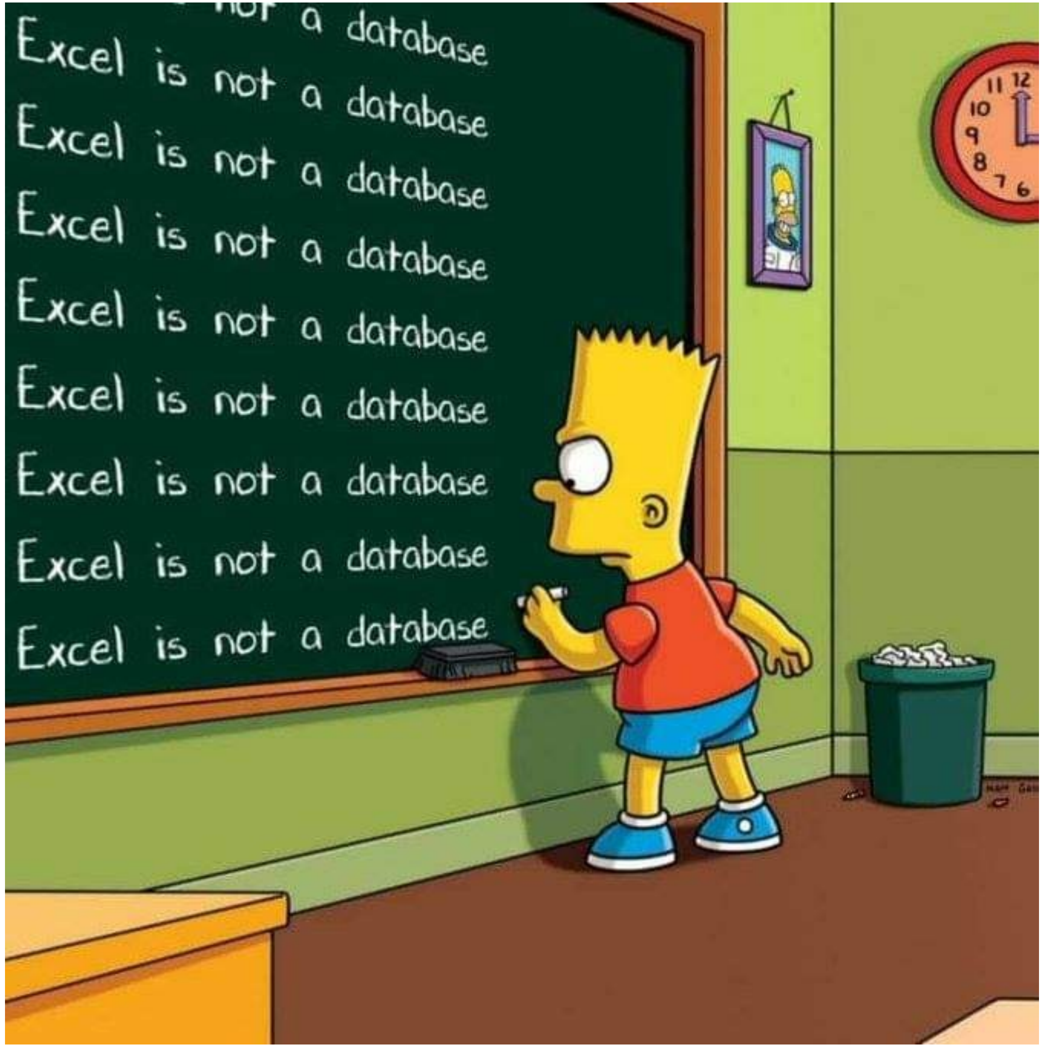
Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

# Databases 🤖

## [Dr. Hanjo Odendaal]

Photo by Stefan Els

# Why do we use databases?



# Why do we use databases?

Although there are many reasons to shift from using Excel to databases, the most important cornerstones are: (i) Data Integrity, (ii) Redundancy, (iii) Error prone, (iv) User Access and Security and (v) Data Accessibility and Speed.

