Relational Databases (SQL)

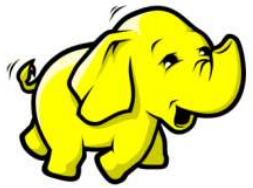
Michael Enudi

Journey through the world of databases and data engineering



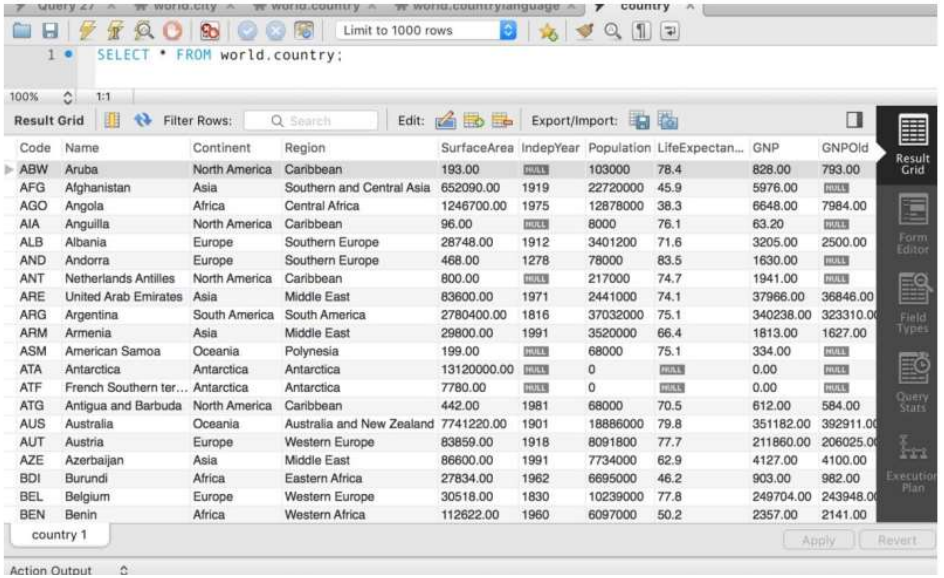




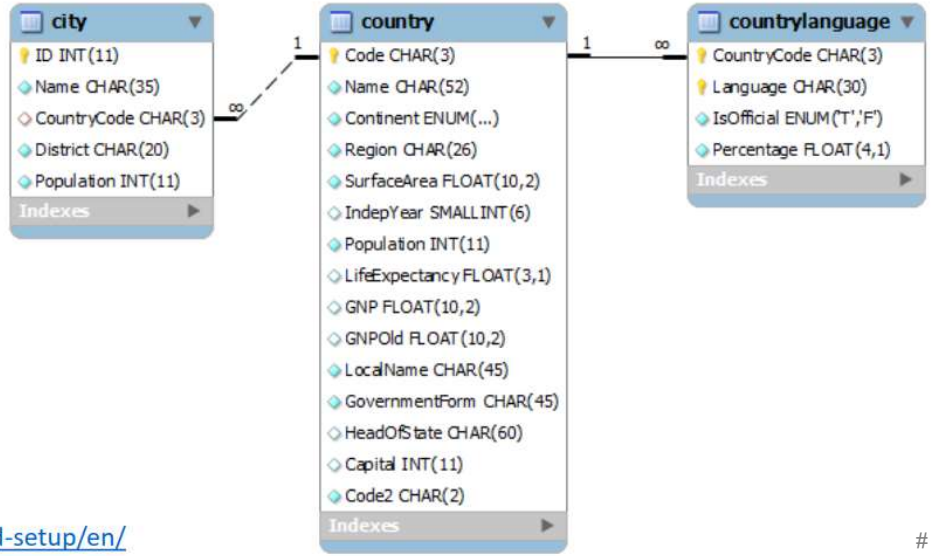




Relational Databases

- Relations / Tables
- Record / Row / Tuple
- Primary Key(s)
- Fields / Columns / Properties
- Data types
- NULL
- Constraints
- Uniqueness
- Relationships
- Transactions (ACID)
- Index
- ERD
- RDBMS

