UNIVERSITÀ
DEGLI STUDI
DI PADOVA

J. Pazzini
PADOVA UNIVERSITY, INFN

5 - RELATIONAL DATABASES

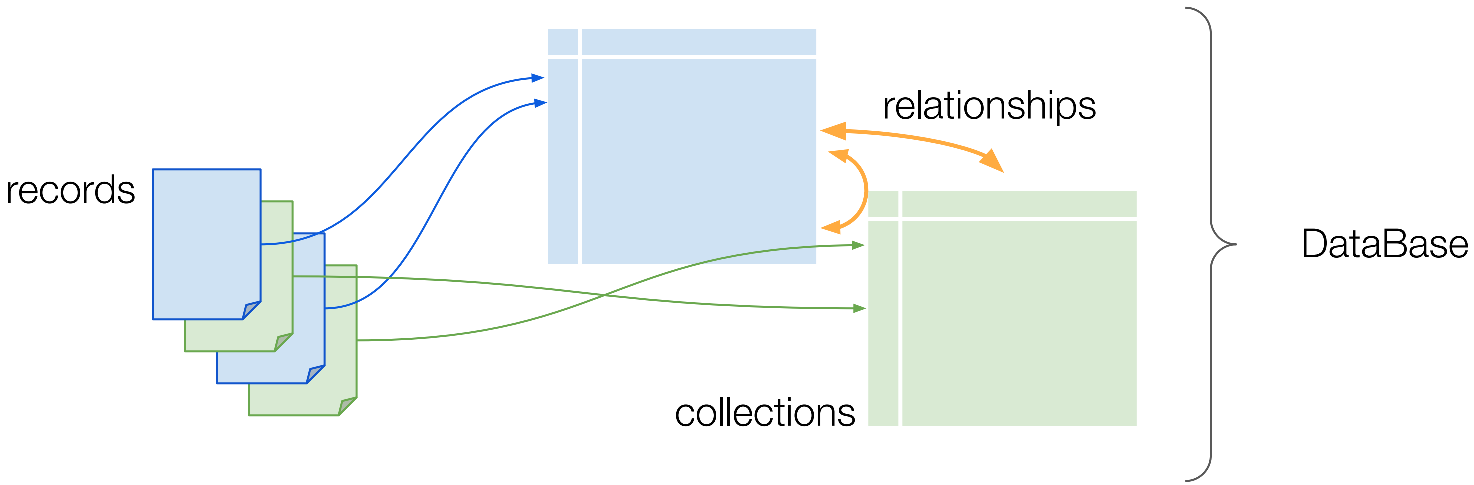
Management and Analysis of Physics Datasets - Module B

Physics of Data

A.A. 2023/2024

DATABASE

A ***database*** can loosely be defined as a set of (often/generally related) ***records*** organized and managed by a dedicated ***management system***



The database management system (DBMS) is a software that provides a number of basic functionalities to operate on the DB, including:

- defining / creating data
- querying / selecting data
- manipulating / processing data

DATABASE

DBs tend to improve the management of large datasets over (D)FSs:

- DBs work with the granularity of individual records:
  (e.g. accessing single events of a dataset instead of all records in a file)
  ⇒ optimization over file-based operations

- ALL data (and metadata) of a dataset is managed by the DBMS
  (e.g. data and detector configuration / banking wire transfers and users' data)
  ⇒ all dataset information is accessible, including relationships between records

- Little-to-no replication is required to handle multiple "views" of the same data
  (e.g. store the whole dataset in a DB, and filter based on relations across records)
  ⇒ allows creating multiple subsets of the dataset without data replication

DATABASE

```
file_1.dat
file_2.dat
file_3.dat
file_4.dat
file_5.dat
file_6.dat
...
```

id	feat1	feat2	feat3
…	…	…	…
100	“foo”	12.3	[1.9, -2.0, 0.3]
101	“something”	1.3e-5	[1.3, 2.3, 5.4]
102	“bar”	-0.2	[5.3, 1.2, -3.4]
…	…	…	…

```
result = []

for single_file in file_list:
   single_file.open()
   for line in single_file:
      if ‘something’ is in line:
         result.append(line)
   single_file.close()

print (result)
```

```
SELECT * FROM data WHERE feat1 = ‘something’;
```

DATABASE

In DBs it's often referred to the difference between the data "description" versus the data "content":

- Database **Model** (or Database **Schema**)
  → the *description of the data and the relationships across various parts of the dataset*

- Database **Instance** (or Database **State**)
  → the *actual data contained in the database at a given moment in time*

Recalling the various data models, it's clear that not all kinds of data can be described according to a *simple formal logical schema*
→ easy for structured data
→ not so much for semi- and un-structured data

DATABASE MANAGEMENT SYSTEM

The DB Management System (DBMS) is in charge of defining the schema, as well as updating/manipulating the instance of the DB

