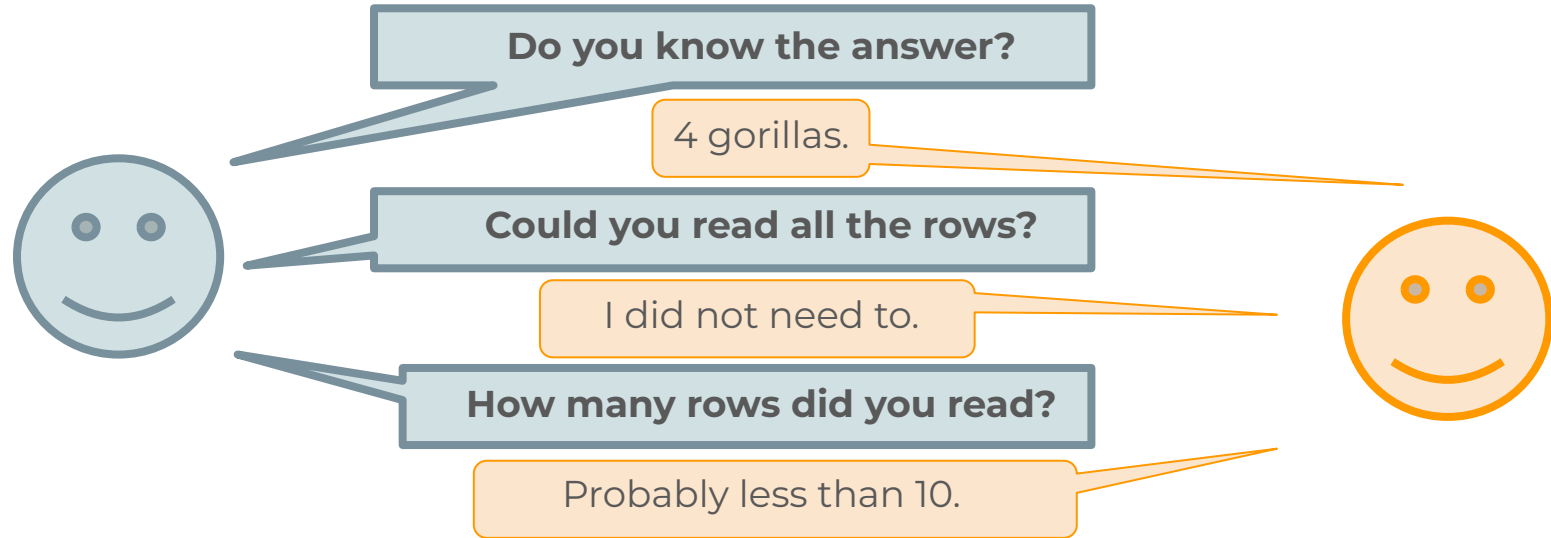
This time, the list will be sorted by team name.

The slide should only be visible for 2 seconds.

Searching Data

Name	Team
Johan	Black
Carmen	Black
Gemma	Black
Samantha	Black
Jennifer	Black
Alicia	Gorilla
Charles	Gorilla
Julia	Gorilla
George	Gorilla
Allison	White
Daniela	White
Patrick	White
Ellen	White
Elliot	White
Sean	White
Michael	White
Martha	White





Test Conclusions.

1. Knowing the kind of operations that will need to be done most often, will help reduce the searching time.
2. Preparing the data in a way that does not require reading all rows, will help reduce the searching time.

Searching Data

Finding data may take different times depending on how the data is organized.



Photo by [Oleksii Hlembotskyi](#) on [Unsplash](#)