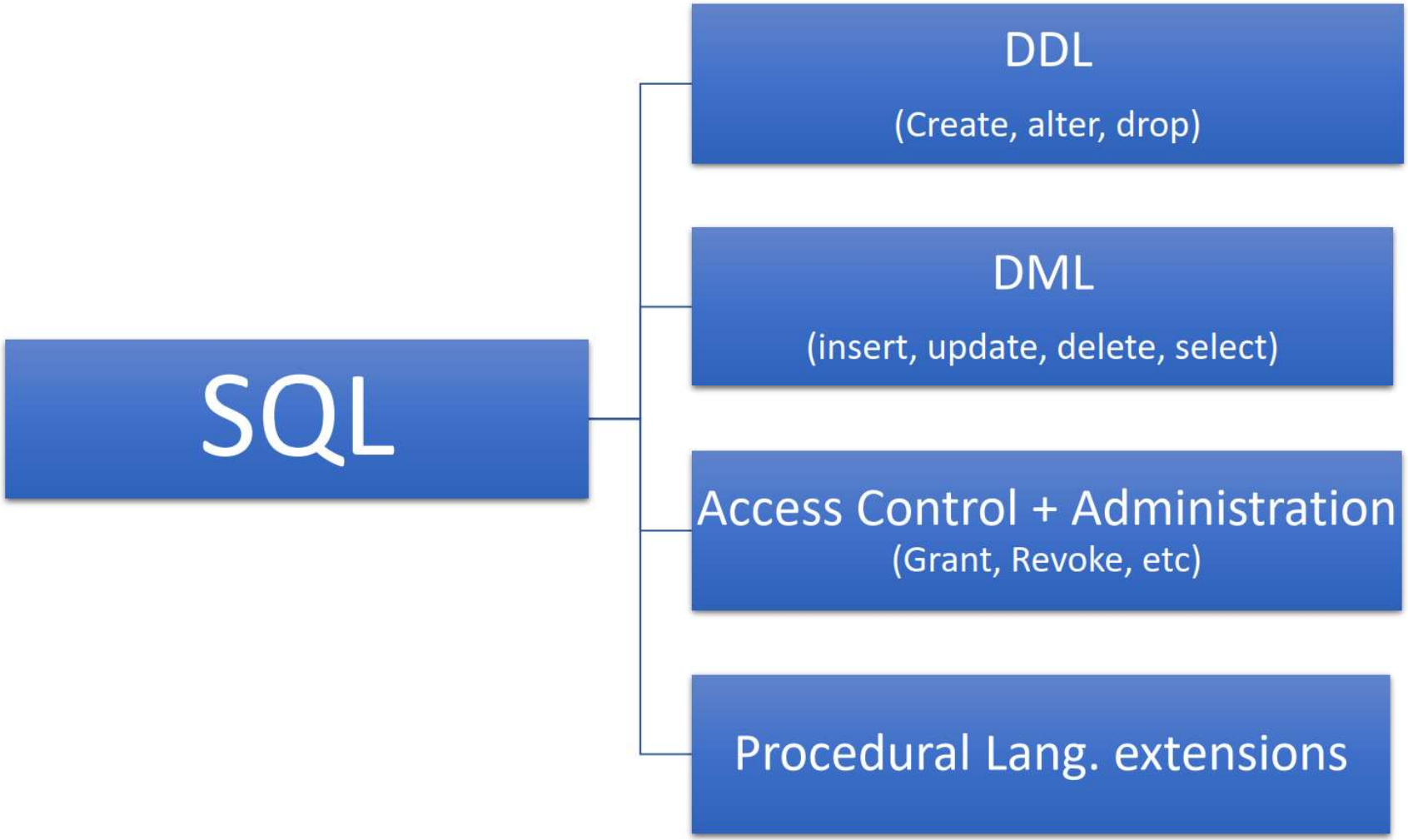


Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectan...	GNP	GNPOld
ABW	Aruba	North America	Caribbean	193.00	NULL	103000	78.4	828.00	793.00
AFG	Afghanistan	Asia	Southern and Central Asia	652090.00	1919	22720000	45.9	5976.00	NULL
AGO	Angola	Africa	Central Africa	1246700.00	1975	12878000	38.3	6648.00	7984.00
AIA	Anguilla	North America	Caribbean	96.00	NULL	8000	76.1	63.20	NULL
ALB	Albania	Europe	Southern Europe	28748.00	1912	3401200	71.6	3205.00	2500.00
AND	Andorra	Europe	Southern Europe	468.00	1278	78000	83.5	1630.00	NULL
ANT	Netherlands Antilles	North America	Caribbean	800.00	NULL	217000	74.7	1941.00	NULL
ARE	United Arab Emirates	Asia	Middle East	83600.00	1971	2441000	74.1	37966.00	36846.00
ARG	Argentina	South America	South America	2780400.00	1816	37032000	75.1	340238.00	323310.00
ARM	Armenia	Asia	Middle East	29800.00	1991	3520000	66.4	1813.00	1627.00
ASM	American Samoa	Oceania	Polynesia	199.00	NULL	68000	75.1	334.00	NULL
ATA	Antarctica	Antarctica	Antarctica	13120000.00	NULL	0	NULL	0.00	NULL
ATF	French Southern ter...	Antarctica	Antarctica	7780.00	NULL	0	NULL	0.00	NULL
ATG	Antigua and Barbuda	North America	Caribbean	442.00	1981	68000	70.5	612.00	584.00
AUS	Australia	Oceania	Australia and New Zealand	7741220.00	1901	18886000	79.8	351182.00	392911.00
AUT	Austria	Europe	Western Europe	83859.00	1918	8091800	77.7	211860.00	206025.00
AZE	Azerbaijan	Asia	Middle East	86600.00	1991	7734000	62.9	4127.00	4100.00
BDI	Burundi	Africa	Eastern Africa	27834.00	1962	6695000	46.2	903.00	982.00
BEL	Belgium	Europe	Western Europe	30518.00	1830	10239000	77.8	249704.00	243948.00
BEN	Benin	Africa	Western Africa	112622.00	1960	6097000	50.2	2357.00	2141.00



