TÉCNICO LISBOA

Sistemas de Informação e Bases de Dados 2020/2021

Aula 02: Introdução às Bases de Dados e ao SQL

Prof. Paulo Carreira

Class Outline

- How a database Organises Data
  - Tables, Columns, Rows, and Cells
  - Key, Primary Key, and Foreign Key
  - Table Schema and Database Schema
- Introduction to SQL
  - Creating database tables and populating them with information
  - Basic query structure: SELECT
  - Filters and Pattern Matching
  - Set operations and Joins

How a Database Organises Data

The Table

A two-dimensional grid

A **two-dimensional** grid that keeps data (facts) about (real-world) **Entities**

ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL
78230	Marta Isabel de Almeida Cardoso	34562732	S
78083	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L
78081	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M
78001	João Gomes Vultos Freitas	22231233	L
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L

Columns, lines, and cells

Data is organised into **columns**, **lines**, and **cells**

Columns are named and identified by Attributes

Lines are called 'Records' or 'Rows'

Data is recorded in Cells called 'Fields'

ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL
78230	Marta Isabel de Almeida Cardoso	34562732	S
78083	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L
78081	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M
78001	João Gomes Vultos Freitas	22231233	L
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L

Data Types, Domains, and Formats

For each column, the data in every cell has the same:

- Data Domain: Non-negative, min and max value range, minimum and maximum length, allowed characters, words, or codes from a set of values
- Data Type: Numeric, Text, Date, Binary
- Data Format: 5 digits, Lowercase vs Capitalised, License Plate, YYYY/MM/DD, etc

Schema = Columns → Data Domain

ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M

Header

Table Data

Every Column has an associated **Domain**, **Type** and **Format**

Atomic values

- Text

```
'John Smith', 'R2D2', 'Red', ''
```

- Numeric values

```
-1, 25, +6.34, 0.5, 25e-03
```

- Floats and Doubles

```
25f, +6.34d, 0.5d
```

- Dates

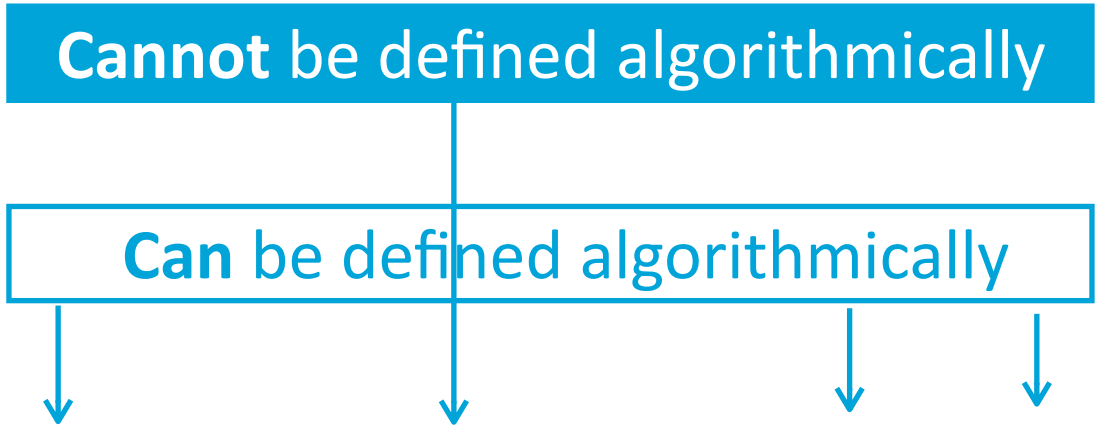
```
'1999-01-08', '08-JAN-1999'
```

Table

Definition

An organisation of data into columns and lines where all lines have a similar structure, defined by the schema of the table

Algorithmically- *vs*. non-algorithmically definable column domains



ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL
78230	Marta Isabel de Almeida Cardoso	34562732	S

Ordering of columns

If the ordering of columns is changed, information remains the same

ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL
78230	Marta Isabel de Almeida Cardoso	34562732	S
78083	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L
78081	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M
78001	João Gomes Vultos Freitas	22231233	L
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L

Ordering of columns

In terms of information contents, the order of the columns is irrelevant

Name	ID	TAX ID	T-Shirt
João Guilherme Silva da Cunha	76440	12345678	M
Tomás Pinto dos Santos	76469	91234567	M
David Miguel Redwanz Duque	76500	89012345	L
Pedro Daniel Diz Pinela	64833	67890123	M
Guilherme de Queiróz Rebelo Brum Gomes	76464	22394856	XL
Marta Isabel de Almeida Cardoso	78230	34562732	S
Filipe Emanuel Lourenço Ramalho Fernandes	78083	82533235	L
Gabriel Filipe Queirós Mesquita Delgado Freire	78081	23134539	M
João Gomes Vultos Freitas	78001	22231233	L
Ricardo Afonso Rodrigues da Silva Oliveira	76504	56372848	L

Ordering of rows

When the order of rows is changed, information remains the same

ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL
78230	Marta Isabel de Almeida Cardoso	34562732	S
78083	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L
78081	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M
78001	João Gomes Vultos Freitas	22231233	L
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L

Ordering of rows

In terms of information contents, the order of the rows is irrelevant

ID	Name	TAX ID	T-Shirt
78083	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L
78081	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M
78001	João Gomes Vultos Freitas	22231233	L
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M

The **row number** is **irrelevant**

Keys

Key values are the way to refer uniquely to a row

Key attributes

Primary Key →

ID	Name	TAX ID	T-Shirt
76440	João Guilherme Silva da Cunha	12345678	M
76469	Tomás Pinto dos Santos	91234567	M
76500	David Miguel Redwanz Duque	89012345	L
64833	Pedro Daniel Diz Pinela	67890123	M
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL
78230	Marta Isabel de Almeida Cardoso	34562732	S
78082	Filipe Emanuel Lourenço Ramalho Fernandes	82523225	L
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L

Retrieving a record given a key value is a fundamental feature/operation of database systems

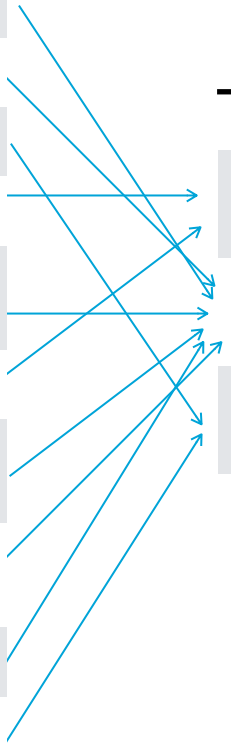
Table Relationships

A foreign key connects two tables

Foreign Key

Employee

ID	Name	TAX ID	T-Shirt	DID
76440	João Guilherme Silva da Cunha	12345678	M	EN
76469	Tomás Pinto dos Santos	91234567	M	EN
76500	David Miguel Redwanz Duque	89012345	L	MK
64833	Pedro Daniel Diz Pinela	67890123	M	HR
76464	Guilherme de Queiróz Rebelo Brum Gomes	22394856	XL	EN
78230	Marta Isabel de Almeida Cardoso	34562732	S	HR
78083	Filipe Emanuel Lourenço Ramalho Fernandes	82533235	L	EN
78081	Gabriel Filipe Queirós Mesquita Delgado Freire	23134539	M	EN
78001	João Gomes Vultos Freitas	22231233	L	EN
76504	Ricardo Afonso Rodrigues da Silva Oliveira	56372848	L	MK