- Very often we (simplistically) refer to “a Database” to describe a “DB+DBMS”

A number of DBs can be defined based on various conditions:

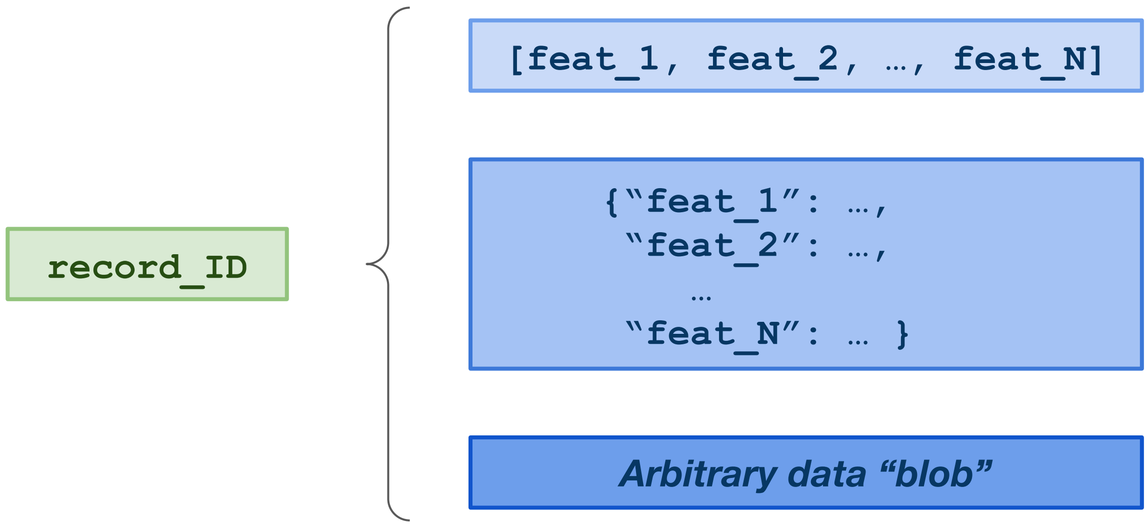
- Data Model
- Type of User access (single/multi-user)
- Architecture (centralized/federated/…)
- …

Possibly the most relevant categorization from the users perspective is the one based on the data model, *i.e.* the way each record is represented

DATABASE MODELS

In simple terms, each DB record can be thought a given data **VALUE** indexed by a **KEY**, used as a unique data identifier

The data itself might be structured, semi-structured, or (less often) unstructured, depending on the application



DATABASE MODELS

Relational

- Easily the most widely used
- Designed for **structured data** and their relationship
- Also (improperly) known as SQL DB

Key-Value

Wide-Column

Document

Graph

- NoSQL (Not Only SQL) DB
- Different implementations depending on the data-model
- Mostly designed for **non structured data** and horizontal scalability

others...

DATABASE MODELS

Relational

Key-Value

Wide-Column

Document

Graph

others...

Easily the most widely used

Designed for **structured data** and their relationship

Also (improperly) known as SQL DB

NoSQL (Not Only SQL) DB

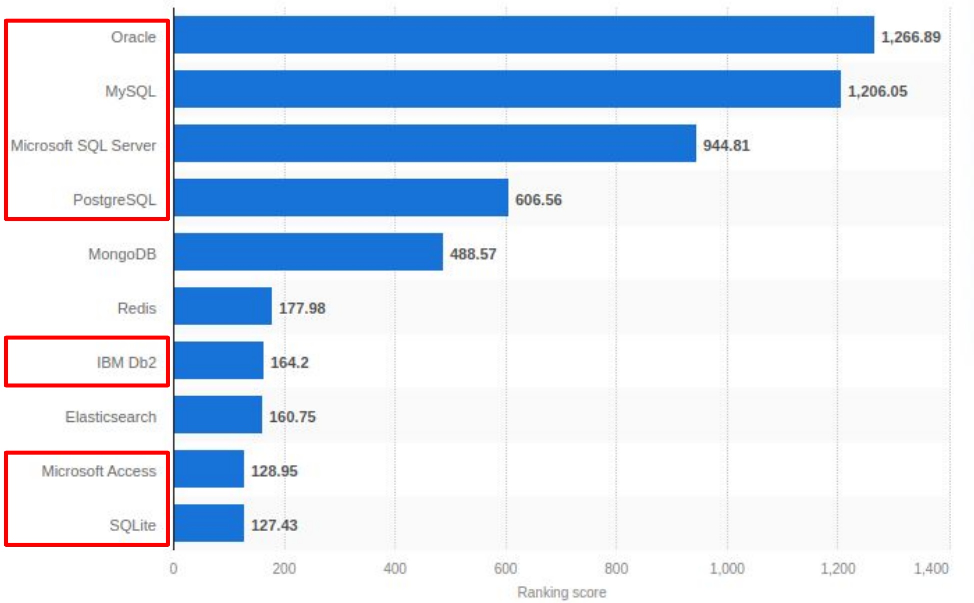
Different implementations depending on the data-model

