



Swiss Institute of
Bioinformatics

SIB TRAINING

SQL for data science

Florent Tassy & Dillenn Terumalai

10th of April 2025, Remote



Who are we?

- » OncoBench
- » GenBench
- » SPSP
- » SVIP
- » SwissGenVar
- » ERPA
- » QCMD
- » SendCrypt



Who are you?

- »» PhD
- »» Postdoc
- »» Professional
- »» Student
- »» Developer
- »» Bioinformatician
- »» Other



Material and content

SQL for data science GIT repository

<https://tinyurl.com/sql-data-science>

Online exercises

<https://digistorm.app/p/3672689>



Pre-requisites

Beekeeper Studio Community Edition

<https://tinyurl.com/2p97hzc9>

Schedule for today

1. Introduction and filters

09:00 – 10:45

2. Aggregate functions

11:00 – 12:30

3. Subqueries and joins

13:30 – 15:15

4. Advanced concepts

15:30 – 17:00

Task:
Store some patient information



First name: Clementine

Last name: Bauch

Birth date: 13.07.1990

Height: 165 cm

Table

Headers or columns

Rows

First name	Last name	Birth date	Height



First name	Last name	Birth date	Height
Clementine	Bauch	13.07.1990	165



First name: Erwin

Last name: Howell

Birth date: 21.02.1985

Height: 183 cm



First name	Last name	Birth date	Height
Clementine	Bauch	13.07.1990	165
Erwin	Howell	21.02.1985	183



First name: Kurtis
Last name: Jackson
Birth date: 13.03.1991
Height: 190 cm



First name	Last name	Birth date	Height
Clementine	Bauch	13.07.1990	165
Erwin	Howell	21.02.1985	183
Kurtis	Jackson	13.03.1991	190



Types

» Text

» Integer

» Date



First name Text	Last name Text	Birth date Date	Height Integer
Clementine	Bauch	13.07.1990	165
Erwin	Howell	21.02.1985	183
Kurtis	Jackson	13.03.1991	190



First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
Clementine	Bauch	13.07.1990	165	TRUE	20.3
Erwin	Howell	21.02.1985	183	FALSE	22.2
Kurtis	Jackson	13.03.1991	190	TRUE	21.5



Types

» Text

» Integer

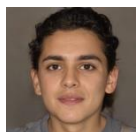
» Date

» Boolean

» Decimal



First name: Clementine
Last name: Rodriguez
Birth date: 13.03.1991
Height: 171 cm
Consent: FALSE
BMI: 21.1

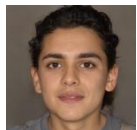


First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
Clementine	Bauch	13.07.1990	165	TRUE	20.3
Erwin	Howell	21.02.1985	183	FALSE	22.2
Kurtis	Jackson	13.03.1991	190	TRUE	21.5
Clementine	Rodriguez	13.03.1991	171	FALSE	21.1



ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3
2	Erwin	Howell	21.02.1985	183	FALSE	22.2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1

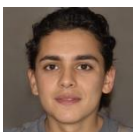
Primary KEY
or PK (UNIQUE)



ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3
2	Erwin	Howell	21.02.1985	183	FALSE	22.2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1



First name: Mark
Last name: Miller
Birth date: 14.06.1990
Height: 192 cm
Consent: ?
BMI: 22.3



ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3
2	Erwin	Howell	21.02.1985	183	FALSE	22.2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1
5	Mark	Miller	14.06.1990	192	<i>NULL</i>	22.3

NULL value

ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal	Country Text
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3	Germany
2	Erwin	Howell	21.02.1985	183	FALSE	22.2	United Kingdom
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5	United Kingdom
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1	Spain
5	Mark	Miller	14.06.1990	192	<i>NULL</i>	22.3	Germany

ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal	Country Text
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3	Germany
2	Erwin	Howell	21.02.1985	183	FALSE	22.2	United Kingdom
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5	United Kingdom
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1	Spain
5	Mark	Miller	14.06.1990	192	NULL	22.3	Germany

Table: patients

ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3
2	Erwin	Howell	21.02.1985	183	FALSE	22.2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1
5	Mark	Miller	14.06.1990	192	NULL	22.3

Table: countries

ID Integer	Country Text
1	Germany
2	United Kingdom
3	Spain

Table: patients

Foreign KEY
or FK

Table: countries

ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal	Country ID Integer
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3	1
2	Erwin	Howell	21.02.1985	183	FALSE	22.2	2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5	2
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1	3
5	Mark	Miller	14.06.1990	192	NULL	22.3	1

Relations

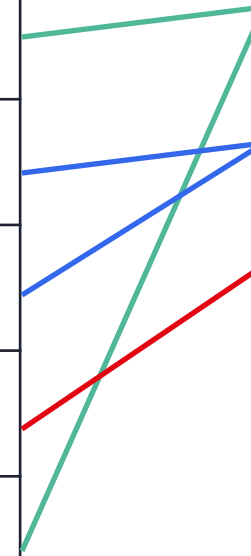
ID Integer	Country Text
1	Germany
2	United Kingdom
3	Spain



Relational database

ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal	Country ID Integer
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3	1
2	Erwin	Howell	21.02.1985	183	FALSE	22.2	2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5	2
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1	3
5	Mark	Miller	14.06.1990	192	NULL	22.3	1

ID Integer	Country Text
1	Germany
2	United Kingdom
3	Spain





Relations

- ⌘ 1:1 or one-to-one
- ⌘ 1:N or one-to many
- ⌘ N:M or many-to-many



1:1 or one-to-one



ONE patient has ONE medical record



1:N or one-to-many



ONE doctor has MANY patients



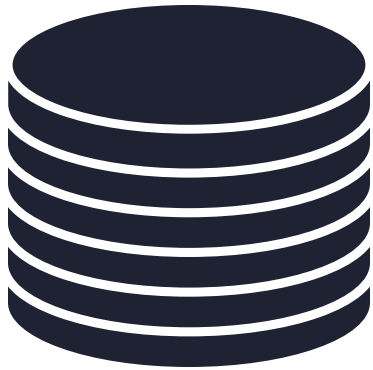
N:M or many-to-many



MANY doctors speak MANY languages



Interact with a DBMS



DBMS or Database
Management
System



User



DBMS main features

- ⌘ Efficient storage
- ⌘ Easily queryable
- ⌘ Largely supported
- ⌘ Easily integrable

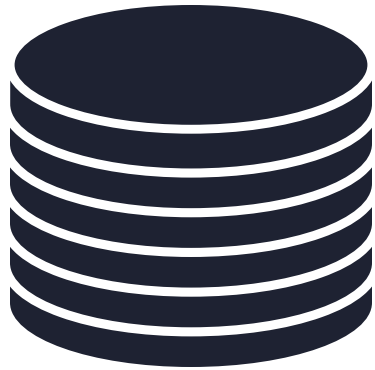


DBMS examples

- » MySQL
- » MariaDB
- » PostgreSQL
- » **SQLite**
- » Microsoft SQL Server
- » Oracle Database



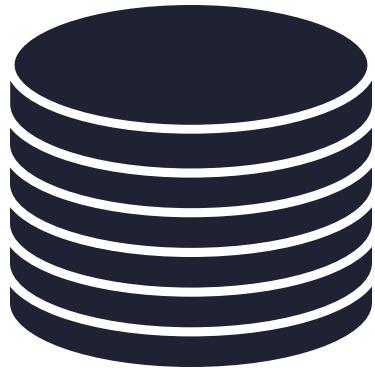
Interact with a DBMS



Hey, do you have a patient
called "Mark Miller"?