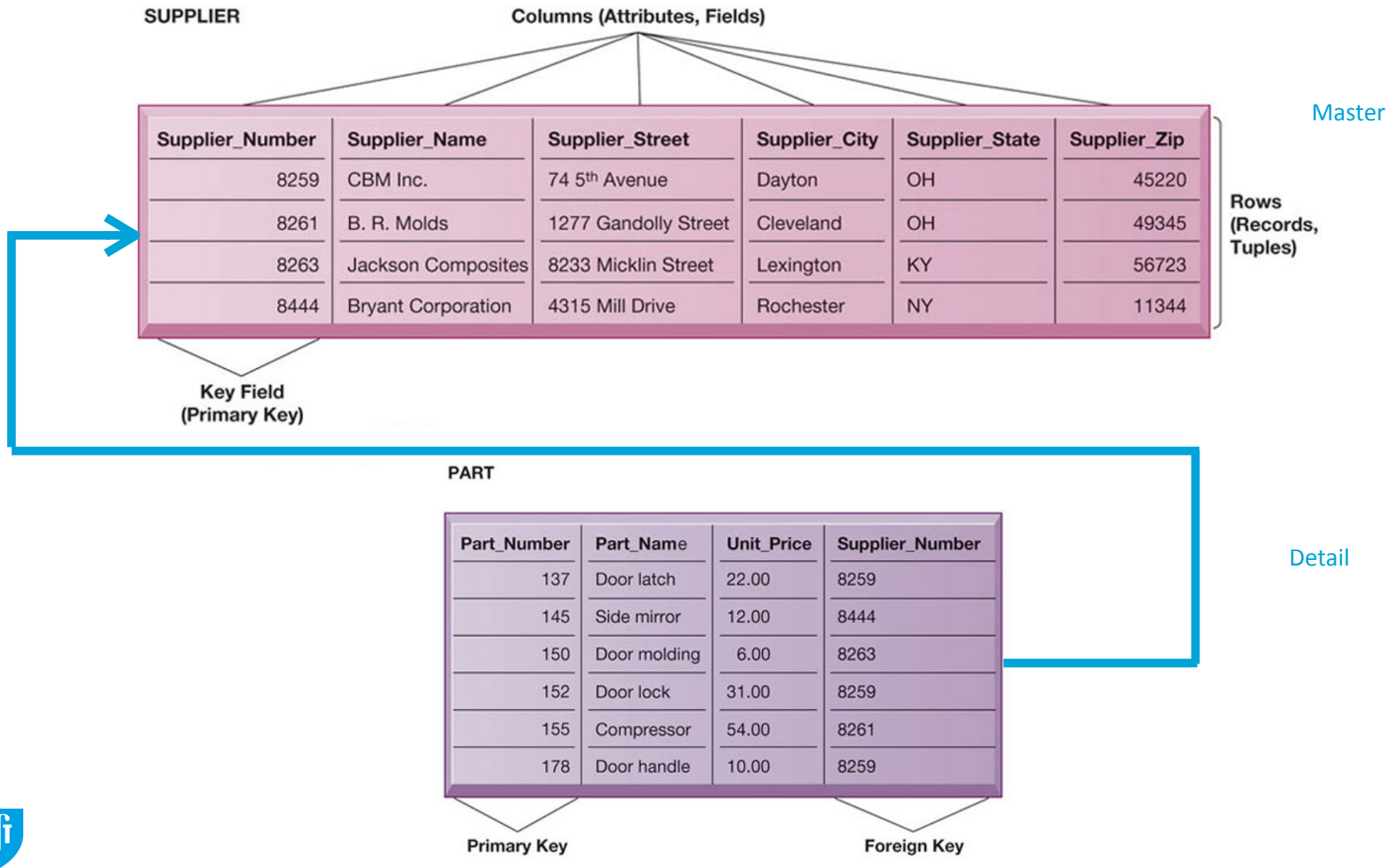


Department

DID	Name	Budget
HR	Human Resources	50 000
EN	Software Engineering	1 200 000
MK	Marketing	150 000

Primary Key

Master-detail related tables



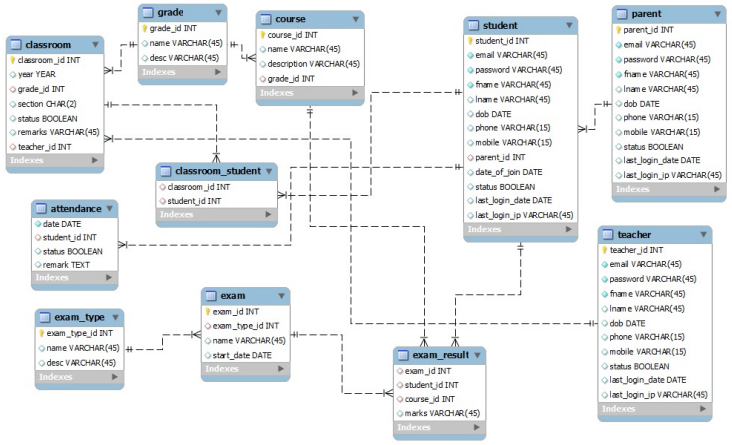
Tables Concepts Summary

- **Rows**: Records data different entities
- **Columns**: Represents attributes for entity
- **Fields**: Represents attribute for entity

- **Key field**: Field used to uniquely identify each record
- **Primary key**: Field in table used for key fields
- **Foreign key**: Primary key used in second table as look-up field to identify records from original table

Databases and Database Schemas

Database Schema



1. **Individual schemas of tables**: Define which data can be captured
2. **Relationships between tables**: Define how data can be related and navigated (the navigational structure)
3. **Integrity Constraints**: Define which data are not valid in the database