Mostly designed for **non structured data** and horizontal scalability

RELATIONAL DB

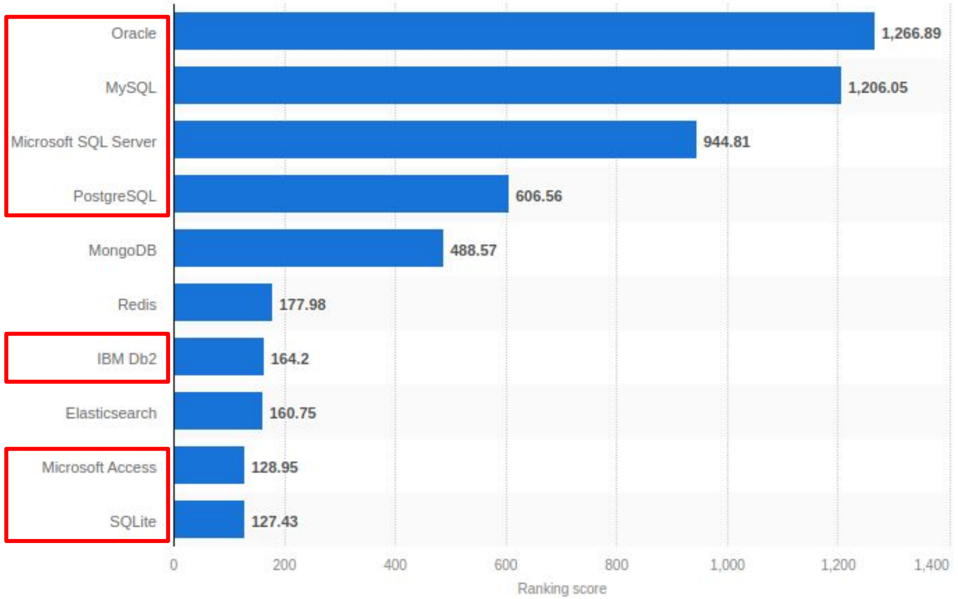
Extremely common DB choice behind most real-life services and applications

Ideal to have a mapping of relationships across multiple records of the dataset



Extremely common DB choice behind most real-life services and applications

Ideal to have a mapping of relationships across multiple records of the dataset



Based on relational algebra (introduced in 1970 by Ted Codd)

A Relational DB requires a well-known and predefined schema
→ ***structured datasets only***

RDBMS organizes the data in ***tables***, allowing for a ***mapping of relations across records of different tables***

RELATIONAL DB

Motorbike(bike_id, model, model_id)

bike_id	model	model_id
100	Goldwing	999
101	**AfricaTwin**	**12**
102	CBR1000	455

RELATIONAL DB

Motorbike(bike_id, model, model_id)

bike_id	model	model_id
100	Goldwing	999
101	**AfricaTwin**	**12**
102	CBR1000	455

Valve(valve_id, part_n)

valve_id	part_n
96	**vlv_1256**
97	vlv_1286
98	vlv_1193

Motorbike(bike_id, model, model_id)

bike_id	model	model_id
100	Goldwing	999
101	**AfricaTwin**	**12**
102	CBR1000	455

Model(model_id, year, engine_id, chassis_id)

model_id	year	engine_id	chassis_id
75	1989	56	90
12	**2016**	**82**	**97**
86	2011	83	112

Valve(valve_id, part_n)

valve_id	part_n
96	**vlv_1256**
97	vlv_1286
98	vlv_1193

Engine(engine_id, engine_name, num_cylinders, num_valves, valve_type)

engine_id	engine name	num cylinders	num valves	valve type
81	699F	4	3	128
82	**1000L**	**2**	**4**	**96**
83	350X	1	2	622

RELATIONAL DB

Key

Attribute

bike_id	model	model_id
100	Goldwing	999
101	**AfricaTwin**	**12**
102	CBR1000	455

Record (row/tuple)

Table

A **primary key** is an *attribute* or a *combination of multiple attributes* which identifies univocally a record

- PKs must be **unique** ⇒ different tuples *must not* have identical keys
- PKs must be **minimal** ⇒ use the bare minimum attributes to define the key (only attributes that cannot be null or "useless")

RELATIONAL DB

Primary Key		Foreign Key	Foreign Key
model_id	**year**	**engine_id**	**chassis_id**
75	1989	56	90
12	**2016**	**82**	**97**
86	2011	83	112

When assessing relationships across multiple tables, we do refer to the **primary key** and to the **foreign key(s)**

→ attribute (or list of attributes) acting as the link to the key to other tables