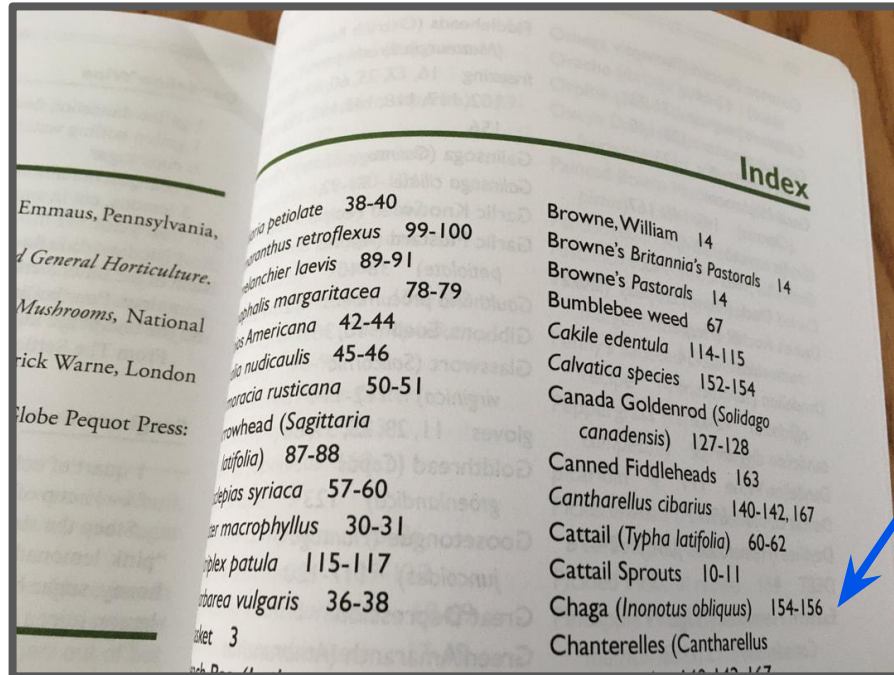


Photo by [CHUTERSNAP](#) on [Unsplash](#)

Indexes

An **index** is a reference to the actual content.
It helps us search for specific information.

Instead of searching
the entire book/table
we **search the index**.



The index will
prevent us from
having to read 154
pages to find
information about
Chaga.

Before computers, libraries used a system based on **indexed cards** to locate books.

An **index** is a list of references to the storage location of each element.



Photo by [Maksym Kaharlytskyi](#) on [Unsplash](#)

The index is sorted according to a specific field (year, author, title, ...).

Each index needs to be kept up to date to make the system useful.

Sequential vs. Indexed Scanning

Searching through all the rows is called **table scanning** or **sequential scanning**.

Name	Team
Allison	White
Alicia	Gorilla
Daniela	White
Charles	Gorilla
Johan	Black
Patrick	White
...	...

Searching through an indexed version of the data is called **indexed scanning**.

Name	Team
Johan	Black
Carmen	Black
Gemma	Black
Samantha	Black
Jennifer	Black
Alicia	Gorilla
...	...

If the search field is not indexed, the RDBMS can only do a table scan.
To be able to do an index scan, the field must have an index.

Database Indexes

A database **index** is a data structure used to improve the performance of search operations on specific fields.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
5	Johan	Schmidt	Black	...
6	Patrick	Peterson	White	...
...

Index table	
Team	Id
Black	5
Black	8
Black	11
Black	14
Black	16
Gorilla	2
...	...

- It is a different data structure that points to the data.
- An index works always on a specific field or combination of fields.

Brief Anatomy of Indexes

An **index** usually has two data: the searching field and a pointer to the actual data in storage.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
5	Johan	Schmidt	Black	...
6	Patrick	Peterson	White	...
...

Index table	
Team	Pointer
Black	CC470F...
Black	611189...
Black	E6894B...
Black	6079F9...
Black	FF18F1...
Gorilla	CC6654...
...	...

- Each row has an address in the RDBMS storage and the index table points to it.
- In some cases, indexes may instead hold the entire data.

Primary indexes are defined on columns that guarantee there is one single record with the same value.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
5	Johan	Schmidt	Black	...
6	Patrick	Peterson	White	...
...

Index table	
Id	Data block
1	CC470F...
2	611189...
3	E6894B...
4	6079F9...
5	FF18F1...
6	CC6654...
...	...

The index table contains the indexed field and a pointer to the data record.

The index table is sorted by the indexed field.

Secondary indexes are defined on columns that may have repeated values.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
5	Johan	Schmidt	Black	...
6	Patrick	Peterson	White	...
...

Index table	
Team	Data block
Black	CC470F...
Black	611189...
Black	E6894B...
Black	6079F9...
Black	FF18F1...
Gorilla	CC6654...
...	...