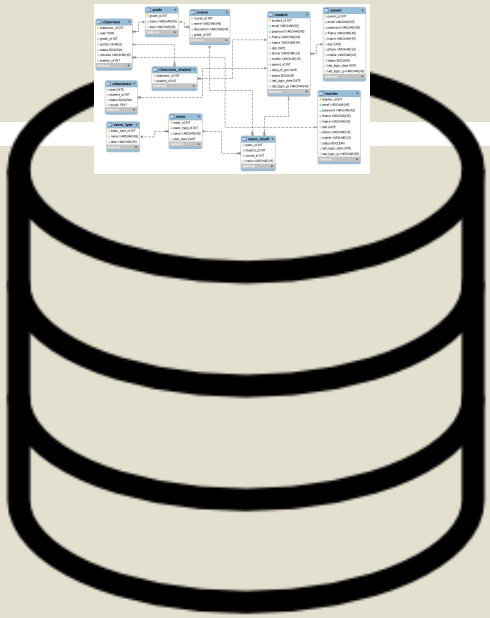
What is kept in the database

Schema

The description of data

Data

The actual data contents



Database

Definition

1. An **organised collection** of **data** (typically of large size) that is inter-relatable

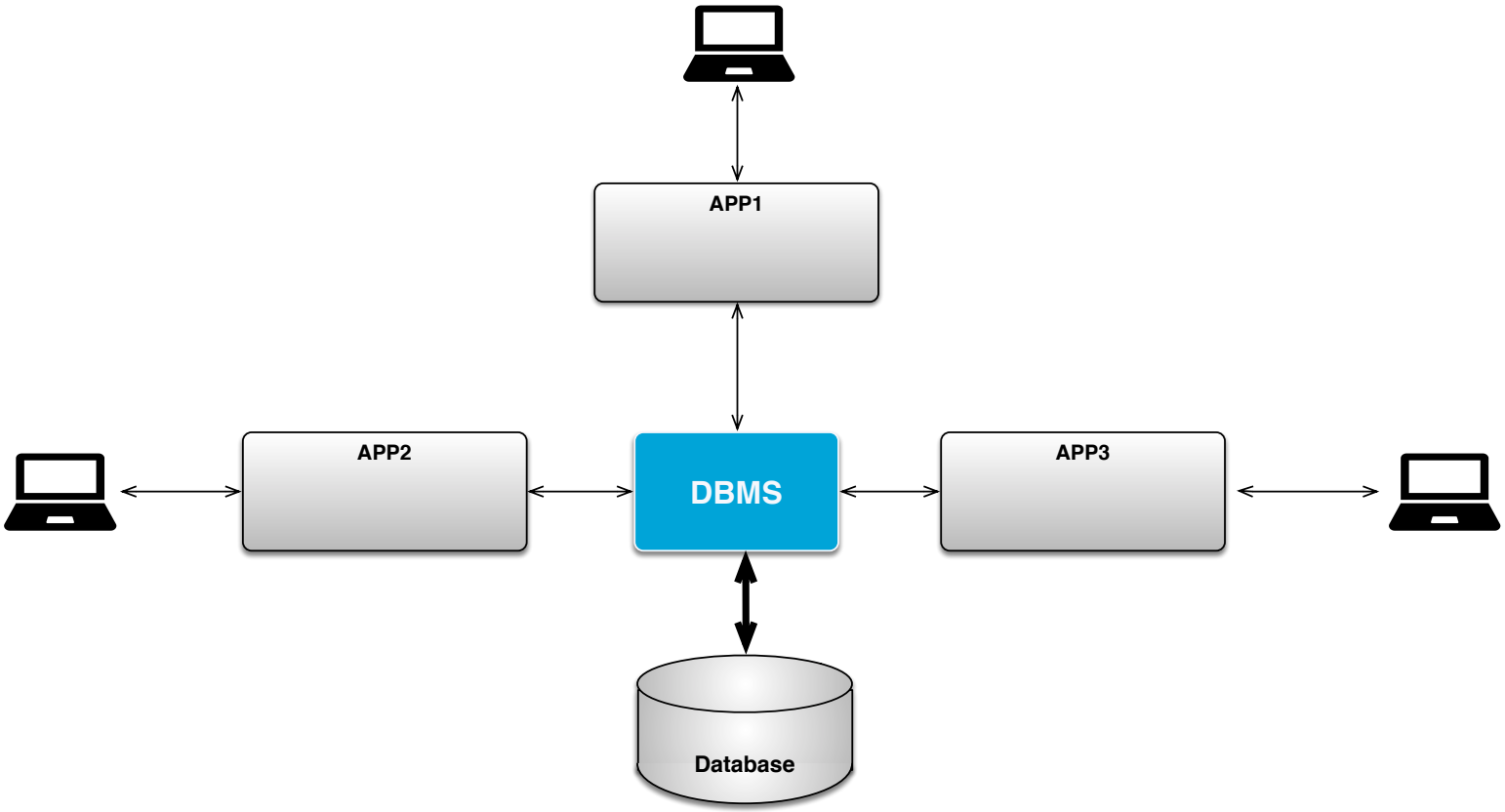
2. A **computerised system**, that allows data to be easily accessed, manipulated and updated

See also

What is a Database?

https://www.youtube.com/watch?v=t8jgX1f8kc4

DBMS



Applications do not contact the database directly

Database Management System (DBMS)

Definition

A **software package** that manages a Database allows data to be easily accessed, manipulated, and updated