Data Integrity:

- There are a set of rules that govern the structure of the data, how different information relate to one another and also what input can be put into a certain column of a table (ex. numeric vs string).
- These rules and protocols build a general framework under which everyone must adhere to, and as such, increases reliability of the data.
- It also removes questions like "*What do I do when I get new data? Where do I store that data? How must the data look?*".
- Excel is limited to 1,000,000 rows of data in a single sheet. If you operating at close to the edge of that amount of data how difficult to you think it is to ensure the quality of the information?

# Why do we use databases?

Redundancy:

- I think all of us know version control of files that are called `final_analytics_tom_v2.3_client_clean.xlsx`. These files eventually just become copies of the same data with small changes that are difficult to track and keep clean.
- Using relational databases also ensures that we separate information out across tables to ensure we do not have multiple versions of the same data in different places. Example would be to separate out customers and their purchases.
  - When updating a customer's information, we only have to update the customers table and not the purchases tables, as the purchases table only links back to the customer via a unique id (also called a key) of some sort.

Error prone:

- Excel sheets (as we saw earlier) is very much susceptible to proliferation of errors, especially when the data gets large. There is no way of know what changed and how when someone accidentally overwrites a cell/row/column.
- Also, because sheets are usually linked together, if the sheet changes unexpectedly (perhaps someone added a column), suddenly the formulas are no longer correct or links to the data are broken.

# Why do we use databases?

Access and Security:

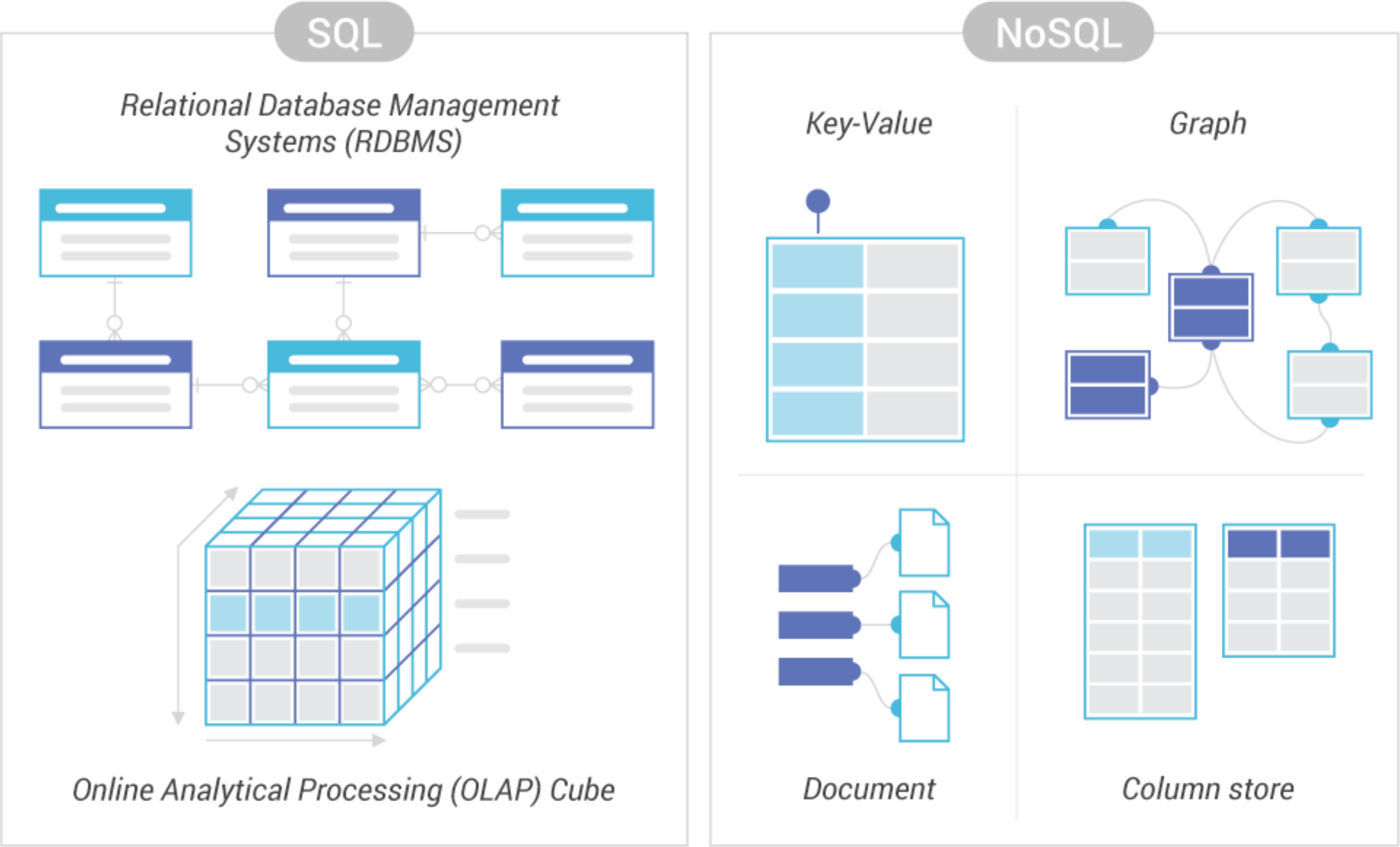
- Multiple users can work on a database at the same time. Most of the time end-users only need to collect or aggregate data for their purposes, which means we can create specials user that are only allowed those operations. This ensures that the risk of unknowledgeable users do not accidentally change the underlying data without them knowing.
- With personal information protection laws coming into force, one also wants to restrict access to certain types of information by only granting access to those who have the right clearance[1].