E.g. **engine_id** is primary key in the **Engine** table, AND a foreign key in the **Model** table

RELATIONAL DB

Each set of operations on a Relational DB is referred to as a ***Transaction***

RELATIONAL DB

Each set of operations on a Relational DB is referred to as a ***Transaction***

Transactions are 

meaning that are operations characterized by:

Atomicity →

Consistency →

Isolation →

Durability →

RELATIONAL DB

Each set of operations on a Relational DB is referred to as a ***Transaction***

Transactions are 

meaning that are operations characterized by:

Atomicity → *Either all operations in a transaction are executed in full or aborted (all-or-nothing)*

Consistency →

Isolation →

Durability →

RELATIONAL DB

Each set of operations on a Relational DB is referred to as a ***Transaction***

Transactions are 

meaning that are operations characterized by:

Atomicity → *Either all operations in a transaction are executed in full or aborted (all-or-nothing)*

Consistency → *Transactions must always move the DB from one valid state into another, preserving all the DB constraints (not the same “C” as in the CAP theorem)*

Isolation →

Durability →

RELATIONAL DB

Each set of operations on a Relational DB is referred to as a ***Transaction***

Transactions are

meaning that are operations characterized by:

Atomicity → *Either all operations in a transaction are executed in full or aborted (all-or-nothing)*

Consistency → *Transactions must always move the DB from one valid state into another, preserving all the DB constraints (not the same "C" as in the CAP theorem)*

Isolation → *Every transaction (even if executed concurrently with multiple other users) must generate the same results as in a single-user environment (unaware of other transactions happening at the same time)*

Durability →

RELATIONAL DB

Each set of operations on a Relational DB is referred to as a ***Transaction***

Transactions are 

meaning that are operations characterized by:

Atomicity → *Either all operations in a transaction are executed in full or aborted (all-or-nothing)*

Consistency → *Transactions must always move the DB from one valid state into another, preserving all the DB constraints (not the same "C" as in the CAP theorem)*

Isolation → *Every transaction (even if executed concurrently with multiple other users) must generate the same results as in a single-user environment (unaware of other transactions happening at the same time)*

Durability → *After a transaction, the DB state persists until it is changed by another transaction*

SQL

The “de facto” standard for transactions on RDBs is ***SQL (Structured Query Language)***

It was developed starting from 1974 and become a standard in 1987

SQL is a declarative language which contains the features of:

- **Data Definition Language (DDL)**
  → to define the DB schemas

- **Data Manipulation Language (DML)**
  → to query and modify data

- **Data Control Language (DCL)**
  → to perform authorization and access control

SQL (DDL) - CREATE TABLE

To create a table a **schema** is required → all attributes must be specified beforehand

```
CREATE TABLE Users (
    UserID     varchar(30),
    BadgeNum   int(16),
    FirstName  varchar(255),
    LastName   varchar(255),
    Age        int,
    OtherAttr  float
);
```

Upon creation of a table, the instance (the actual data contained) is empty

SQL (DDL) - CREATE TABLE

Data Type	Description
CHAR(n)	Holds a fixed length string with size n
VARCHAR(n)	Holds a variable length string with maximum size n
SMALLINT	Small integer (no decimal) between -32768 to 32767
INT	Integer (no decimal) between -2147483648 to 2147483647
FLOAT(n,d)	Small number with a floating decimal point. The total maximum number of digits is n with a maximum of d digits to the right of the decimal point.
DOUBLE(n,d)	Large number with a floating decimal point. The total maximum number of digits is n with a maximum of d digits to the right of the decimal point.
DATE	Date in format YYYY-MM-DD
DATETIME	Date and time in format YYYY-MM-DD HH:MI:SS
TIME	Time in format HH:MI:SS
BOOLEAN	True or False

SQL (DDL) - KEYS AND CONSTRAINTS

One primary key can be specified at most for each table, either inline

```
CREATE TABLE Users (
    UserID    varchar(30) PRIMARY KEY,
    BadgeNum  int(16),
    FirstName varchar(255),
    LastName  varchar(255),
    Age       int,
    OtherAttr float
);
```

AUTO_INCREMENT could be added to increment an Integer KEY by 1 with every new record added to the table

One primary key can be specified at most for each table, either inline

```
CREATE TABLE Users (
    UserID    varchar(30) PRIMARY KEY,
    BadgeNum  int(16),
    FirstName varchar(255),
    LastName  varchar(255),
    Age       int,
    OtherAttr float
);
```

AUTO_INCREMENT could be added to increment an Integer KEY by 1 with every new record added to the table

Or as a combination of multiple attributes

```
CREATE TABLE Test (
    AttrOne   varchar(255),
    AttrTwo   varchar(255),
    PRIMARY KEY (AttrOne, AttrTwo)
);
```