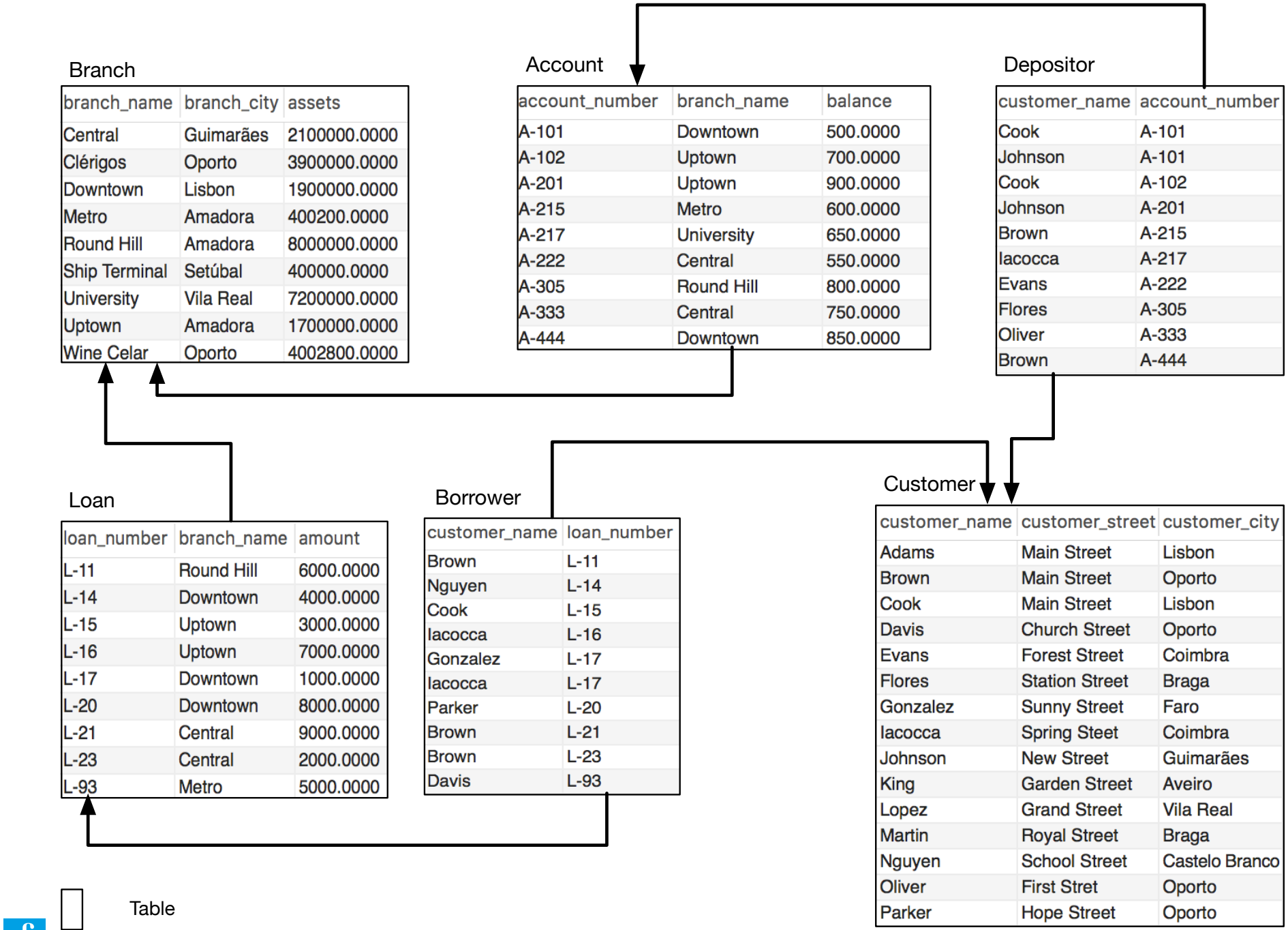
Introducing SQL

SQL Language

Organised into two sub-languages (parts):

1. Data Definition Language (SQL-DDL)
2. Data Query Language (SQL-DQL)

The Bank running example


Branch
branch_name | branch_city | assets
Central | Guimarães | 2100000.0000
Clérigos | Oporto | 3900000.0000
Downtown | Lisbon | 1900000.0000
Metro | Amadora | 400200.0000
Round Hill | Amadora | 8000000.0000
Ship Terminal | Setúbal | 400000.0000
University | Vila Real | 7200000.0000
Uptown | Amadora | 1700000.0000
Wine Celar | Oporto | 4002800.0000
Account
account_number | branch_name | balance
A-101 | Downtown | 500.0000
A-102 | Uptown | 700.0000
A-201 | Uptown | 900.0000
A-215 | Metro | 600.0000
A-217 | University | 650.0000
A-222 | Central | 550.0000
A-305 | Round Hill | 800.0000
A-333 | Central | 750.0000
A-444 | Downtown | 850.0000
Depositor
customer_name | account_number
Cook | A-101
Johnson | A-101
Cook | A-102
Johnson | A-201
Brown | A-215
Iacocca | A-217
Evans | A-222
Flores | A-305
Oliver | A-333
Brown | A-444
Loan
loan_number | branch_name | amount
L-11 | Round Hill | 6000.0000
L-14 | Downtown | 4000.0000
L-15 | Uptown | 3000.0000
L-16 | Uptown | 7000.0000
L-17 | Downtown | 1000.0000
L-20 | Downtown | 8000.0000
L-21 | Central | 9000.0000
L-23 | Central | 2000.0000
L-93 | Metro | 5000.0000
Borrower
customer_name | loan_number
Brown | L-11
Nguyen | L-14
Cook | L-15
Iacocca | L-16
Gonzalez | L-17
Iacocca | L-17
Parker | L-20
Brown | L-21
Brown | L-23
Davis | L-93
Customer
customer_name | customer_street | customer_city
Adams | Main Street | Lisbon
Brown | Main Street | Oporto
Cook | Main Street | Lisbon
Davis | Church Street | Oporto
Evans | Forest Street | Coimbra
Flores | Station Street | Braga
Gonzalez | Sunny Street | Faro
Iacocca | Spring Steet | Coimbra
Johnson | New Street | Guimarães
King | Garden Street | Aveiro
Lopez | Grand Street | Vila Real
Martin | Royal Street | Braga
Nguyen | School Street | Castelo Branco
Oliver | First Stret | Oporto
Parker | Hope Street | Oporto
Table
Value dependency (field)