Interact with a DBMS



Let me check!



Table: patients

ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3
2	Erwin	Howell	21.02.1985	183	FALSE	22.2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1
5	Mark	Miller	14.06.1990	192	NULL	22.3

Table: countries

ID Integer	Country Text
1	Germany
2	United Kingdom
3	Spain

Table: patients

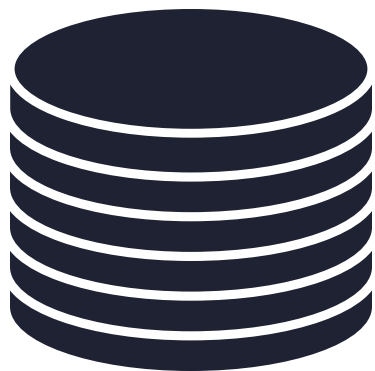
ID Integer	First name Text	Last name Text	Birth date Date	Height Integer	Consent Boolean	BMI Decimal
1	Clementine	Bauch	13.07.1990	165	TRUE	20.3
2	Erwin	Howell	21.02.1985	183	FALSE	22.2
3	Kurtis	Jackson	13.03.1991	190	TRUE	21.5
4	Clementine	Rodriguez	13.03.1991	171	FALSE	21.1
5	Mark	Miller	14.06.1990	192	NULL	22.3

Table: countries

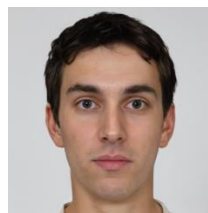
ID Integer	Country Text
1	Germany
2	United Kingdom
3	Spain



Interact with a DBMS



Yes, I have this patient:

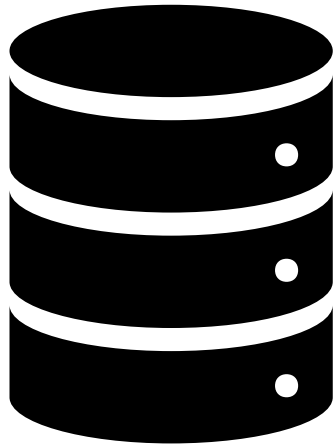


ID: 5
First name: Mark
Last name: Miller
Birth date:
14.06.1990
Height: 192
Consent: NULL
Country ID: 1

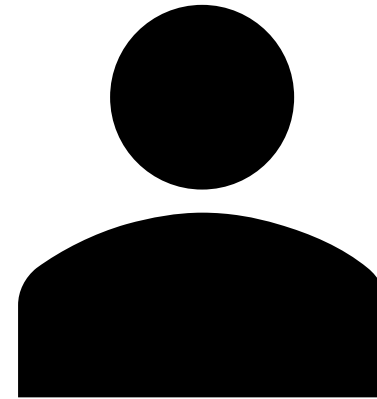


What is SQL?

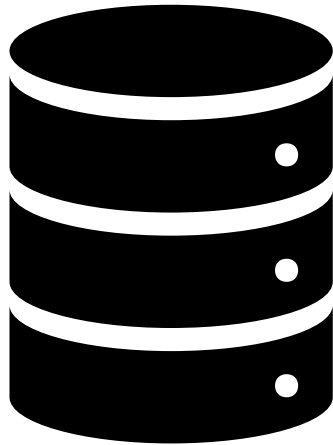
SQL or Structured Query Language



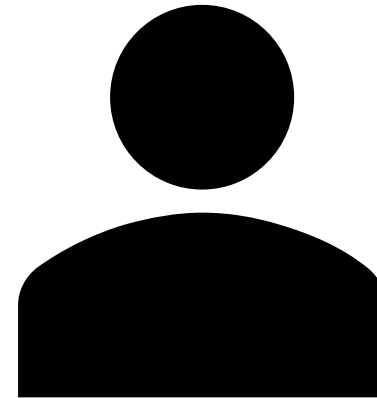
?



SQL or Structured Query Language

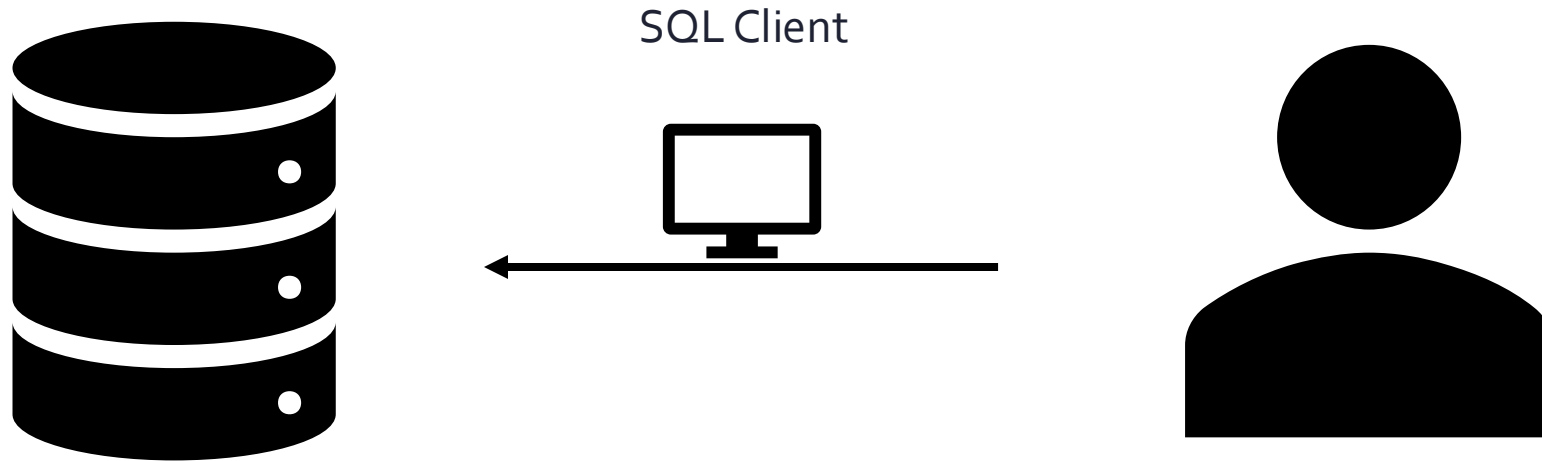


?



```
SELECT  
  *  
FROM  
  patients  
WHERE  
  first_name = 'Mark'  
AND  
  last_name = 'Miller';
```

SQL or Structured Query Language





SQL Clients

- » DBeaver
- » TablePlus
- » DataGrip
- » **Beekeeper Studio**
- » Etc.