World database : https://dev.mysql.com/doc/world-setup/en/

SQL

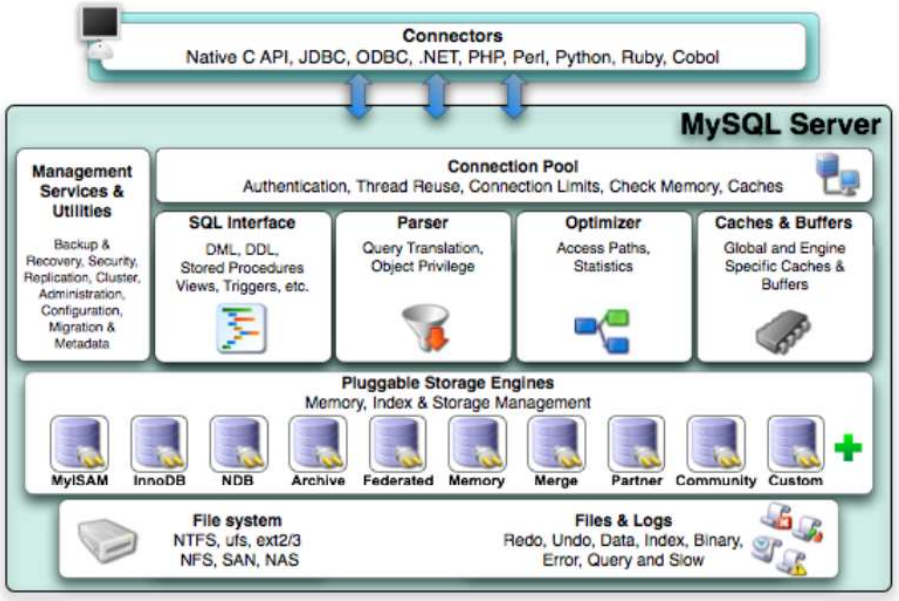

SQL
DDL
(Create, alter, drop)
DML
(insert, update, delete, select)
Access Control + Administration
(Grant, Revoke, etc)
Procedural Lang. extensions

MySQL

- Authored in 1995 by a Swedish company MySQL AB
- Currently developed by Oracle Incorporated
- Current version – 8.0.15 as at Feb. 2019
- According to db-engines
  - The second most used database in the world
  - The most deployed database in the world for web servers
- Written C and C++
- Vast language support
- Facebook has one of the known largest deployment of MySQL with over 1.3 billion users.

https://www.itworld.com/article/2831999/facebook--other-web-giants-unite-to-scale-mysql.html