Creating Tables

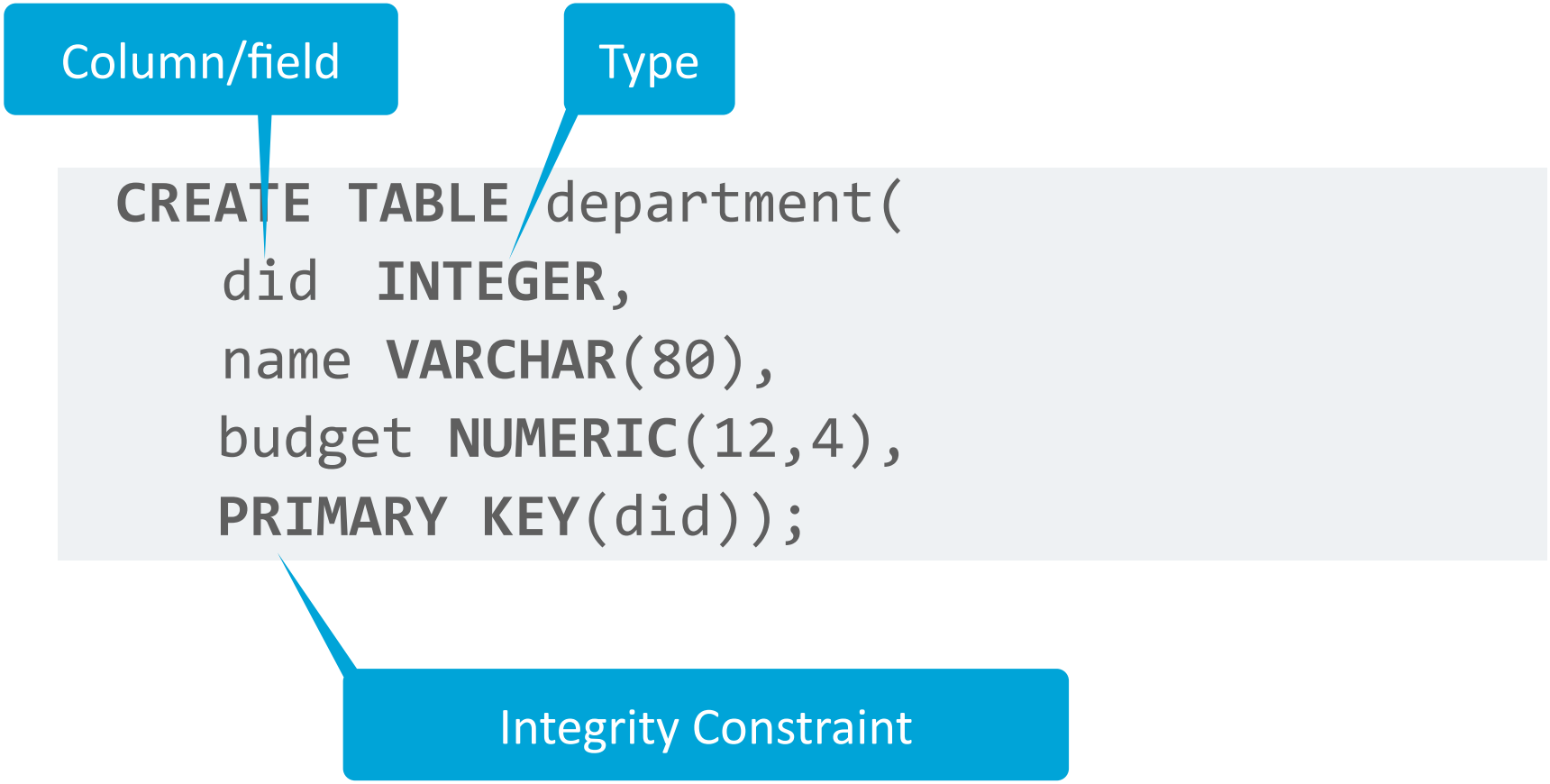
The CREATE TABLE statement

A new **table** is created with the command:

```
CREATE TABLE <table-name> (
    column1 type1,
     …
    columnn typen,
    <integrity-constraint1>,
    …
    <integrity-constraintm>
); [storage-options]
```

- The syntax of the storage options is dependent on the SGBD manufacturer

CREATE TABLE: Example



Removing a table

- Removing a table

```
DROP TABLE <table-name>
```

- Removing the table **department** from the database

```
DROP TABLE department;
```

Specifying foreign key constraints

```
CREATE TABLE account
   (account_number CHAR(5),
    branch_name  VARCHAR(80),
    balance NUMERIC(16,4),
    PRIMARY KEY(account_number),
    FOREIGN KEY(branch_name)
          REFRENCES branch(branch_name));
```

Creating, updating, and deleting rows

Inserting and querying rows

```
INSERT INTO <table-name>
  VALUES(<field-values>)
```

- Inserting a record into a table

```
INSERT INTO department VALUES(1,
  'Finance', 1000.0);
```

- Querying the records of a table

```
SELECT * FROM <table-name>
```

Schema Definition – Example 1

```
CREATE TABLE department(
    did INTEGER,
    name VARCHAR(80),
    budget NUMERIC(12,2));
```

```
INSERT INTO department VALUES(1, 'Finance', 1000.0);
INSERT INTO department VALUES(2, 'Marketing', 2000.0);
```

```
SELECT * FROM department;
```

```
did  |    name     |  budget
-----+-------------+-----------
   1 | Finance     | 1000.0000
   2 | Marketing   | 2000.0000
```

Schema Definition – Example 2

```
INSERT INTO department VALUES(1, 'Finance', 1000.0);
INSERT INTO department VALUES(2, 'Marketing', 2000.0);
```

```
INSERT INTO department VALUES(1, 'Engineering', 5000.0);
INSERT INTO department VALUES(3, 'Marketing', 3000.0);
```

```
SELECT * FROM department;
```

```
 did |     name      |   budget
-----+---------------+------------
   1 | Finance       | 1000.0000
   2 | Marketing     | 2000.0000
   1 | Engineering   | 5000.0000
   3 | Marketing     | 3000.0000
```

⚠ We have a repeated ***did*** key and a repeated ***name***

Schema Definition – Example 3

```
CREATE TABLE department(
    did INTEGER,
    name VARCHAR(80),
    budget NUMERIC(12,4),
    PRIMARY KEY(did),
    UNIQUE(name));
```

```
INSERT INTO department VALUES(1, 'Finance', 1000.0);
INSERT INTO department VALUES(2, 'Marketing', 2000.0);
```

```
INSERT INTO department VALUES(1, 'Engineering', 5000.0);
INSERT INTO department VALUES(3, 'Marketing', 3000.0);
```

⚠ The last two statements fail!

Retrieving Data

What is the result of a query?

sid	name	grade
53666	Maria	10
53688	Joseph	11
53650	Chris	12
53831	Ana	13
53832	John	14

Who are the students with grades above 11?

sid	name	grade
53650	Chris	12
53831	Ana	13
53832	John	14

What is the percent of students that finished with a grade greater that 13?

percent
20%

Queries convert data into information!

Basic Query Structure

- SQL is based on set and relational operations with certain modifications and enhancements
- The fundamental idea is that of selecting the records that satisfy a given set of criteria; SQL allows for the definition of arbitrarily complex criteria.
- A typical SQL query has the form:

```
select C1, C2, ..., Cn
from table
where condition
```

- The result of an SQL query is also a **table**

The **SELECT** clause

- The select clause lists the attributes desired in the result of a query
- **Example**: find the names of all branches in the loan relation:

```
select branch_name
from loan;
```

```
branch_name
-------------
 Downtown
 Central
 Uptown
 Downtown
 Metro
 Round Hill
 Uptown
 Downtown
 Central
```

NB: SQL names are case insensitive (i.e., you may use upper- or lower-case characters)

The **SELECT** clause (cont.)

- An asterisk in the select clause denotes "*all attributes*"

```
select *
from loan;
```

```
loan_number | branch_name |  amount
------------+-------------+-----------
 L-17       | Downtown    | 1000.0000
 L-23       | Central     | 2000.0000
 L-15       | Uptown      | 3000.0000
 L-14       | Downtown    | 4000.0000
 L-93       | Metro       | 5000.0000
 L-11       | Round Hill  | 6000.0000
 L-16       | Uptown      | 7000.0000
 L-20       | Downtown    | 8000.0000
 L-21       | Central     | 9000.0000
```

Expressions and derived fields

The **SELECT** clause (cont.)