The NGS database



<https://tinyurl.com/msr2fg7m>



NGS database

- Sample database
- Intends to demonstrate queries examples
- BSD license



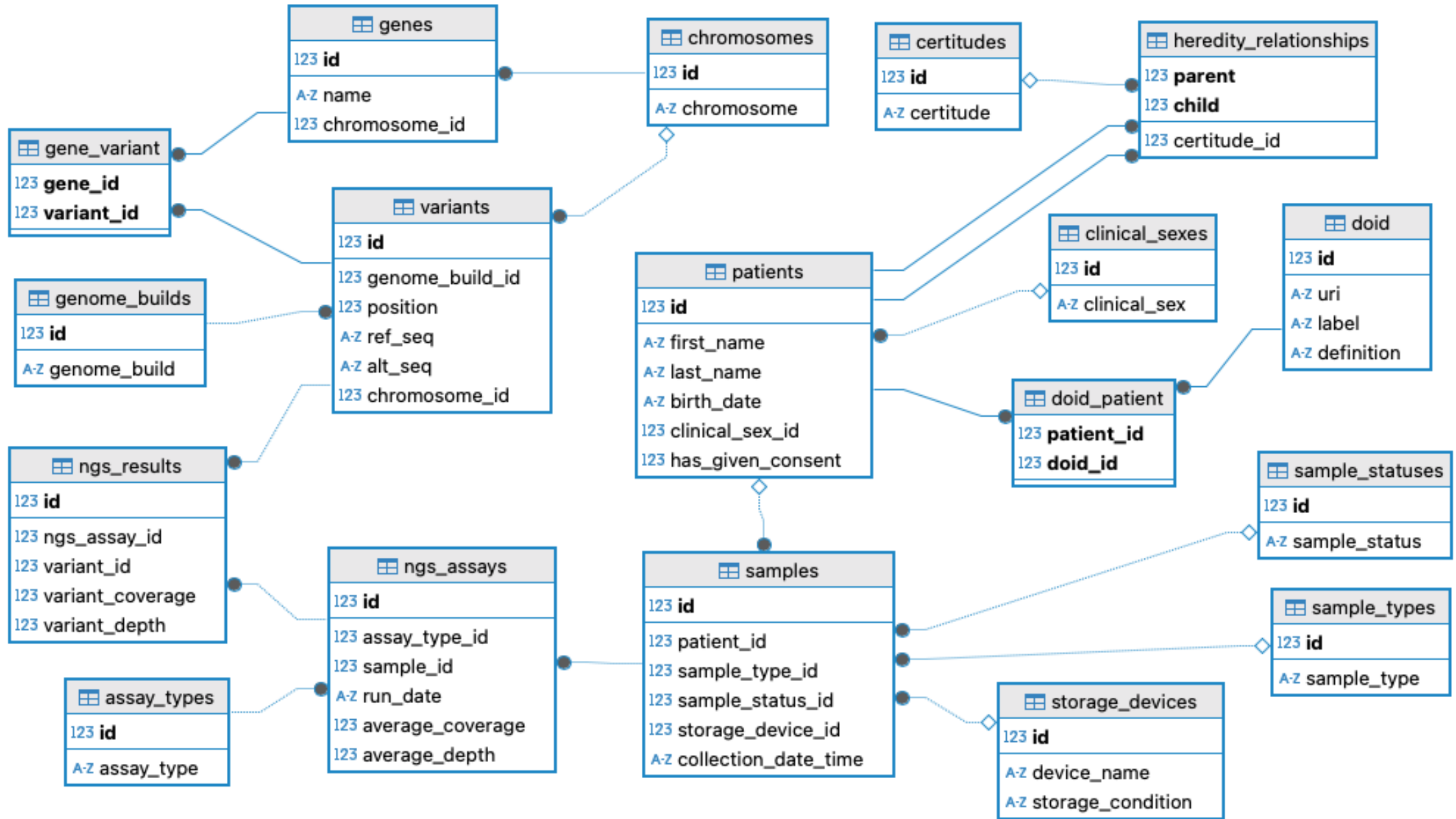
NGS database content

Represents a medical information database

- Patients, with their relationships, clinical sexes and diseases
- Samples, with their types, statuses and storage
- Variants, linked to their genome builds, genes and chromosomes
- NGS assays, made on samples, with NGS results containing variants



NGS database model



Open `ngs_database` with **Beekeeper**

04. Retrieve data with SELECT

The **SELECT** statement

04. Retrieve data with SELECT

SELECT

column1, column2, ...

FROM

table_name;

04. Retrieve data with SELECT

SELECT

first_name, last_name

FROM

patients;

04. Retrieve data with SELECT

SELECT

*

FROM

patients;



04. Retrieve data with SELECT

Format your SQL queries



04. Retrieve data with SELECT

Questions

Filtering with SQL



05. Filtering with SQL

SELECT

column1, column2, ...

FROM

table_name

WHERE

criteria;



05. Filtering with SQL

SELECT

*

FROM

ngs_results

WHERE

variant_coverage > 50;



05. Filtering with SQL

Primary comparison operators



05. Filtering with SQL

Operator	Description	Example
"="	Equal	age = 30
"<>" or "!="	Not equal	age != 30
">"	Greater than	age > 30
"<"	Less than	age < 30
">="	Greater than or equal	age >= 30
"<="	Less than or equal	age <= 30



05. Filtering with SQL

Questions

LIMIT



o6. LIMIT

SELECT

column1, column2, ...

FROM

table_name

WHERE

criteria

LIMIT

n;



o6. LIMIT

SELECT

*

FROM

genes

WHERE

chromosome_id = 1

LIMIT

10;

Questions



07. IN, OR, AND and NOT

IN, OR, AND and NOT



07. IN, OR, AND and NOT

SELECT

column1, column2, ...

FROM

table_name

WHERE

column1 **IN** (value1, value2, ...);



07. IN, OR, AND and NOT

SELECT

COUNT(*)

FROM

genes

WHERE

chromosome_id **IN** (1, 2, 3);



07. IN, OR, AND and NOT

SELECT

column1, column2, ...

