# SQL language

Use the `space_raw` table for this introduction. It is installed along with a clean installation of the hydenv database. If the table is not found, run:

```
python -m hydenv examples space
```

Below, you will find the SQL queries used in the video, followed by a summary of the lessons learned.

## SQL commands in the video

### Introduction 01

4:18:

```sql
-- count all rows
SELECT count(*) FROM space_raw;
```

5:26:

```sql
-- Get the first 5 rows
SELECT * FROM space_raw LIMIT 5;
```

6:10:

```sql
-- get the first 50 datum and detail information
SELECT datum, detail FROM space_raw LIMIT 50;
```

### Introuction 02

1:07:

```sql
-- order by datum
SELECT datum, detail FROM space_raw
ORDER BY datum ASC
LIMIT 50;
```

### Introduction 03

1:16:

```sql
-- filter for Apollo 13 status_mission
SELECT * FROM space_raw WHERE detail='Saturn V | Apollo 13'
```

2:56:

```sql
-- filter for all Saturn missions
SELECT * FROM space_raw WHERE detail LIKE 'Saturn%'
```

3:50:

```sql
-- filter for all Saturn missions
SELECT * FROM space_raw WHERE detail LIKE '%Apollo%'
```

### Introduction 04

3:15:

```sql
-- reduce to company_name using count function
SELECT
  company_name,
  count(*) as launches
FROM space_raw
GROUP BY company_name
ORDER BY launches DESC
```

## Summary

### Language

- standardized language, that works across all relational systems
- many packages implement pseudo-SQL for managing data, like `pandas` (Python) or `dplyr` (R)
- most RDBMS ship with their own SQL accent, thus code is not 100% compatible
- everything you do to a database, you do in SQL. It's a common language for building up a database **and** using it.

### Syntax

- SQL is **not** case sensitive
- commands are ended by `;`, except only one command is executed
- comments start with `--`
- **quotes and double-quotes are distinguished.** Single quotes are used for `string` literals, double quotes for structure names (like table or database names)
- SQL is type sensitive, thus the function `myFunction(5)` and `myFunction('five')` are two different functions.

### Basic commands

There are four basic commands for working with data:

- `SELECT` for requesting data or function output
- `INSERT` for adding datasets
- `UPDATE` for editing datasets
- `DELETE` for deleting data

In addition, the most important structural commands are:

- `CREATE` to add new structual elements, like `TABLE`, `CONSTRAINT`, `VIEW` or `FUNCTION`.
- `DROP` to delete database objects
- `ALTER` to edit database objects

## SELECT

The basic syntax to select is

```
SELECT column_1, cololumn_2 FROM tablename
```

```
SELECT location, detail, status_mission from space_raw
```

Instead of typing down **all** columns, there is the asterisk `*`, as a shortcut:

```
SELECT * FROM tablename
```

### Filter

To filter the datasets, use `WHERE`. You can use the single equal sign for *exact matches*, where an attribute has to exactly match the given string. With `LIKE` you can filter by *partial matches* on string fields. Use `%` as a wildcard. `LIMIT` the results, to prevent PostgreSQL from returning everything, that matches a filter.

```
SELECT * FROM space_raw where detail LIKE '%Sputnik%'
```

```
SELECT * FROM space_raw WHERE detail = 'Saturn V | Apollo 13'
```

### Order

Order results depending on the data type. Combined with a `LIMIT`, you can quickly access the largest, smallest values.

```
SELECT * FROM space_raw ORDER BY datum ASC LIMIT 5
```

### Aggregation

You can aggregate result by grouping them on one or many columns using `GROUP BY` and then reduce all remaining (ungrouped) columns to a scalar value. For available functions, search the documentation.

The most important are:

- avg - mean value
- min - minimum value
- max - maximum value
- sum - sum of all values

- count - number of values
- date_part - for `TIMESTAMP` data types; **extracts** the given date part
- date_trunc - for `TIMESTAMP` data types; **truncates** to the given date part