- The **select** clause can contain arithmetic expressions involving the operations +, −, ∗, and /, as well as functions operating on constants or attributes of tuples

```
select loan_number, branch_name, amount * 1.1
from loan
```

- Returns a relation that is the same as the loan relation, except that the value of the attribute amount is multiplied by 1.1

Field and expression aliases

- SQL allows renaming tables and columns using the **AS** clause:

```
old-name AS new-name
```

- **Example**: Find the new balance of every account after subtracting 10€ in expenses

```
select account_number as acct, (balance-10) as new_bal
from account;
```

```
 acct | new_bal
----------------+----------
 A-101          | 490.0000
 A-215          | 590.0000
 A-102          | 690.0000
 A-305          | 790.0000
 A-201          | 890.0000
 A-222          | 540.0000
 A-217          | 640.0000
 A-333          | 740.0000
 A-444          | 840.0000
```

Filtering rows

The **WHERE** clause

- The **where** clause specifies conditions that the table rows must satisfy (i.e., acts as a filter)
- **Example**: Find all loan numbers of loans made at the 'Central' branch with loan amounts greater than 5000€

```
select loan_number, amount
from loan
where branch_name = 'Central'
  and amount > 5000
```

```
 loan_number |  amount
-------------+-----------
 L-21        | 9000.0000
```

The **WHERE** clause

- Comparison results can be combined using the logical connectives **and**, **or**, and **not**
- Comparisons can be applied to results of arithmetic expressions

The **WHERE** clause

- SQL includes the **between** … **and** operator
- **Example**: Find the loan number of those loans whose amounts are between 5 000€ and €10 000 (i.e., ≥ €5 000 and ≤ €10 000)

```
select loan_number
from loan
where amount between 5000 and 10000
```

```
 loan_number
-------------
 L-93
 L-11
 L-16
 L-20
 L-21
```

Handling Duplicates

The **SELECT DISTINCT** clause

- SQL allows duplicates in relations as well as in query results
- To force the elimination of duplicates, insert the keyword **distinct**
- **Example**: find the names of all branches in the loan relations, and remove the duplicates

```
SELECT DISTINCT branch_name
FROM loan;
```

```
branch_name
-------------
 Central
 Uptown
 Round Hill
 Downtown
 Metro
```

Searching and Pattern Matching

String Pattern Matching

- The operator **LIKE** uses patterns that are described using two special characters:
  - **Underscore** '_': matches any (at least one) single character.
  - **Percent** '%': matches any sequence of variable length (including the empty sequence)

Pattern	Matches
`Ana%`	Ana, Anastasia, Anacleto, Anafora, Ananás, …
`%ana%`	Banana, Cana, Panacota, Feliciana, …
`Ana_%s`	Ana Santos, Ana Martins, Ananás
`Ana %s`	Ana Santos, Ana Martins, …
`__-__-__`	XT-09-10, 20-45-XY, 10-AD-90, AB-CD-EF, …

String Pattern Matching

- **Example**: Find the street of all customers whose name starts with “Ana”

```
select customer_name, customer_street
from customer
where customer_name like 'Ana%'
```

Ana, Anastasia, Anacleto, Anafora, Ananás, …

- **Example**: Find the street of all customers whose name has “ana”

```
select customer_name, customer_street
from customer
where customer_name like '%ana%'
```

Banana, Cana, Panacota, Feliciana, …

Exercise

Exercise

A. Get the data of all accounts

B. Get the number of all accounts with a balance between 700€ and 900€

C. Get the name of all customers whose street name starts in 'C'

```
SELECT *
FROM account
```

```
SELECT account_number
FROM account
WHERE balance BETWEEN 700 AND 900
```

```
SELECT customer_name
FROM customer
WHERE customer_street LIKE 'C%'
```

Sorting Rows

Changing the **display order** of rows

- List in alphabetic order the names of all customers

```
SELECT DISTINCT customer_name
FROM customer
ORDER BY customer_name;
```

- We may specify **desc** for descending order or **asc** for ascending order (the default)

```
SELECT DISTINCT customer_name
FROM customer
ORDER BY customer_name DESC
```

```
 customer_name
---------------
 Adams
 Brown
 Cook
 Davis
 Evans
 Flores
 Gonzalez
 Iacocca
 Johnson
 King
 Lopez
 Martin
 Nguyen
 Oliver
 Parker
```

```
 customer_name
---------------
 Parker
 Oliver
 Nguyen
 Martin
 Lopez
 King
 Johnson
 Iacocca
 Gonzalez
 Flores
 Evans
 Davis
 Cook
 Brown
 Adams
```

Order in DBMS

Order-dependent Storage

- Each application may require data sorted in a different way; even distinct screens in the same application
- If records were stored ordered by a certain column, then sorting them by other order is costly

Order-independent Storage

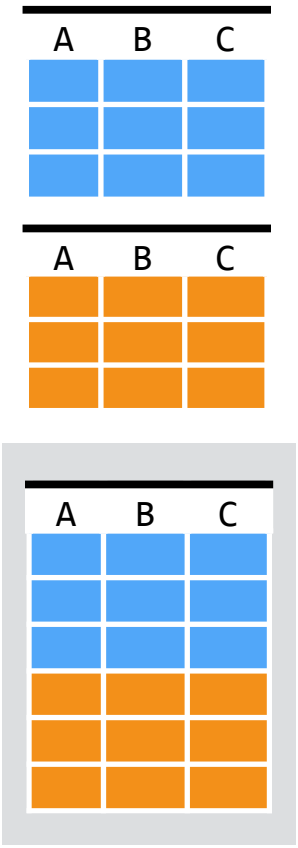
- The order by which records are organised in the storage should be irrelevant (data are as a principle independent from the application)
- DBMSs provide efficient ways to sort records in whatever way the application requires

Set Operations

Set Operations

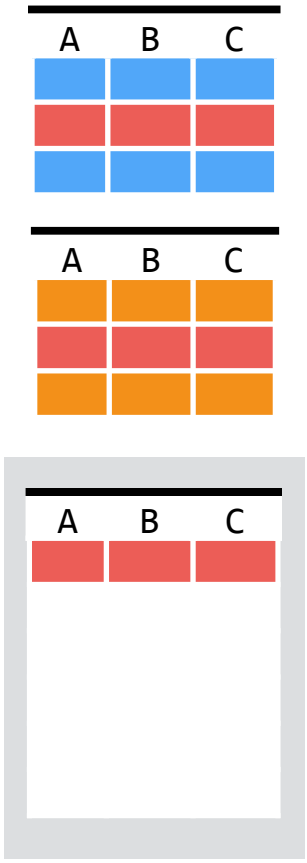
Union of tables

```
SELECT …
UNION
SELECT …
```



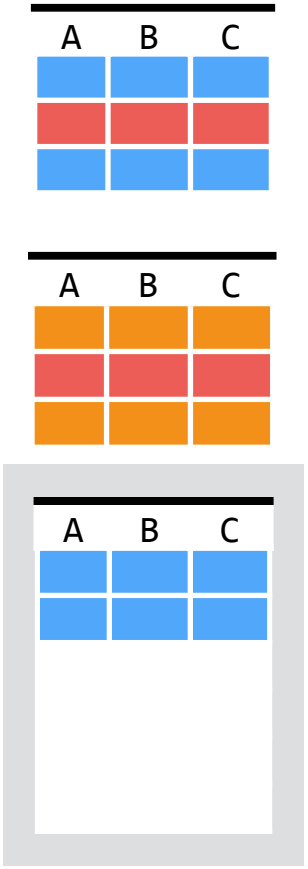
Intersection of tables

```
SELECT …
intersect
SELECT …
```



Difference of tables

```
SELECT …
except
SELECT …
```



⚠ Set operators eliminate duplicate tuples by default. The `all` clause should be specified in order to consider the duplicates.