Create Indexes

The best way to create a **primary index** is to create a UNIQUE constraint.

```
CREATE TABLE people (  
    id            integer PRIMARY KEY,  
    first_name    varchar(20),  
    last_name     varchar(50),  
    social_sec    varchar(100) UNIQUE  
);
```

PostgreSQL automatically creates a primary index when a **UNIQUE** constraint is created.

Defining a **PRIMARY KEY** automatically creates a **UNIQUE** constraint.

Create Indexes

```
personal=# \d people
```

Table "public.people"				
Column	Type	Collation	Nullable	Default
id	integer		not null	
first_name	character varying(20)			
last_name	character varying(50)			
social_sec	character varying(100)			

```
Indexes:
```

```
    "people_pkey" PRIMARY KEY, btree (id)
```

```
    "people_social_sec_key" UNIQUE CONSTRAINT, btree (social_sec)
```

Two indexes were created with the SQL of the previous slide.

Create Indexes

```
CREATE INDEX <index_name>  
ON <table_name> (first_name);
```

An **index** can be manually created with SQL.

```
personal=# CREATE INDEX my_own_index ON people(last_name);  
CREATE INDEX
```

An **index** can also be removed.

```
personal=# DROP INDEX my_own_index;  
DROP INDEX
```

Create Indexes

```
personal=# CREATE INDEX ON people(last_name);
CREATE INDEX
personal=# \d people
```

Table "public.people"				
Column	Type	Collation	Nullable	Default
id	integer		not null	
first_name	character varying(20)			
last_name	character varying(50)			
social_sec	character varying(100)			

Indexes:

```
"people_pkey" PRIMARY KEY, btree (id)
"people_last_name_idx" btree (last_name)
"people_social_sec_key" UNIQUE CONSTRAINT, btree (social_sec)
```

Omitting the index name will create an automated name.

Create Indexes

```
CREATE INDEX <index_name>  
ON <table_name> USING <method>  
(<column_name> DESC) ;
```

The method used to search the indexed key field can be specified with the **USING** construct.

By default, the method used is a self-balancing tree (B-Tree).


The values available for **<method>** are: btree, hash, gist & gin.

The sorting order of the indexed search field can be specified using **ASC** and **DESC**. By default it is sorted ascendingly.

Indexes Impact on Performance

The data structure must be created and maintained.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla Black	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
5	Johan	Schmidt	Black	...
6	Patrick	Peterson	White	...
...



Index table	
Team	Id
Black	5
Black	8
Black	11
Black	14
Black	16
Gorilla Black	2
...	...

Any DML command on the table (**UPDATE**, **DELETE**, **INSERT**) will require updating the index and will add an overhead to these operations.

Indexes Impact on Performance

Defining indexes for every field
would make all the search operations fast.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
5	Johan	Schmidt	Black	...
6	Patrick	Peterson	White	...
...

Index 1

Index 2

Index 3

Index 4

...

But would also multiply the overhead of any **UPDATE**, **DELETE** and **INSERT** operation.

Indexes Impact on Performance

Bigger tables will have bigger indexes that will take longer to build and maintain.

Fact table				
Id	Name	Last name	Team	...
1	Allison	Müller	White	...
2	Alicia	Smith	Gorilla	...
3	Daniela	Hebbs	White	...
4	Charles	Bane	Gorilla	...
...
999...	Patrick	Peterson	White	...
...

Index 1

Index 2

Index 3

Index 4

...

Having too many fields indexed on large tables may reduce dramatically the performance of DML operations on the table.

We learned ...

- What are the database throughput and workload.
- That database throughput can be optimized using faster hardware or combining multiple database instances in a cluster.
- That database throughput can be optimized using a cache and indexes.
- That caching is a mechanism to store data in a more accessible location.
- That indexes are very useful to speed up the searching time, but will have a negative impact on DML commands.

Self Study



- Dense vs. Sparse indexes.
- Explore what other things can **CREATE INDEX** do.
 - Partial indexes.
 - Multiple column indexes.
- PostgreSQL Index methods: B-Tree, GiST, Hash and GIN.

A large group of people, mostly young adults, are posing for a group photo in a room with a projector screen in the background. They are arranged in several rows, some standing, some sitting, and some lying on the floor. Many are making peace signs or other celebratory gestures. The room has a white ceiling with recessed lights and a white wall with a projector screen. The overall atmosphere is festive and celebratory.

THANK YOU

Contact Details
DCI Digital Career Institute gGmbH