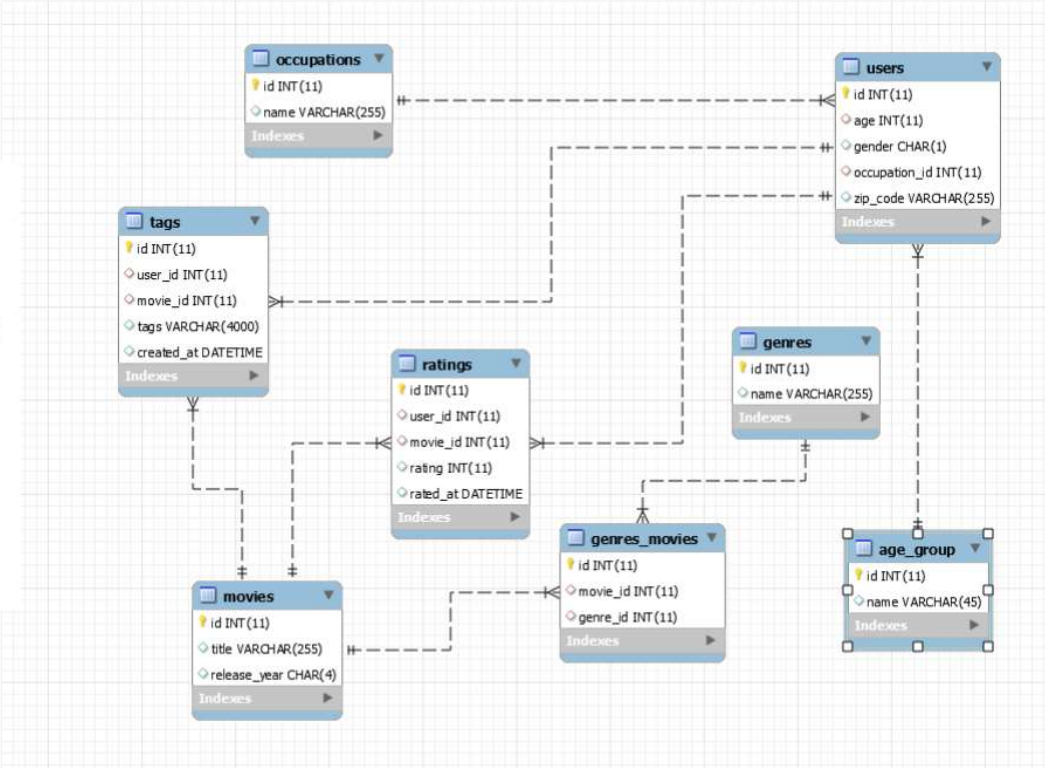
- Pluggable and flexible storage engine architecture. Supports a number out of the box
- Default storage engine today is InnoDB