Set Operations

- Find all customers who have a loan, an account, or both

```
(SELECT customer_name FROM depositor)
UNION
(SELECT customer_name FROM borrower)
```

- Find all customers who have both a loan and an account

```
(SELECT customer_name FROM depositor)
INTERSECT
(SELECT customer_name FROM borrower)
```

- Find all customers who have an account but no loan

```
(SELECT customer_name FROM depositor)
EXCEPT
(SELECT customer_name FROM borrower)
```

Navigating by Joining Relations

JOIN Example

List the names of all employees along their corresponding departments

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	Z
4	Duarte	W

Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca
W	Support	Cacém

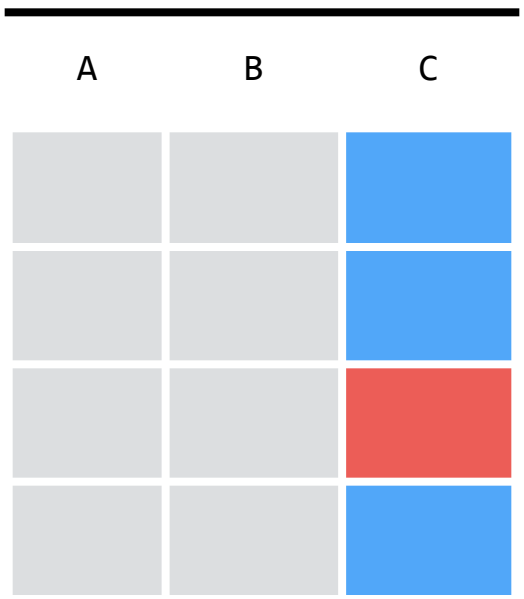
```
SELECT e.ename, d.name
FROM Employee e JOIN Department d
    ON e.did = d.did
```

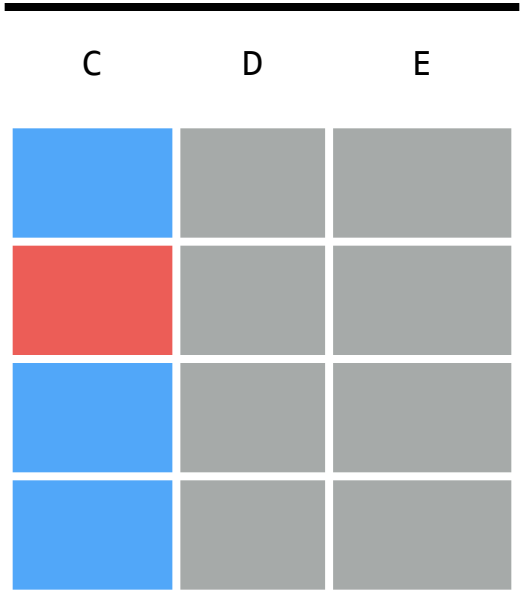
e.name	d.name
Alice	Marketing
Barbara	Sales
Carlos	Production
Duarte	Support

JOIN Operations

Join Operations

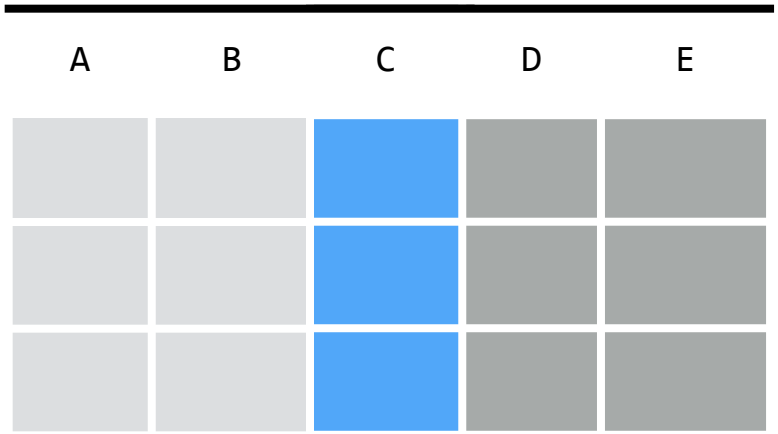
Joining two tables





Join Operations

Results on a table that ties together the records that agree on (that match) the values of a column (or columns) subject to a condition



Like extending one table with another

Behaviour of INNER JOIN

Record Mapping

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	Z
4	Duarte	W

Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca
W	Support	Cacém

```
SELECT *
FROM Employee e JOIN Department d
        ON e.did = d.did
```

e.eid	e.name	did	d.name	e.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
3	Carlos	Z	Production	Buraca
4	Duarte	W	Support	Cacém

Values are all distinct

One-to-One = **Mapping**

Record Multiplication

Employee

eid	name	did
1	Alice	X
2	Barbara	X
3	Carlos	Z
4	Duarte	W

Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca
W	Support	Cacém

```
SELECT *
FROM Employee e JOIN Department d
     ON e.did = d.did
```

e.ei	ename	did	dname	d.loc
1	Alice	X	Marketing	Damaia
2	Barbara	X	Marketing	Damaia
3	Carlos	Z	Production	Buraca
4	Duarte	W	Support	Cacém

Some values repeat

Many-to-One = **Multiplication**

Record Filtering

Employee

eid	ename	did
1	Alice	X
2	Barbara	Y
3	Carlos	T
4	Duarte	W

Department

did	dname	loc
X	Marketing	Damaia
Y	Sales	Amadora
W	Support	Cacém

```
SELECT *
FROM Employee e JOIN Department d
     ON e.did = d.did
```

Some values don't have a match

e.eid	e.ename	did	d.dname	e.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
4	Duarte	W	Production	Buraca

Zero-to-One = **Filter**

Behaviour of OUTTER JOIN

LEFT OUTER JOIN

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	T
4	Duarte	W

Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca
W	Support	Cacém

```
SELECT e.name, d.name
FROM Employee e LEFT OUTER JOIN Department d
    ON d.did = d.did
```

e.ei	e.name	did	d.name	d.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
3	Carlos	T	NULL	NULL
4	Duarte	W	Support	Buraca