SQL (DDL) - KEYS AND CONSTRAINTS

Additionally, a number of constraints can be assigned to the attributes:

- **UNIQUE** → defines an attribute with no duplicate values (as the *key*)
- **NOT NULL** → prohibits NULL values for a column
- **DEFAULT** → sets a default value for a column
- **CHECK** → defines a constraint on the column values

```
CREATE TABLE Users (
    UserID    varchar(30) PRIMARY KEY,
    BadgeNum  int(16) NOT NULL UNIQUE,
    FirstName varchar(255) NOT NULL,
    LastName  varchar(255) NOT NULL,
    Age       int DEFAULT 18,
    OtherAttr float CHECK (OtherAttr > 0)
);
```

All SQL conditions (CHECK and others, see later…) are stated between parentheses, e.g.: **`(Attr. condition Value)`**

Additionally, a number of constraints can be assigned to the attributes:

- **FOREIGN KEY** → defines a foreign key of a table to reference attributes of other tables

```
CREATE TABLE Table1 (
    ThisKey   varchar(8) PRIMARY KEY,
    Feat1     varchar(30)
);


CREATE TABLE Table2 (
    PrimKey   varchar(10) PRIMARY KEY,
    Feat2     int(16),
    ForKey    varchar(8) UNIQUE,
    FOREIGN KEY (ForKey) REFERENCES Table1(ThisKey)
);
```

SQL (DDL) - TABLE ALTERATION

Once created, the schema of a table can be modified, keeping in mind all the constraints previously defined.

```
ALTER TABLE Users
ADD COLUMN Role CHAR(1) DEFAULT 'A' CHECK (Role IN ('A','B','C'));
```

The alteration of the schema is going to be propagated to the entire table
→ all records are going to be affected ⇒ it might take a really long time

SQL (DDL) - TABLE ALTERATION

Once created, the schema of a table can be modified, keeping in mind all the constraints previously defined.

```
ALTER TABLE Users
ADD COLUMN Role CHAR(1) DEFAULT 'A' CHECK (Role IN ('A','B','C'));
```

The alteration of the schema is going to be propagated to the entire table → all records are going to be affected ⇒ it might take a really long time

In the case a new NOT NULL attribute is added, all records contained in the table must now contain a valid value, hence a DEFAULT must be specified

This is one of the (many) issues related to the maintenance of a Relational DB.
The process of defining and maintaining the schema of a RDB is referred to as ***Database normalization***

SQL (DML)

SQL offers all CRUD operations:

Create	→	**INSERT**	→	Insert new records in a Table
Read	→	**SELECT**	→	Performs queries on the DB
Update	→	**UPDATE**	→	Modifies records
Delete	→	**DELETE**	→	Deletes records

In addition to the list of basic operations, it offers a number of additional features, e.g.:

- Conditional queries
- Table joins
- Aggregations
- Renaming and aliases
- Range operations
- Views

SQL (DML) - INSERT

Adding values to a table, with all limits and constraints provided by the table definition

Attributes which can be NULL or have a DEFAULT can be omitted

```
INSERT INTO Users (UserId, BadgeNum, FirstName, LastName, OtherAttr)
VALUES ('usr:00001',100,'Jacopo','Pazzini',1.8);
```

Multiple records can be inserted separated by a comma

```
INSERT INTO Users (UserId, BadgeNum, FirstName, LastName, OtherAttr)
VALUES
   ('usr:00002',101,'Matteo','Migliorini',3.5),
   ('usr:00003',102,'Stefano','Campese',2.9),
   ('usr:00004',103,'Federico','Agostini',4.7);
```

SQL (DML) - UPDATE

UPDATE allows to modify individual or multiple records by means of the WHERE statement

```
UPDATE Users
SET Role = 'C'
WHERE (LastName = 'Pazzini');
```

WHERE defines a condition, e.g.:

Attr = ‘value’

Attr > ‘value’

Att <= ‘value’

SQL (DML) - UPDATE

UPDATE allows to modify individual or multiple records by means of the WHERE statement

```
UPDATE Users
SET Role = 'C'
WHERE (LastName = 'Pazzini');
```

WHERE defines a condition, e.g.:
Attr = ‘value’
Attr > ‘value’
Att <= ‘value’

```
UPDATE Users
SET Role = 'B'
WHERE (LastName LIKE '%i');
```

All records where LastName ends with ‘i’

LIKE is used in a **WHERE** clause to search for a specified pattern:

- **_** → char wildcard
- **%** → string wildcard

SQL (DML) - DELETE

DELETE allows to remove all records from a table, or a selection of records based on a condition

```
DELETE FROM Users
WHERE (FirstName LIKE '__c%');
```

All records where the third character of the FirstName attribute is ‘c’

SQL (DML) - SELECT

The most powerful (and used) statement in SQL is SELECT

SELECT allows to query an entire table and extract the records fulfilling specific conditional statements