FROM

table_name

WHERE

condition1

OR condition2

OR ...;



07. IN, OR, AND and NOT

SELECT

*

FROM

patients

WHERE

first_name = 'Marc'

OR first_name = 'Mark';



07. IN, OR, AND and NOT

SELECT

column1, column2, ...

FROM

table_name

WHERE

condition1

AND condition2

AND ...;



07. IN, OR, AND and NOT

SELECT

*

FROM

patients

WHERE

first_name = 'Dylan'

AND last_name = 'Barton';



07. IN, OR, AND and NOT

SELECT

column1, column2, ...

FROM

table_name

WHERE

NOT condition;

Questions

LIKE operator



o8. LIKE operator

SELECT

column1, column2, ...

FROM

table_name

WHERE

column1 **LIKE** pattern;



o8. LIKE operator

SELECT

*

FROM

patients

WHERE

first_name **LIKE** 'Z%';



o8. LIKE operator

SELECT

*

FROM

patients

WHERE

first_name **LIKE** '____';

Questions

Sorting your results with **ORDER BY**

09. Sorting your results with ORDER BY

Never assume that a query result is
sorted

09. Sorting your results with ORDER BY

Example of risky assertions:

- ⚠ Primary keys are in ascending order
- ⚠ Next primary key will be highest one + 1
- ⚠ If primary key n exists, then there is also one of value $n-1$

09. Sorting your results with ORDER BY

...unless explicitly defined with **ORDER**
BY

09. Sorting your results with ORDER BY

```
SELECT  
    column1, column2  
FROM  
    table_name  
ORDER BY  
    column1;
```

09. Sorting your results with ORDER BY

SELECT

id, run_date, average_coverage

FROM

ngs_assays

ORDER BY

average_coverage; -- number column

09. Sorting your results with ORDER BY

```
SELECT
    id, run_date, average_coverage
FROM
    ngs_assays
ORDER BY
    run_date; -- datetime column
```


09. Sorting your results with ORDER BY

SELECT

id, first_name, last_name

FROM

patients

ORDER BY

last_name; -- text column

09. Sorting your results with ORDER BY

SELECT

*

FROM

samples

ORDER BY

storage_device_id; -- NULL values

09. Sorting your results with ORDER BY

```
SELECT
```

```
    *
```

```
FROM
```

```
    patients
```

```
ORDER BY
```

```
    last_name DESC;
```

```
-- Descending order
```

```
SELECT
```

```
    *
```

```
FROM
```

```
    patients
```

```
ORDER BY
```

```
    last_name ASC;
```

```
-- Ascending order
```

09. Sorting your results with ORDER BY

SELECT

id, run_date, average_coverage, average_depth

FROM

ngs_assays

ORDER BY

average_coverage **ASC**,

average_depth **DESC**;

-- multiple columns

09. Sorting your results with ORDER BY

The column in **ORDER BY** must be
part of the **SELECT**

09. Sorting your results with ORDER BY

Questions

10. AVG, COUNT, MIN, MAX and SUM

**AVG, COUNT, MIN, MAX
and SUM**

10. AVG, COUNT, MIN, MAX and SUM

Also known as **AGGREGATE
FUNCTIONS**



10. AVG, COUNT, MIN, MAX and SUM

Aims to "aggregate" or "summarize"
several rows in one



10. AVG, COUNT, MIN, MAX and SUM

Difficult to mix with non aggregated selections

10. AVG, COUNT, MIN, MAX and SUM

-- Average value of a numeric column

SELECT

AVG(variant_depth)

FROM

ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

-- Nicer result with column alias

SELECT

 AVG(variant_depth) AS "Average variant depth"

FROM

 ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

--  Mixing with non aggregate

SELECT

id,

AVG(variant_depth) AS "Average variant depth"

FROM

ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

-- Aggregate multiple columns at once

SELECT

AVG(variant_depth) AS "Average variant depth",

AVG(variant_coverage) AS "Average variant coverage"

FROM

ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

Check DBMS **documentation to know
how the aggregate functions work with
NULL values**

10. AVG, COUNT, MIN, MAX and SUM

*The avg() function returns the average value of all non-NULL X within a group.
(...) The result of avg() is NULL if and only if there are no non-NULL inputs.*

10. AVG, COUNT, MIN, MAX and SUM

-- Lowest and highest values of a column

SELECT

MIN(variant_depth) AS "Lowest variant depth",
MIN(variant_coverage) AS "Lowest variant coverage",
MAX(variant_depth) AS "Highest variant depth",
MAX(variant_coverage) AS "Highest variant coverage"

FROM

ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

-- Sum column values

SELECT

SUM(has_given_consent) AS "Consenting patients"

FROM

ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

-- Limit the number of decimals with ROUND

SELECT

ROUND(AVG(variant_depth), 2) AS "Average variant
depth",

ROUND(AVG(variant_coverage), 2) AS "Average
variant coverage"

FROM

 ngs_results;

10. AVG, COUNT, MIN, MAX and SUM

-- A good old friend: count the number of rows

SELECT

COUNT(*) as "Number of patients"

FROM

patients;



10. AVG, COUNT, MIN, MAX and SUM

Questions

11. Grouping your results with GROUP BY

Grouping your results with GROUP BY

11. Grouping your results with GROUP BY

Allows using aggregate functions at a finer level



11. Grouping your results with GROUP BY

-- How many samples of each type do we have?

SELECT

sample_type_id,
count(*)

FROM

samples

GROUP BY

sample_type_id;



11. Grouping your results with GROUP BY

-- How many available (status id 1) samples of each type "Blood" (id 1), "Saliva" (id 2) and "Hair" (id 4) do we have? Display the results with the most frequent on top.

SELECT

sample_type_id,
COUNT(*)

FROM

samples

WHERE

sample_type_id IN (1, 2, 4)
AND sample_status_id = 1

GROUP BY

sample_type_id

ORDER BY

COUNT(*) DESC;



11. Grouping your results with GROUP BY

-- GROUP BY several columns to create sub groups

SELECT

sample_type_id,
sample_status_id,
COUNT(*)

FROM

samples

WHERE

sample_type_id IN (1, 5)

GROUP BY

sample_type_id,
sample_status_id

ORDER BY

sample_type_id;



11. Grouping your results with GROUP BY

The query golden rule for mixing aggregate and non aggregate selections: if it is *not* an **aggregate function**, then it should be in the **GROUP BY**



11. Grouping your results with GROUP BY

-- Filter on aggregate function results with HAVING

SELECT

sample_type_id,
COUNT(*) AS "Nb available samples"

FROM

samples

WHERE

sample_status_id = 1

GROUP BY

sample_type_id

HAVING

COUNT(*) > 60

ORDER BY

sample_type_id;



11. Grouping your results with GROUP BY

-- Remove duplicate rows with DISTINCT

-- How many patients have *at least one* disease?