Movielens Dataset

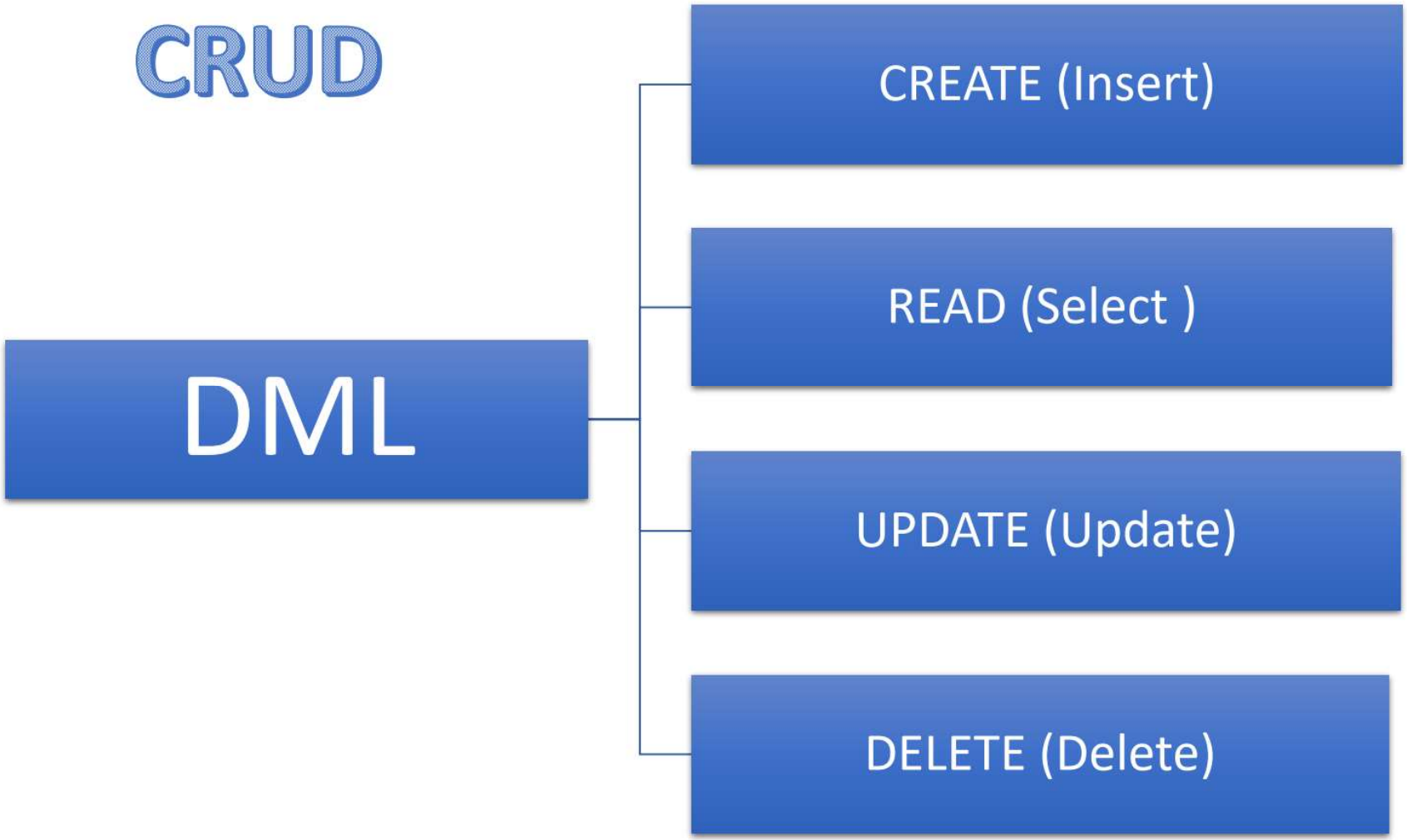
movielens

MovieLens is a web site that helps people find movies to watch. It has hundreds of thousands of registered users. We conduct online field experiments in MovieLens in the areas of automated content recommendation, recommendation interfaces, tagging-based recommenders and interfaces, member-maintained databases, and intelligent user interface design.



Movielens dataset : https://grouplens.org/datasets/movielens/ from GroupLens Research

Data Processing in RDBMS



```
>>> Let the learning begin
```

A Relational Database Management System (RDBMS).

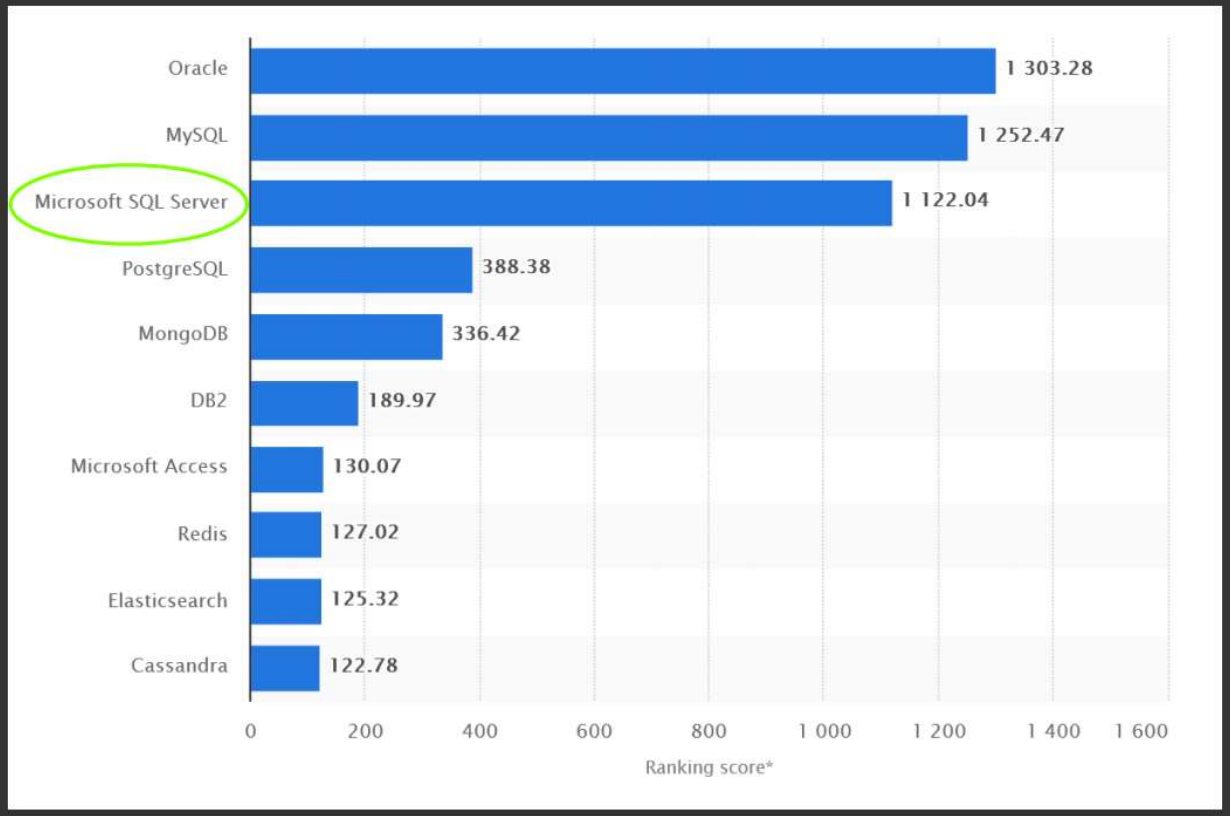
Definition

A DBMS is a large collection of interdependent programs all working together to define, construct, manipulate, protect and otherwise manage a database. An RDBMS is the most popular kind of DBMS.