`*` is the notation used to indicate ALL the records of a table

```
SELECT * FROM Users;
```

Select all attributes of all records

```
SELECT FirstName,BadgeNum FROM Users;
```

Select specific attributes of all records

```
SELECT FirstName,BadgeNum FROM Users
WHERE (OtherAttr > 3);
```

Select specific attributes of records matching a condition

SQL (DML) - SELECT - ATTRIBUTE MODIFIERS

A SELECT statement can also contain operations on the attributes
→ the operations will not change the record, but only the value returned by the query

```
SELECT FirstName,OtherAttr*2,POWER(BadgeNum,2)
FROM Users
WHERE (OtherAttr > 3);
```

SQL (DML) - SELECT - ATTRIBUTE MODIFIERS

A SELECT statement can also contain operations on the attributes
→ the operations will not change the record, but only the value returned by the query

```
SELECT FirstName,OtherAttr*2,POWER(BadgeNum,2)
FROM Users
WHERE (OtherAttr > 3);
```

Additional conditions can be applied with operators such as BETWEEN, IN, and with boolean operators

```
SELECT FirstName,BadgeNum
FROM Users
WHERE (OtherAttr BETWEEN 1 AND 2)
   OR (Role IN ('A','C'));
```

BETWEEN → check if in range
IN → check if in list of values

SQL (DML) - SELECT - AGGREGATORS AND ORDERING

A number of other operators are defined in SQL to refine queries manipulating the results

```
MIN(attr)
MAX(attr)
COUNT(attr)
AVG(attr)
SUM(attr)
```

```
SELECT MAX(LastName)
FROM Users
WHERE FirstName LIKE ('%t%');
```

Ordering is defined in SQL with the ORDER BY clause, with default ordering being ascending (ASC) wrt the chosen attribute
To show only a given number of records after the query, LIMIT can be used

```
ORDER BY attr (DESC)
LIMIT n
```

```
SELECT FirstName
FROM Users
ORDER BY BadgeNum DESC
LIMIT 2;
```

SQL (DML) - SELECT - ALIASES

Very often is useful to rename the attributes of the result of a query to simplify further access to the records

```
SELECT DISTINCT FirstName AS UniqueNames
FROM Users
WHERE LastName LIKE ('%i');
```

DISTINCT check for unique values as attribute
(similar to unique in other languages)

SQL (DML) - SELECT - ALIASES

Very often is useful to rename the attributes of the result of a query to simplify further access to the records

```
SELECT DISTINCT FirstName AS UniqueNames
FROM Users
WHERE LastName LIKE ('%i');
```

DISTINCT check for unique values as attribute

The same can be done for entire tables

```
SELECT DISTINCT U.FirstName UniqueNames
FROM Users U
WHERE LastName LIKE ('%i');
```

table.attr can be also used to refer to a specific attribute

SQL (DML) - SELECT

The most powerful RDB feature is the ability to map relationships across records of multiple tables

Orders(OrderID, CustomerID, OrderDate, ShipperID)

OrderID	CustomerID	OrderDate	ShipperID
1001	94	2019-02-22	2
1002	22	2020-01-17	9
1003	94	2021-05-23	9

Customers(CustomerID, Name, Contact, Country, Address)

CustomerID	Name	Contact	Country	Address
92	ABC Inc	some@email.com	France	…
93	This S.P.A	this@email.com	Italy	…
94	That Inc	that@mail.com	Spain	…

Shippers(ShipperID, Name, Phone)

ShipperID	Name	Phone
2	DHL	123456789
3	UPS	987654321
4	SDA	123454321

SQL (DML) - SELECT - GROUP BY

A SELECT query with aggregators which returns a number of subset of records can be issued with a GROUP BY statement

```
SELECT COUNT(Country), Country
FROM customers
GROUP BY Country;
```

SQL (DML) - SELECT - GROUP BY

A SELECT query with aggregators which returns a number of subset of records can be issued with a GROUP BY statement

```
SELECT COUNT(Country), Country
FROM customers
GROUP BY Country;
```

The HAVING statement allows to filter the results by the aggregated attribute:

```
SELECT COUNT(Country) AS nCountry, Country
FROM customers
GROUP BY Country
HAVING nCountry > 5
ORDER BY Country;
```

HAVING only applies to the *aggregate* values, not to the plain attributes

SQL (DML) - SELECT - JOINS

The DB schema will reflect the relationships across multiple tables, which can be retrieved with ***join*** operations on the keys

Join is a central feature for mapping relationships, and can used to perform a number of relational operations

SQL (DML) - SELECT - JOINS

The DB schema will reflect the relationships across multiple tables, which can be retrieved with ***join*** operations on the keys

Join is a central feature for mapping relationships, and can used to perform a number of relational operations