Data Accessibility and Speed:

- Ain't no one got time to work on an Excel sheet over 10,000 rows. Having to quickly analyse information using aggregation tools such as *pivot* or *vlookups* becomes a total nightmare. Excel is dynamic, which means every action causes all the information to automatically recalculate.
- Databases allows you to do deep analysis of data over millions (even billions) of rows of data in seconds.
- Because you most likely will be using a relational setup in a database, you only query the data you need, not the whole information set.

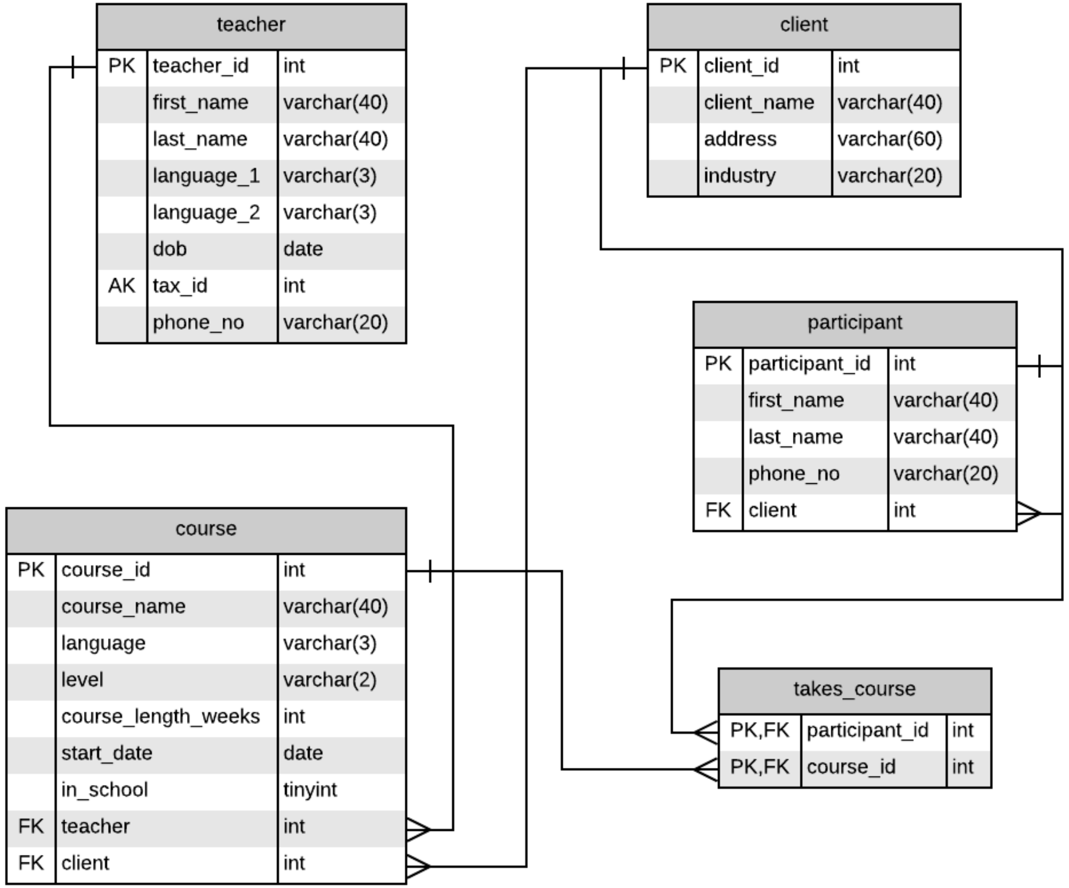
[1] *Please do NOT ever send personal information in plain text in Excels spreadsheets over email, unless you want to end up in jail.* 💀