SQL is a programming language for talking to your RDBMS.

>>> RDBMS

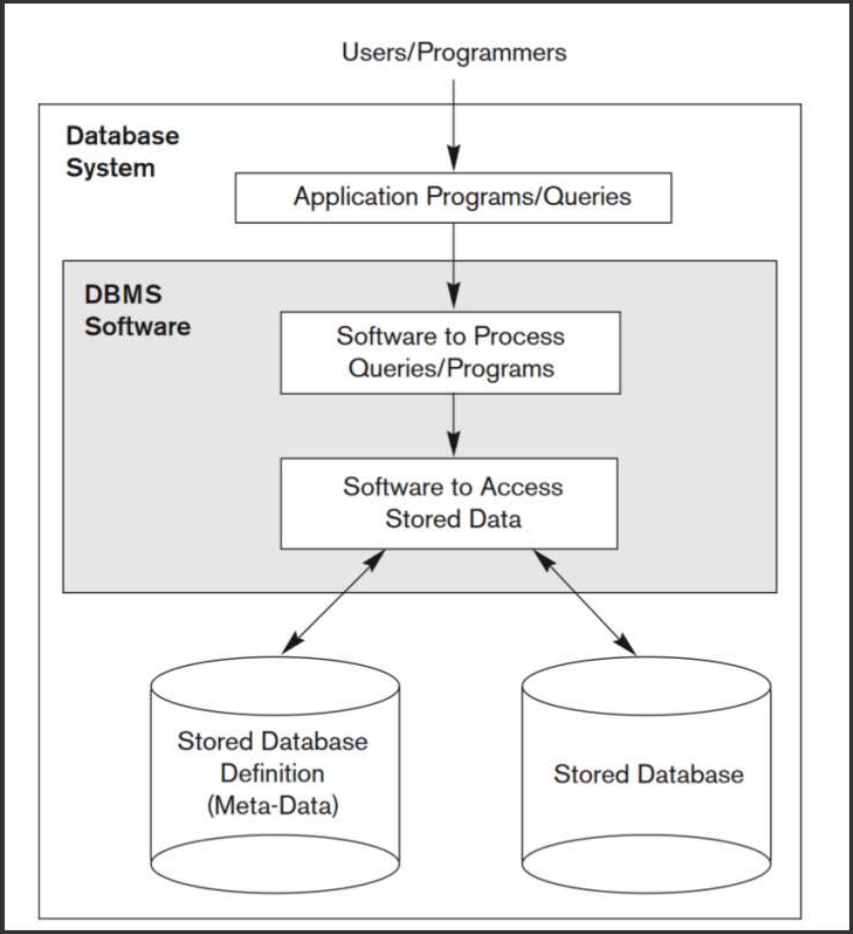
The most popular Relational Database Management Systems



Source: statista.com

>>> DBMS

A layer of abstraction between human and machine



```
>>> RDBMS
```

To talk to humans and machines, the RDBMS should have a model of the world that is intuitive to both. This model is called the Relational Model.

Intuition

The Relational Model is the 'common tongue' between the humans and the machines. It has a nice formal mathematical definition, so it is easy for machines to work with. For the humans, it has a simple intuitive description in terms of tables and relationships between tables!

>>> The anatomy of a table

Table name

Friends			
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

>>> The anatomy of a table

Row (record)

Friends			
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

>>> The anatomy of a table

Column names (attribute names)

Friends			
FriendID	FirstName	LastName	FavColour
1	*X*	*A*	red
2	*Y*	*B*	blue
3	*Z*	*C*	NULL

>>> The anatomy of a table

Primary key	Friends		
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

* Every table should have a primary key
* No two rows can have the same entry
* There must be no NULL entries

```
>>> One more thing: The data types of attributes
```

```
Friends(FriendID, FirstName, LastName, FavColour)
```

>>> One more thing: The data types of attributes

Friends(FriendID, FirstName, LastName, FavColour)

int

Definition

An integer is a positive or negative whole number.

>>> One more thing: The data types of attributes

Friends(FriendID, FirstName, LastName, FavColour)

varchar varchar varchar

Definition

Varchar stands for 'variable length character.'
It is a string of characters of undetermined length.

```
>>> Relationships between tables overview
```

1. One-to-many relationships
2. Primary and foreign keys
3. Many-to-many relationships
4. One-to-one relationships

>>> One-to-many relationships

* For each car there are *many* wheels.

>>> One-to-many relationships

* For each car there are *many* wheels.
  But each wheel belongs to only *one* car.
* One bank can have *many* accounts.
  But each account belongs to *one* bank.

>>> One-to-many relationships

* For each friend there are *many* pets.
  But each pet belongs to only *one* friend.