OrderID	CustomerID	OrderDate	ShipperID
1001	94	2019-02-22	2
1002	22	2020-01-17	9
1003	94	2021-05-23	9

CustomerID	Name	Contact	Country	Address
92	ABC Inc	some@email.com	France	…
93	This S.P.A	this@email.com	Italy	…
94	That Inc	that@mail.com	Spain	…

SQL (DML) - SELECT - JOINS

The DB schema will reflect the relationships across multiple tables, which can be retrieved with ***join*** operations on the keys

Join is a central feature for mapping relationships, and can used to perform a number of relational operations

OrderID	CustomerID	OrderDate	ShipperID
1001	94	2019-02-22	2
1002	22	2020-01-17	9
1003	94	2021-05-23	9

CustomerID	Name	Contact	Country	Address
92	ABC Inc	some@email.com	France	…
93	This S.P.A	this@email.com	Italy	…
94	That Inc	that@mail.com	Spain	…

```
SELECT DISTINCT c.CustomerName, c.Country, c.ContactName
FROM customers c, orders o
WHERE c.CustomerID = o.CustomerID
AND o.ShipperID = 3;
```

SQL (DML) - SELECT - NESTED QUERIES

The same query could have been "combined" with another query via a ***nested query***

E.g.: *"what's that shipper name…? It's something like Federal whatever…"*

```
SELECT s.ShipperID
FROM shippers s
WHERE s.ShipperName LIKE 'Federal%';
```

Combining the two queries:

```
SELECT DISTINCT c.CustomerName, c.Country, c.ContactName
FROM customers c, orders o
WHERE c.CustomerID = o.CustomerID
AND o.ShipperID = (SELECT s.ShipperID
                   FROM shippers s
                   WHERE s.ShipperName LIKE 'Federal%');
```

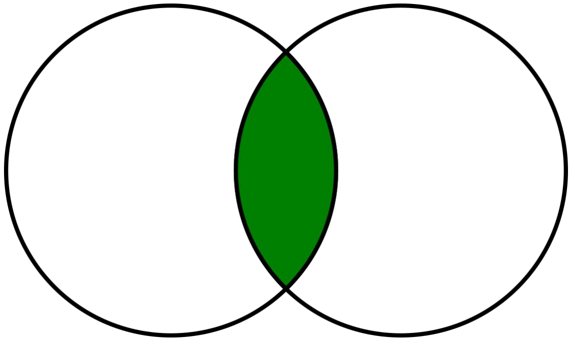
When possible, one should avoid nesting queries and JOIN tables *explicitly* beforehand!

SQL (DML) - SELECT - INNER JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

INNER JOIN



ID	Attr_B1	Attr_B2
2		
3		
7		

ID	Attr_A1	Attr_B1	Attr_B2
2			
3			

SQL (DML) - SELECT - INNER JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

```
SELECT DISTINCT c.CustomerName, c.Country, c.ContactName
FROM customers c
INNER JOIN orders o ON c.CustomerID = o.CustomerID
WHERE o.ShipperID = 3;
```

ID	Attr_B1	Attr_B2
2		
3		
7		

SQL (DML) - SELECT - INNER JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

```
SELECT DISTINCT c.CustomerName, c.Country, c.ContactName
FROM customers c
INNER JOIN orders o ON c.CustomerID = o.CustomerID
WHERE o.ShipperID = 3;
```

ID	Attr_B1	Attr_B2
2		
3		
7		

The same result as obtained before with the “implicit” join on the keys

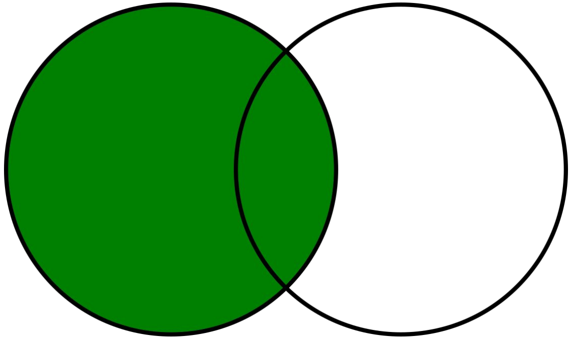
```
SELECT DISTINCT c.CustomerName, c.Country, c.ContactName
FROM customers c, orders o
WHERE c.CustomerID = o.CustomerID
AND o.ShipperID = 3;
```

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

ID	Attr_B1	Attr_B2
2		
3		
7		

LEFT JOIN



ID	Attr_A1	Attr_B1	Attr_B2
1		**NULL**	**NULL**
2			
3			

SQL (DML) - SELECT - LEFT JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

```
SELECT c.CustomerID, c.CustomerName, o.OrderID
FROM customers c
LEFT JOIN orders o ON c.CustomerID = o.CustomerID;
```