SELECT

COUNT(**DISTINCT** patient_id)

FROM

doid_patient;



11. Grouping your results with GROUP BY

Questions

Subqueries



12. Subqueries

```
SELECT
    column1, column2, ...
FROM
    table_name
WHERE
    column1 IN (
        SELECT
            column_name
        FROM
            table_name
        WHERE
            condition
    );
```




12. Subqueries

```
SELECT
    first_name,
    last_name
FROM
    patients
WHERE
    clinical_sex_id IN (
        SELECT
            id
        FROM
            clinical_sexes
        WHERE
            clinical_sex IN ('UNKNOWN', 'UNDETERMINED')
    );
```

Questions



13. Introduction to JOINS

Introduction to **JOINS**



13. Introduction to JOINS

id	first_name	last_name	clinical_sex_id	...
1	Kathryne	Fay	1	...
2	Izabella	Harris	1	...
...

id	clinical_sex
1	MALE
2	FEMALE
...	...



13. Introduction to JOINS

id	first_name	last_name	clinical_sex_id	...
1	Kathryne	Fay	1	...
2	Izabella	Harris	1	...
...

id	clinical_sex
1	MALE
2	FEMALE
...	...

SELECT

patients.first_name,
patients.last_name,
clinical_sexes.clinical_sex

FROM

patients
JOIN clinical_sexes ON
patients.clinical_sex_id = clinical_sexes.id;

first_name	last_name	clinical_sex
Kathryne	Fay	MALE
Izabella	Harris	MALE
...



13. Introduction to JOINS

SELECT

patients.first_name,
patients.last_name,
clinical_sexes.clinical_sex

FROM

patients
JOIN clinical_sexes ON patients.clinical_sex_id = clinical_sexes.id;

=

SELECT

p.first_name,
p.last_name,
cs.clinical_sex

FROM

patients p
JOIN clinical_sexes cs ON p.clinical_sex_id = cs.id;



13. Introduction to JOINS

-- Concatenating strings: the || operator

```
SELECT 'Hello' || ' ' || 'World';
```

=> 'Hello World'



13. Introduction to JOINS

-- Concatenating NULL values does not work

```
SELECT 'Hello' || NULL || 'World';
```

⇒ NULL

-- Concatenating NULL values: the *coalesce* function

```
SELECT COALESCE(NULL, 'Hello', 'World');
```

⇒ Hello



13. Introduction to JOINS

We want to create a report of all our samples:

Sample ID, storage_device_id from **samples**,

Sample type from **samples_types**,

Patient first name and last name from patients,

... in 3 columns: "Patient full name", "Sample ID", "Sample information", the lastest following a template "Type: {sample_type}, storage: {storage_device_id or '-' if not stored}, "



13. Introduction to JOINS

SELECT

```
p.first_name || ' ' || p.last_name AS "Patient full name",  
s.id as "Sample ID",  
'Type: ' || st.sample_type || ', Storage: ' || COALESCE(s.storage_device_id, '-  
' ) AS "Sample information"
```

FROM

samples s

JOIN sample_types st **ON** s.sample_type_id = st.id

JOIN patients p **ON** s.patient_id = p.id;



13. Introduction to JOINS

--  Implicit join

SELECT

s.id,
st.sample_type

FROM

samples s,
sample_types st

WHERE

s.sample_type_id = st.id;



13. Introduction to JOINS

Questions



14. CROSS JOINS

CROSS JOINS



14. CROSS JOINS

Returns the Cartesian product of the tables to join.

Rare use cases.

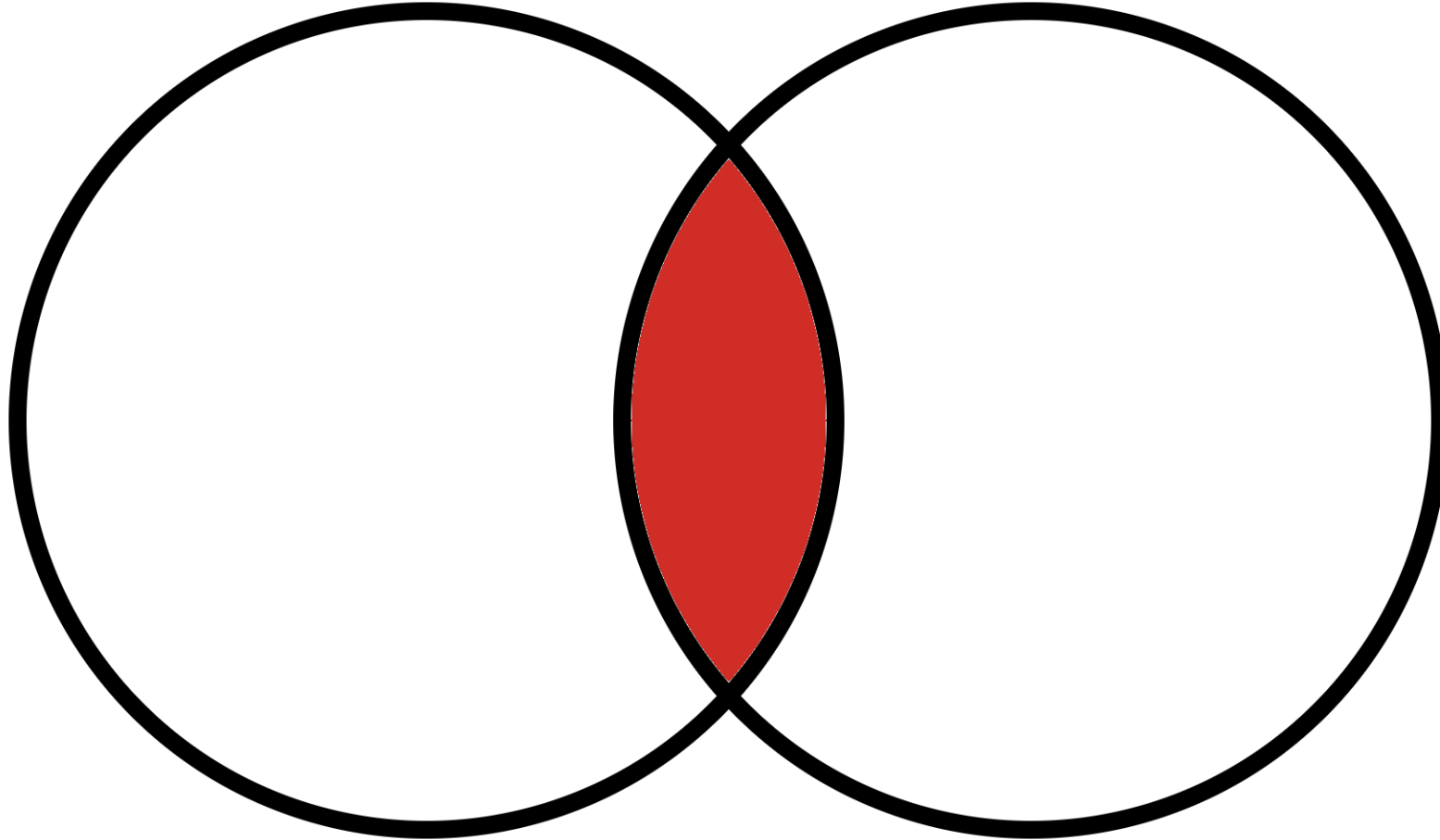
If needed, refer to the training repository.