Blank line with NULL values

One output row for every row of the table on the **LEFT**

RIGHT OUTER JOIN

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	T
4	Duarte	W

Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca
W	Support	Cacém

```
SELECT e.name, d.name
FROM Employee e RIGHT OUTER JOIN Department d
    ON d.did = d.did
```

e.ei	e.name	did	d.name	e.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
NULL	NULL	Z	Production	Buraca
4	Duarte	W	Support	Buraca

Blank line with NULL values

One output row for every row of the table on the **RIGHT**

FULL OUTER JOIN

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	T
4	Duarte	W

Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca
W	Support	Cacém

```
SELECT e.name, d.name
FROM Employee e FULL OUTER JOIN Department d
     ON d.did = d.did
```

e.eid	e.name	did	d.name	e.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
3	Carlos	T	NULL	NULL
	NULL	Z	Production	Buraca
4	Duarte	W	Support	Cacém

One output row for every row on **EACH** table

Querying the Bank Database with a Join

Example

Find the names of the depositors with accounts with more than 750 € in balance

Step 1: Find all Accounts with balance > 750€

```
SELECT *
FROM account
WHERE balance > 750
ORDER BY balance;
```

```
 account_number | branch_name | balance
----------------+-------------+----------
 A-305          | Round Hill  | 800.0000
 A-444          | Downtown    | 850.0000
 A-201          | Uptown      | 900.0000
```

Step 2: Find the depositors of those accounts

```
SELECT *
FROM depositor
ORDER BY account_number;
```

```
 customer_name | account_number
---------------+----------------
 Cook          | A-101
 Johnson       | A-101
 Cook          | A-102
 Johnson       | A-201
 Brown         | A-215
 Iacocca       | A-217
 Evans         | A-222
 Flores        | A-305
 Oliver        | A-333
 Brown         | A-444
```

Example

*Find the **names of the customers** with accounts having more than 750 € in balance*

Joining Account and Depositor on the attribute 'account_number'

```
SELECT *
FROM account a JOIN depositor d
  ON a.account_number = d.account_number
WHERE balance > 750;
```

We are selecting all attributes

```
 account_number | branch_name | balance  | customer_name
----------------+-------------+----------+---------------
 A-305          | Round Hill  | 800.0000 | Flores
 A-201          | Uptown      | 900.0000 | Johnson
 A-444          | Downtown    | 850.0000 | Brown
```

Example

*Find the **names** **of the customers** with accounts having more than 750 € in balance*

```
SELECT customer_name
FROM account a JOIN depositor d
  ON a.account_number = d.account_number
WHERE balance > 750;
```

Still joining tables but only selection the customer_name

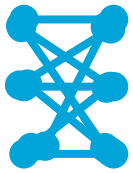
```
 customer_name
---------------
 Flores
 Johnson
 Brown
```

Cartesian Product

Cartesian Product

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	Z



Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca

```
SELECT e.name, d.name
FROM Employee e, Department d
```

The FROM clause lists the tables involved in the query

e.ei	e.name	e.did	d.di	d.name	e.loc
1	Alice	X	X	Marketing	Damaia
2	Barbara	Y	X	Marketing	Damaia
3	Carlos	T	X	Marketing	Damaia
1	Alice	X	Y	Sales	Amadora
2	Barbara	Y	Y	Sales	Amadora
3	Carlos	T	Y	Sales	Amadora
1	Alice	X	Z	Production	Buraca
2	Barbara	Y	Z	Production	Buraca
3	Carlos	T	Z	Production	Buraca

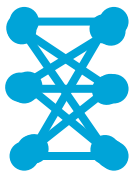
Employee X Department

Outputs **all combinations of rows,** irrespective of the contents

Cartesian Product with a Filter

Employee

eid	name	did
1	Alice	X
2	Barbara	Y
3	Carlos	Z



Department

did	name	loc
X	Marketing	Damaia
Y	Sales	Amadora
Z	Production	Buraca

```
SELECT e.name, d.name
FROM Employee e, Department d
WHERE e.did = d.did
```

Query has a filter

e.eid	e.name	did	d.name	e.loc
1	Alice	X	Marketing	Damaia
2	Barbara	Y	Sales	Amadora
3	Carlos	Z	Production	Buraca

Rows that **do not match** the filter **are dropped**

```
SELECT *
FROM Employee e JOIN Department d
       ON e.did = d.did
```

The Cartesian product

depositor × account

```
SELECT *
FROM depositor d, account a
```

customer_name	account_number	account_number	branch_name	balance
Johnson	A-101	A-101	Downtown	500.0000
Johnson	A-101	A-215	Metro	600.0000
Johnson	A-101	A-102	Uptown	700.0000
Johnson	A-101	A-305	Round Hill	800.0000
Johnson	A-101	A-201	Uptown	900.0000
Johnson	A-101	A-222	Central	550.0000
Johnson	A-101	A-217	University	650.0000
Johnson	A-101	A-333	Central	750.0000
Johnson	A-101	A-444	Downtown	850.0000
Brown	A-215	A-101	Downtown	500.0000
Brown	A-215	A-215	Metro	600.0000
Brown	A-215	A-102	Uptown	700.0000
Brown	A-215	A-305	Round Hill	800.0000
Brown				900.0000
Brown				550.0000
Brown				650.0000
Brown				750.0000
Brown	A-215	A-444	Downtown	850.0000

⚠ What happens if we filter the rows on d.account_number = a.account number?

Cartesian product with a filter

- What happens when we filter a Cartesian product?

```
SELECT *
FROM depositor d, account a
WHERE d.account_number = a.account_number;
```

```
customer_name | account_number | account_number | branch_name | balance
---------------+----------------+----------------+-------------+----------
 Johnson       | A-101          | A-101          | Downtown    | 500.0000
 Brown         | A-215          | A-215          | Metro       | 600.0000
 Cook          | A-102          | A-102          | Uptown      | 700.0000
 Cook          | A-101          | A-101          | Downtown    | 500.0000
 Flores        | A-305          | A-305          | Round Hill  | 800.0000
 Johnson       | A-201          | A-201          | Uptown      | 900.0000
 Iacocca       | A-217          | A-217          | University  | 650.0000
 Ev                                                                | 550.0000
 O                                                                 | 750.0000
 Br                                                                | 850.0000
(10
```

```
SELECT *
FROM account a JOIN depositor d
  ON a.account_number = d.account_number
```

Example

Find the names of the cities of the customers with accounts having more than 750 € in balance

```
SELECT customer_city
FROM account a, depositor d, customer c
WHERE a.account_number = d.account_number
  AND c.customer_name = d.customer_name
  AND balance > 750;
```

```
customer_city
---------------
 Oporto
 Braga
 Cascais
```