Where do we put the extra pets?

Friends			
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

>>> One-to-many relationships

* For each friend there are *many* pets.
  But each pet belongs to only *one* friend.

Where do we put the extra pets?

Friends				
FriendID	FirstName	…	PetName_1	PetName_2
1	X	…	NULL	NULL
2	Y	…	Chikin	NULL
3	Z	…	Cauchy	Gauss

>>> Problems with putting them in the same table

Ideas?

Friends				
FriendID	FirstName	…	PetName_1	PetName_2
1	X	…	NULL	NULL
2	Y	…	Chikin	NULL
3	Z	…	Cauchy	Gauss

>>> Problems with putting them in the same table

* Have to store NULL in every entry with no pet
* What if I meet a friend with 3+ pets? Many more NULLs
* New one-to-many relationship between pets and toys?
* Pets are tied to owners. Delete an owner $\rightarrow$ delete pets
* Ambiguity. Is information related to pets or owners?

>>> What if we do this instead?

Friends			
FriendID	FirstName	…	PetName
1	X	…	NULL
2	Y	…	Chikin
3	Z	…	Cauchy
3	Z	…	Gauss

This causes data redundancy

>>> What we do instead is…

Create another table.

Pets			
PetID	PetName	PetDOB	FriendID
1	Chikin	24/09/2016	2
2	Cauchy	01/03/2012	3
3	Gauss	01/03/2012	3

Foreign key

>>> The foreign key 'points at' the primary key

Pets			
PetID	PetName	…	FriendID
1	Chikin	…	2
2	Cauchy	…	3
3	Gauss	…	3

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Many

>>> The foreign key 'points at' the primary key

Pets			
PetID	PetName	…	FriendID
1	Chikin	…	2
2	Cauchy	…	3
3	Gauss	…	3

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

>>> Joining the tables

FriendsPets						
PetID	PetName	…	FriendID	FriendID	FirstName	…
1	Chikin	…	2	2	Y	…
2	Cauchy	…	3	3	Z	…
3	Gauss	…	3	3	Z	…

Primary/foreign key pair

>>> Primary and foreign keys

* Foreign key 'points at' the primary key
* Two rows can share same foreign key value
* Two rows can not share same primary key value
* Primary key can never be NULL
* All tables should have a primary key
* A PK or FK can be made of more than one column.

For example, a company might sell group holiday packages and the primary key of their Customer table might be made of a GroupID and GroupMemberNumber.

>>> Many-to-many relationship

* A class has many students,
  and a student attends many classes.
* A company has many investors,
  and an investor invests in many companies.
* A person engages with many government departments,
  and a government department engages with many people.

>>> Many-to-many relationship

* Each friend can scratch many backs,
  and a back can be scratched by many friends

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Scratched			
ScratcherID	Date	Time	ScratcheeID
1	05/09/2018	12:00pm	2
1	05/09/2018	12:30pm	3
2	06/09/2018	11:00am	1
3	07/09/2018	10:00am	1

>>> Many-to-many relationship

* Each friend can scratch many backs,
  and a back can be scratched by many friends

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Scratched			
ScratcherID	Date	Time	ScratcheeID
1	05/09/2018	12:00pm	2
1	05/09/2018	12:30pm	3
2	06/09/2018	11:00am	1
3	07/09/2018	10:00am	1

>>> Many-to-many relationship

* Each friend can scratch many backs,
  and a back can be scratched by many friends

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Friends		
FriendID	FirstName	…
1	X	…
2	Y	…
3	Z	…

Scratched			
ScratcherID	Date	Time	ScratcheeID
1	05/09/2018	12:00pm	2
1	05/09/2018	12:30pm	3
2	06/09/2018	11:00am	1
3	07/09/2018	10:00am	1

>>> One-to-one relationship

* A person can have at most one head,
  and each head belongs to only one person
* A table record has exactly one primary key value,
  and each primary key value belongs to exactly one record
* A user has one set of log-in details,
  and each set of log-in details belong to one user

>>> One-to-one relationship

* One friend can have at most one passport, and each passport belongs to only one friend

Friends					
FriendID	FirstName	…	PptCountry	PptNo	PptExpiry
1	X		Australia	E1321	12/03/2021
2	Y		New Zealand	LA123	01/09/2032
3	Z		Monaco	S9876	19/06/2028

>>> SQL clause: FROM

The FROM clause specifies table(s)
to access in the SELECT statement (and others).

```
FROM MySchema.MyTable MyAlias
```

The above will not run because there is no SELECT. You'll use FROM in almost every query, though.