15. INNER JOINS

INNER JOINS

15. INNER JOINS





15. INNER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4



15. INNER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4

SELECT

u.*,
c.contact_id,
c.contact_info

FROM

users u
INNER JOIN contacts c
ON u.user_id = c.user_id;

user_id	username	contact_id	contact_info
1	jdoe	1	<u>jdoe@mail.com</u>
1	jdoe	2	+41 71 345 32 65
3	fdupont	3	<u>fdupont@mail.com</u>



15. INNER JOINS

Questions

16. LEFT, RIGHT and OUTER JOINS

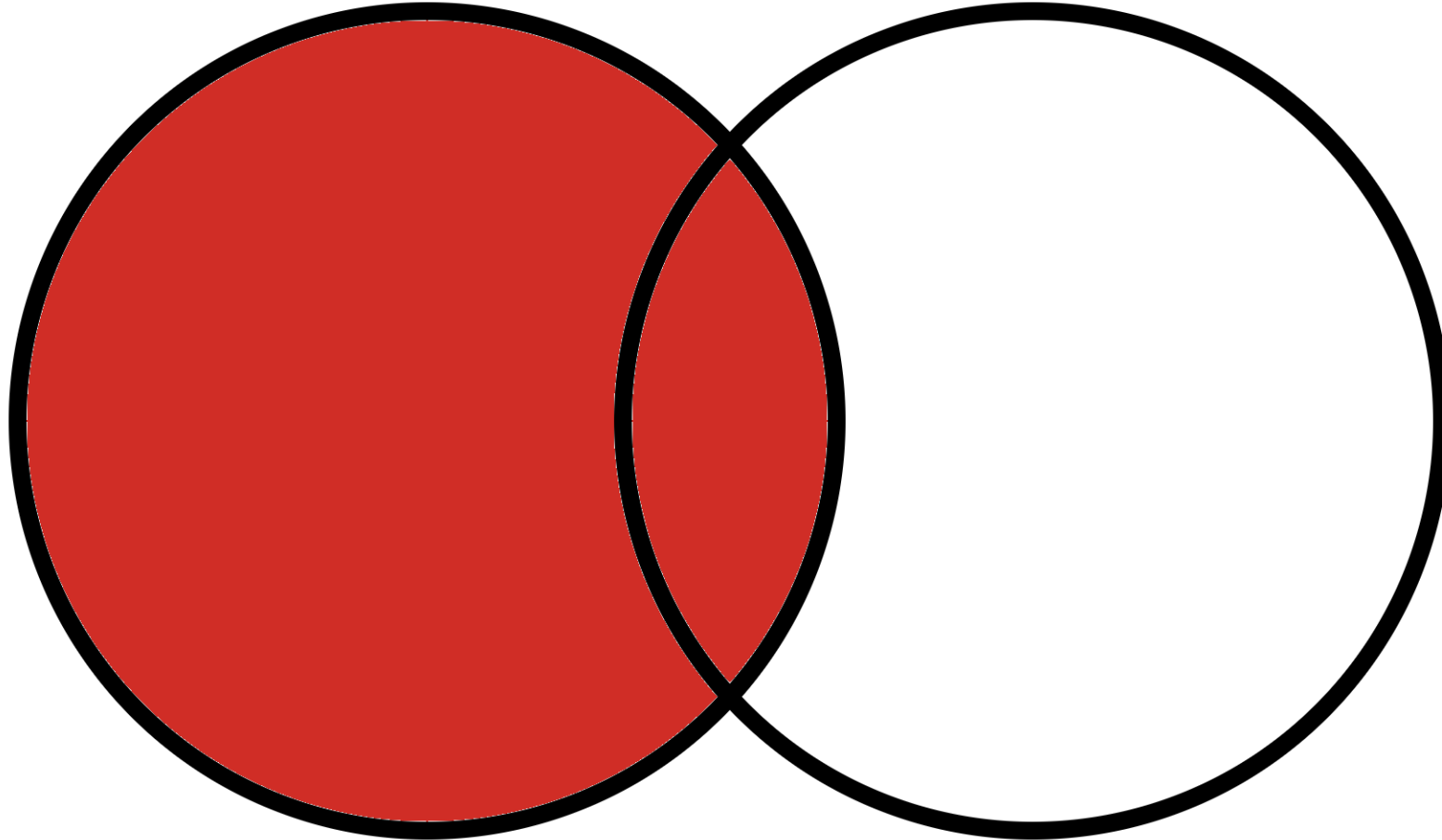
LEFT, RIGHT and OUTER JOINS

16. LEFT, RIGHT and OUTER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4

16. LEFT, RIGHT and OUTER JOINS



16. LEFT, RIGHT and OUTER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4