ID	Attr_B1	Attr_B2
2		
3		
7		

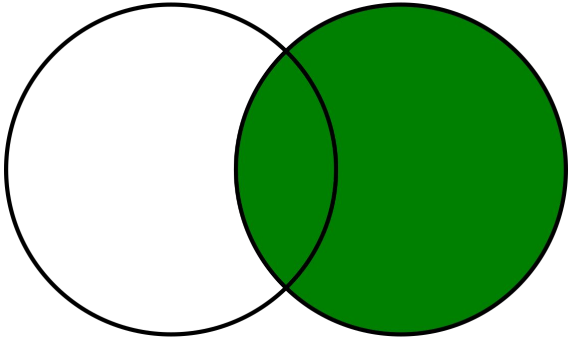
SQL (DML) - SELECT - RIGHT JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

ID	Attr_B1	Attr_B2
2		
3		
7		

RIGHT JOIN



ID	Attr_A1	Attr_B1	Attr_B2
2			
3			
7	**NULL**		

SQL (DML) - SELECT - RIGHT JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

```
SELECT c.CustomerID, c.CustomerName, o.OrderID
FROM customers c
RIGHT JOIN orders o ON c.CustomerID = o.CustomerID;
```

ID	Attr_B1	Attr_B2
2		
3		
7		

SQL (DML) - SELECT - RIGHT JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

```
SELECT c.CustomerID, c.CustomerName, o.OrderID
FROM customers c
RIGHT JOIN orders o ON c.CustomerID = o.CustomerID;
```

ID	Attr_B1	Attr_B2
2		
3		
7		

In this specific example, the RIGHT table is the Orders:

- *1 Customer might be associated to 0, 1 or more Orders*
- *1 Order is associated to 1 and only 1 Customer*

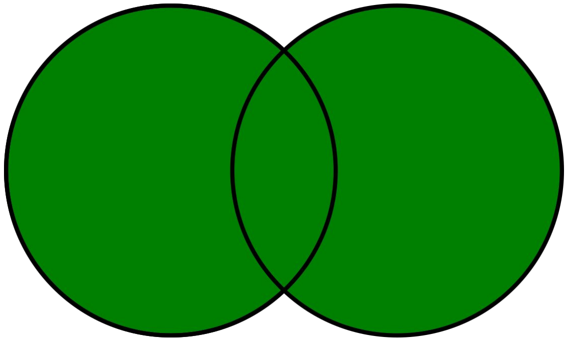
SQL (DML) - SELECT - FULL JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

ID	Attr_B1	Attr_B2
2		
3		
7		

FULL (OUTER) JOIN



ID	Attr_A1	Attr_B1	Attr_B2
1		**NULL**	**NULL**
2			
3			
7	**NULL**		

SQL (DML) - SELECT - FULL JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

ID	Attr_B1	Attr_B2
2		
3		
7		

No FULL/OUTER JOIN syntax in MySQL…

```
SELECT c.CustomerID, c.CustomerName, o.OrderID
FROM customers c
FULL JOIN orders o ON c.CustomerID = o.CustomerID;
```

SQL (DML) - SELECT - FULL JOIN

In SQL, and depending on the DBMS, we have 4 main type of explicit JOIN operations

ID	Attr_A1
1	
2	
3	

ID	Attr_B1	Attr_B2
2		
3		
7		

However, it can still be performed… with a workaround

```
SELECT c.CustomerID, c.CustomerName, o.OrderID
FROM customers c
LEFT JOIN orders o ON c.CustomerID = o.CustomerID

UNION

SELECT c.CustomerID, c.CustomerName, o.OrderID
FROM customers c
RIGHT JOIN orders o ON c.CustomerID = o.CustomerID;
```

UNION → Combines together the tables resulting from multiple SELECT statements

SQL (DML) - SELECT - VIEWS

In some cases, complex queries can be simplified by means of intermediate "virtual" tables, hosting the DB instance after a specific transaction

The View is kept up-to-date by the DBMS
⇒ every new query to a View will reflect the most recent DB instance

```sql
CREATE VIEW ItalianCustomers AS
SELECT *
FROM customers
WHERE customers.Country = 'Italy';
```

```sql
SELECT * FROM ItalianCustomers;
```

SQL (DCL)

The DCL component of the SQL language is extremely powerful, as it allows to:

- Create/remove users
- Assign/update passwords

```
CREATE USER ‘my_new_user’@’localhost’ IDENTIFIED BY ‘user_new_password’;
```

- Assign/update/remove privileges to users

```
GRANT SELECT, UPDATE ON db_name.table_name TO ‘my_new_user’@’localhost’;

REVOKE UPDATE ON db_name.table_name TO ‘my_new_user’@’localhost’;
```

- Create/maintain/roll-back DB transitions

Albeit this is arguably the most important component of the DBMS for the development, maintenance and deployment of a DB, we’ll not focus on it as it is less “user-oriented” and more “administrator-related”