>>> SQL clause: SELECT

The SELECT clause allows you to choose columns.
You can select all columns with SELECT *

We will look at the execution of this query:

```
SELECT F.FirstName, F.FavColour
FROM Notes.Friends F;
```

Note: the alias F seems to have been used before it was created! We will learn about the (sometimes confusing) SQL order of execution.

>>> SELECT execution

```
SELECT F.FirstName, F.FavColour
```

Friends			
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

```
>>> SELECT execution
```

result

Unnamed	
FirstName	FavColour
X	red
Y	blue
Z	NULL

```
>>> WHERE execution
```

FROM Notes.Friends

Friends			
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

```
WHERE FavColour = 'red'
```

Friends			
FriendID	FirstName	LastName	FavColour
1	X	A	red
2	Y	B	blue
3	Z	C	NULL

```
SELECT FirstName, LastName
```

Unnamed			
ID	FirstName	LastName	FavColour
1	X	A	red

result

Unnamed	
FirstName	LastName
X	A

>>> Order of execution

1. FROM
2. WHERE
3. SELECT

>>> Two string functions

Function	Description
CONCAT	Concatenate columns
SUBSTRING	Extract characters

I will go through some examples...

>>> Three date/time functions

Function	Description
DAY	Extract the day (of the month)
MONTH	Extract the month
YEAR	Extract the year

I will go through some examples...

>>> SQL query: JOIN

A JOIN (also known as an INNER JOIN) pairs the records from one table with the records from another table, using a primary/foreign key pair.

We will look at the execution of this query:

```
SELECT F.firstName, P.petName
FROM Notes.Friends F JOIN Notes.Pets P
ON F.friendID = P.friendID;
```

>>> SQL query: JOIN

We will look at the execution of this query:

```
SELECT F.firstName, P.petName
FROM Notes.Friends F JOIN Notes.Pets P
ON F.friendID = P.friendID;
```

Another way to write the same query: implicit syntax

```
SELECT F.firstName, P.petName
FROM Notes.Friends F, Notes.Pets P
WHERE F.friendID = P.friendID;
```

```
>>> SQL query: JOIN
```

Yet another way to write the same query:

```
SELECT F.firstName, P.petName
FROM Notes.Friends F INNER JOIN Notes.Friends P
ON F.friendID = P.friendID
```

>>> SQL query: JOIN

Note that JOIN is an operator that is inside the FROM clause.

```
FROM Friends F JOIN Pets P ON F.FriendID = P.FriendID
```

Pets			
PetID	PetName	…	FriendID
1	Chikin		2
2	Cauchy		3
3	Gauss		3

Friends		
FriendID	FirstName	…
1	X	
2	Y	
3	Z	

>>> SQL query: JOIN

```
SELECT F.FirstName, P.PetName
```

Unnamed						
PetID	PetName	…	FriendID	FriendID	FirstName	…
1	Chikin	…	2	2	Y	…
2	Cauchy	…	3	3	Z	…
3	Gauss	…	3	3	Z	…

>>> SQL query: LEFT JOIN

The join query below (that we looked at earlier)
excludes any friends that have no pets (and vice versa).

```
SELECT F.firstName, P.petName
FROM Notes.Friends F JOIN Notes.Pets P
ON F.friendID = P.friendID;
```

LEFT JOIN keeps every row from the table on the left.

```
SELECT F.firstName, P.petName
FROM Notes.Friends F LEFT JOIN Notes.Pets P
ON F.friendID = P.friendID;
```

>>> SQL query: LEFT JOIN. Remember this?

```
FROM Friends F JOIN Pets P ON F.FriendID = P.FriendID
```

Pets			
PetID	PetName	…	FriendID
1	Chikin		2
2	Cauchy		3
3	Gauss		3

Friends		
FriendID	FirstName	…
1	X	
2	Y	
3	Z	

>>> The result was...

Unnamed						
PetID	PetName	…	FriendID	FriendID	FirstName	…
1	Chikin	…	2	2	Y	…
2	Cauchy	…	3	3	Z	…
3	Gauss	…	3	3	Z	…

>>> SQL operator: LEFT JOIN

If we did a LEFT JOIN instead we would get:

```
FROM Friends F LEFT JOIN Pets P ON F.FriendID = P.FriendID
```

Unnamed						
PetID	PetName	…	FriendID	FriendID	FirstName	…
NULL	NULL	…	NULL	1	X	…
1	Chikin	…	2	2	Y	…
2	Cauchy	…	3	3	Z	…
3	Gauss	…	3	3	Z	…

```
>>> SQL query: LEFT JOIN
```

result

Unnamed	
FirstName	PetName
X	NULL
Y	Chikin
Z	Cauchy
Z	Gauss

>>> SQL query: RIGHT JOIN

Question for the class:
What does RIGHT JOIN do?