SELECT

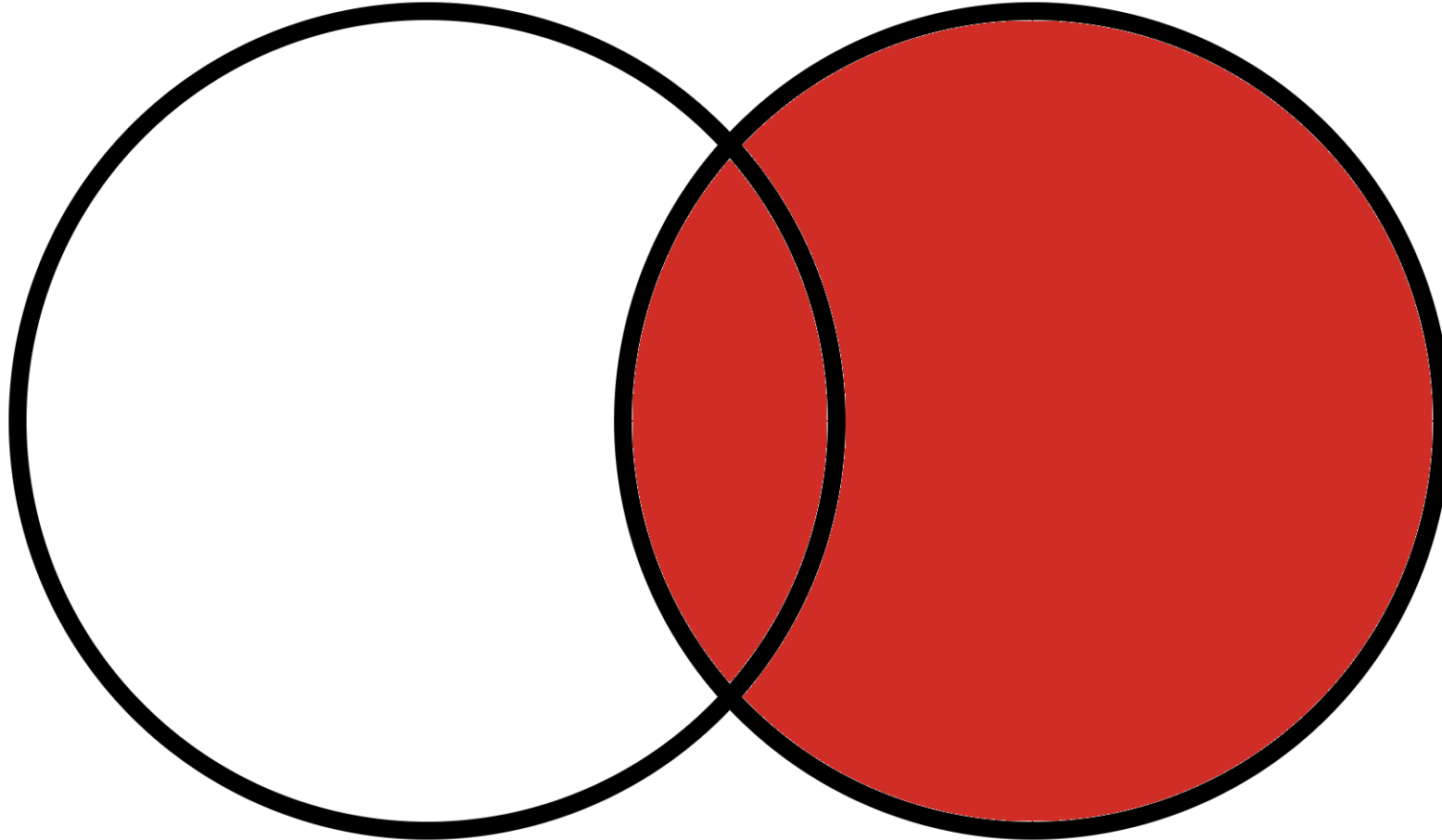
u.*,
c.contact_id,
c.contact_info

FROM

users u
LEFT JOIN contacts c
ON u.user_id = c.user_id;

user_id	username	contact_id	contact_info
1	jdoe	1	<u>jdoe@mail.com</u>
1	jdoe	2	+41 71 345 32 65
2	userY		
3	fdupont	3	<u>fdupont@mail.com</u>

16. LEFT, RIGHT and OUTER JOINS



16. LEFT, RIGHT and OUTER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4

SELECT

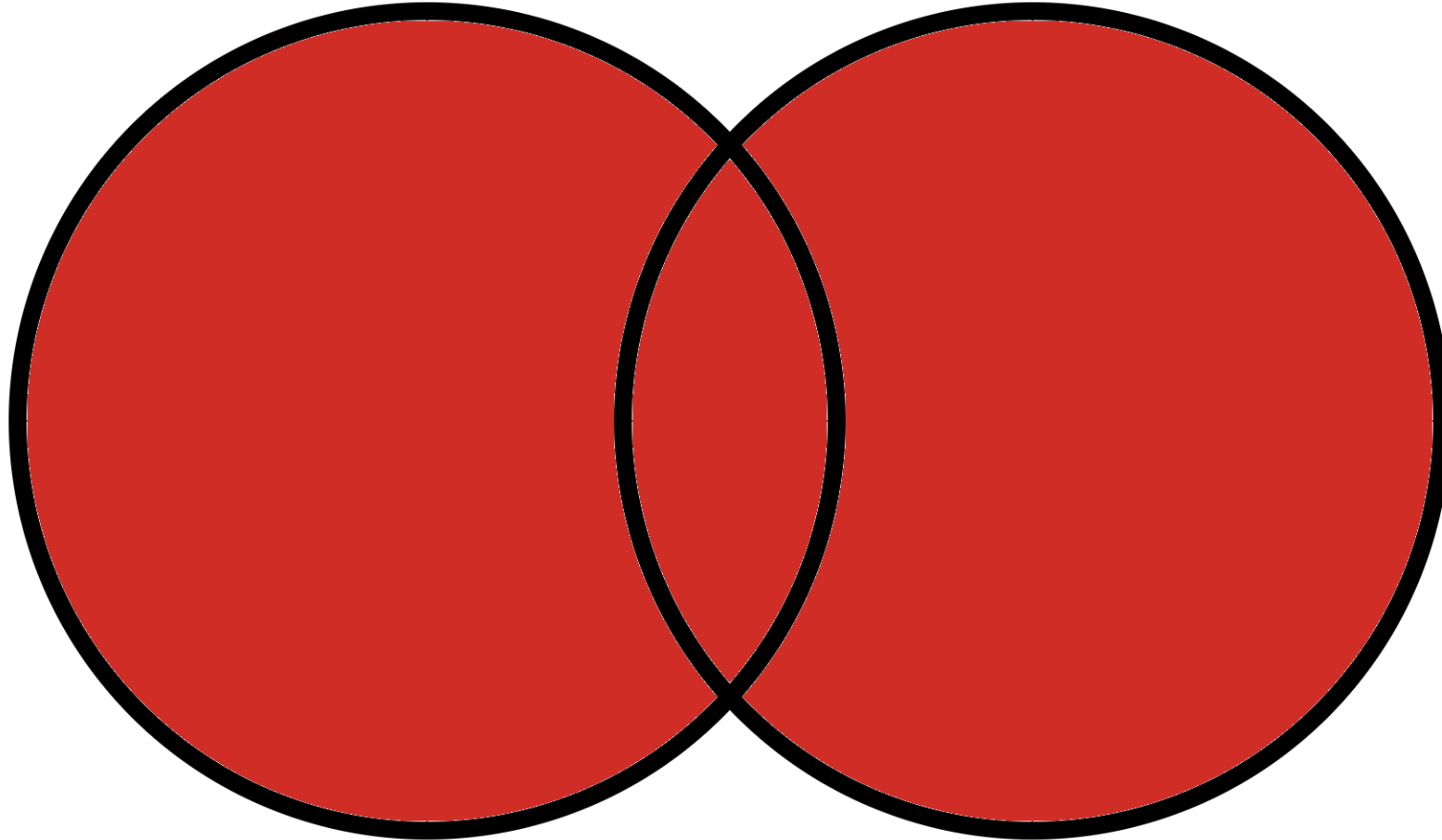
u.*,
c.contact_id,
c.contact_info

FROM

users u
RIGHT JOIN contacts c
ON u.user_id = c.user_id;

user_id	username	contact_id	contact_info
1	jdoe	1	<u>jdoe@mail.com</u>
1	jdoe	2	+41 71 345 32 65
3	fdupont	3	<u>fdupont@mail.com</u>
		4	<u>someone@mail.com</u>

16. LEFT, RIGHT and OUTER JOINS



16. LEFT, RIGHT and OUTER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4

SELECT

u.*,
c.contact_id,
c.contact_info

FROM

users u
FULL JOIN contacts c
ON u.user_id = c.user_id;

user_id	username	contact_id	contact_info
1	jdoe	1	<u>jdoe@mail.com</u>
1	jdoe	2	+41 71 345 32 65
2	userY		
3	fdupont	3	<u>fdupont@mail.com</u>
		4	<u>someone@mail.com</u>

16. LEFT, RIGHT and OUTER JOINS

user_id	username
1	jdoe
2	userY
3	fdupont

contact_id	contact_info	user_id
1	<u>jdoe@mail.com</u>	1
2	+41 71 345 32 65	1
3	<u>fdupont@mail.com</u>	3
4	<u>someone@mail.com</u>	4

SELECT

```
COALESCE(  
    u.user_id,  
    c.user_id  
) AS "user_id",  
username,  
contact_id,  
contact_info
```

FROM

```
users u  
FULL JOIN contacts c  
ON u.user_id = c.user_id;
```

user_id	username	contact_id	contact_info
1	jdoe	1	<u>jdoe@mail.com</u>
1	jdoe	2	+41 71 345 32 65
3	fdupont	3	<u>fdupont@mail.com</u>
4		4	<u>someone@mail.com</u>
2	userY		



16. LEFT, RIGHT and OUTER JOINS

Questions



17. Set operations with UNION, INTERSECT and EXCEPT

Set operations with **UNION, INTERSECT** and **EXCEPT**



17. Set operations with UNION, INTERSECT and EXCEPT

SELECT

column1, column2, ...

FROM

table_name1

UNION

SELECT

column1, column2, ...

FROM

table_name2;



17. Set operations with UNION, INTERSECT and EXCEPT

SELECT

sample_type as type

FROM

sample_types

UNION

SELECT

assay_type

FROM

assay_types;



17. Set operations with UNION, INTERSECT and EXCEPT

SELECT

column1, column2, ...

FROM

table_name1

INTERSECT

SELECT

column1, column2, ...

FROM

table_name2;



17. Set operations with UNION, INTERSECT and EXCEPT

SELECT

position

FROM

variants

WHERE

chromosome_id = 1

INTERSECT

SELECT

position

FROM

variants

WHERE

chromosome_id = 2;



17. Set operations with UNION, INTERSECT and EXCEPT

SELECT

column1, column2, ...

FROM

table_name1

EXCEPT

SELECT

column1, column2, ...

FROM

table_name2;



17. Set operations with UNION, INTERSECT and EXCEPT

SELECT

position

FROM

variants

WHERE

chromosome_id = 1

EXCEPT

SELECT

position

FROM

variants

WHERE

chromosome_id = 2;



17. Set operations with UNION, INTERSECT and EXCEPT

Questions



18. Date and Time

Date and Time



18. Date and Time

SELECT

*

FROM

ngs_assays

WHERE

DATE(run_date) = '2023-06-02';



18. Date and Time

SELECT

column1, column2, ...

FROM

table_name

WHERE

column1 **BETWEEN** value1 **AND** value2;



18. Date and Time

SELECT

*

FROM

ngs_assays

WHERE

DATE(run_date) BETWEEN '2023-06-02' AND '2023-06-03';



18. Date and Time

SELECT

*

FROM

ngs_assays

WHERE

DATETIME(run_date) = '2022-12-31 08:21:28';



18. Date and Time

SELECT

*

FROM

ngs_assays

WHERE

DATETIME(run_date) BETWEEN '2020-04-05 08:00:00' AND '2020-04-05 12:00:00';

Questions

VIEW



19. VIEW

CREATE VIEW

view_name **AS**

SELECT

column_name (s)

FROM

table_name

WHERE

condition;



19. VIEW

CREATE VIEW

available_samples AS

SELECT

s.id,
s.collection_date_time,
st.sample_type,
sd.device_name,
sd.storage_condition

FROM

samples s
INNER JOIN sample_statuses ss ON s.sample_status_id = ss.id
INNER JOIN storage_devices sd ON s.storage_device_id = sd.id
INNER JOIN sample_types st ON s.sample_type_id = st.id

WHERE

ss.sample_status = 'AVAILABLE';



19. VIEW

SELECT

*

FROM

available_samples;



19. VIEW

CREATE OR REPLACE VIEW – Not implemented in SQLite

available_samples **AS**

SELECT

s.id,
s.collection_date_time,
st.sample_type,
sd.device_name,
sd.storage_condition

FROM

samples s

INNER JOIN storage_devices sd **ON** s.storage_device_id = sd.id

INNER JOIN sample_types st **ON** s.sample_type_id = st.id

WHERE

s.sample_status_id = 1;



19. VIEW

DROP VIEW available_samples;



21. Next steps

ORM

or

Object relational mapper



21. Next steps

Datatypes



21. Next steps and conclusion

Database design



Conclusion

- Relational databases offer great **performances** and **flexibility** to store data
- Use **DBMS client** to write SQL queries so you can benefit from **syntax highlighting** and you can easily **format your queries**
- **Understanding your database schema** is the best way to help you write efficient queries
- **Filtering data** through **criteria** or **subqueries** is a good way to retrieve only what you need. Limiting your results is also a good strategy.
- **Aggregate functions** allow you to summarize several rows **into one value** but may be mixed with other columns as long as you use a **GROUP BY**.
- **Joining tables** allows you to reconstruct entities whose information is dispatched into multiple tables.
- Take advantage of **built-in functions** to manipulate your dates such as **DATE()** or **DATETIME()**.
- **Views** are an efficient way to **persist** a query inside your database. Consider using for **reporting** or **monitoring** your data.

Your **feedback** is precious



<https://tinyurl.com/tzu6s5tp>