# Different database types


SQL
Relational Database Management Systems (RDBMS)
Online Analytical Processing (OLAP) Cube
NoSQL
Key-Value
Graph
Document
Column store

*link to original website

# RDMS database example data schema



| | teacher | |
|---|---|---|
| PK | teacher_id | int |
| | first_name | varchar(40) |
| | last_name | varchar(40) |
| | language_1 | varchar(3) |
| | language_2 | varchar(3) |
| | dob | date |
| AK | tax_id | int |
| | phone_no | varchar(20) |

| | client | |
|---|---|---|
| PK | client_id | int |
| | client_name | varchar(40) |
| | address | varchar(60) |
| | industry | varchar(20) |

| | participant | |
|---|---|---|
| PK | participant_id | int |
| | first_name | varchar(40) |
| | last_name | varchar(40) |
| | phone_no | varchar(20) |
| FK | client | int |

| | course | |
|---|---|---|
| PK | course_id | int |
| | course_name | varchar(40) |
| | language | varchar(3) |
| | level | varchar(2) |
| | course_length_weeks | int |
| | start_date | date |
| | in_school | tinyint |
| FK | teacher | int |
| FK | client | int |

| | takes_course | |
|---|---|---|
| PK,FK | participant_id | int |
| PK,FK | course_id | int |

*link to original website

# Different database structures

Do note, that although we do not cover OLAP database in this course, they tend to be a little bit different as they try to avoid complex joins which could slow down analytics.

Database structures, or *schema* design, depends a lot on the application of the database. Although there are different schemas and designs, they do have some common traits:

- Includes the name of the fields in the table.
- The type that the field consists of (numeric, date, varchar etc.).
- Associations and keys linking fields.

Common schemas that you might encounter are:

- Star schema
- Snowflake schema
- Fact constellation

# Different database structures

To understand schemas a bit better, we need an understanding of the pieces. The two most important components consist of: `fact` and `dimension` tables.

FACT:

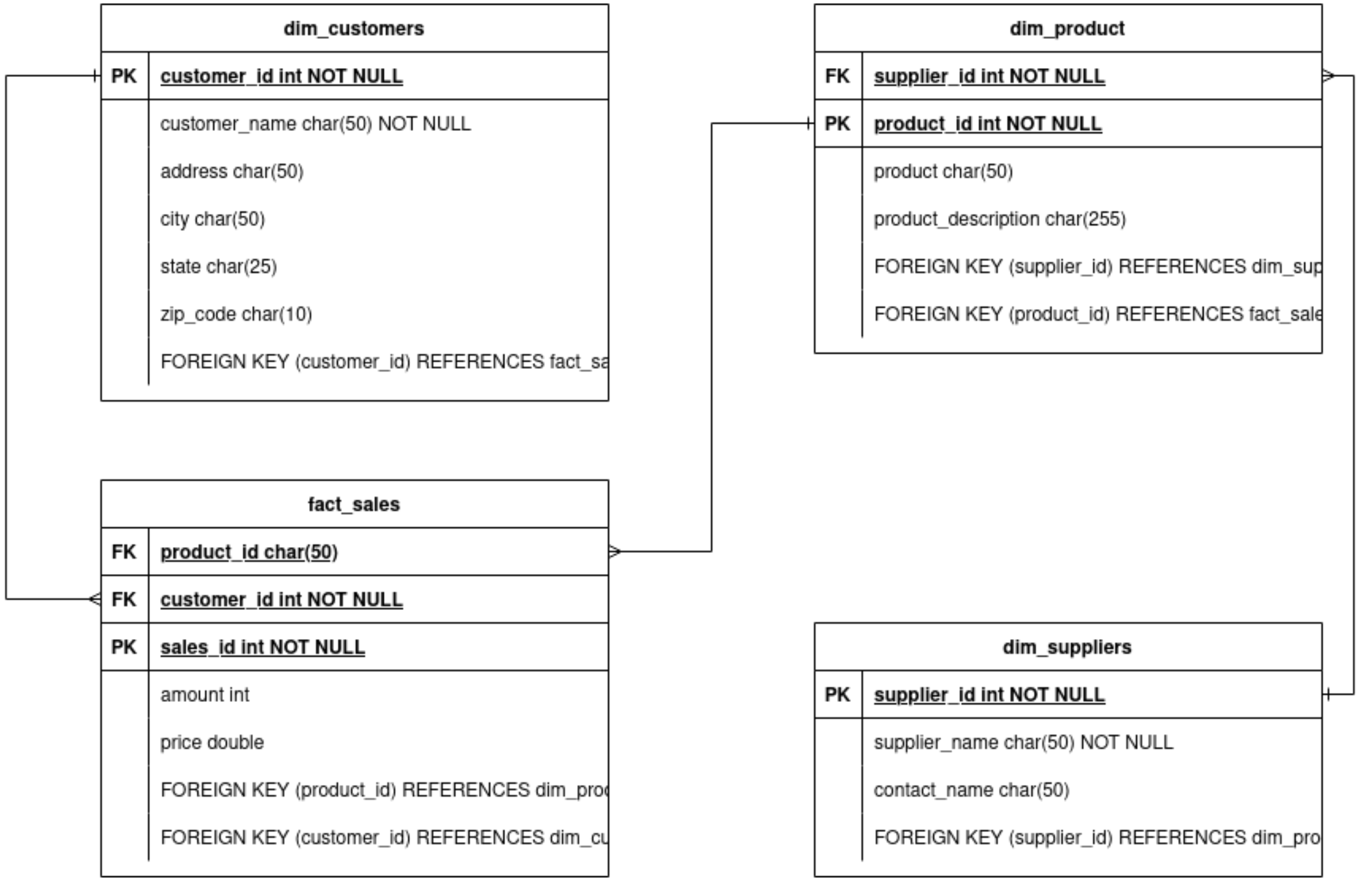
- Fact table contains measurements, metrics, and *facts* about a business process
    - ex. *sales* or *webpage visits*.
- Fact tables form the primary table of the design and are usually normalized
    - We assign a numerical number or code to an attribute for better performance
    - An example of this would be where we code GENDER as 1 for male and 2 for female.

DIMENSION:

- Provides the information about the facts
    - ex. *location* of transaction, *customer*
- These tables are de-normalized and have to be joined to the fact table table before analysis can happen.
- The tables also usually contain descriptions of the field in order to make it easier to understand.

# Different database structures: Star Schema

> Star schema design has only *one* fact table and multiple dimension tables. This is a very common design as its relational properties are easily understood.



| dim_customers | |
|---|---|
| PK | customer_id int NOT NULL |
| | customer_name char(50) NOT NULL |
| | address char(50) |
| | city char(50) |
| | state char(25) |
| | zip_code char(10) |
| | FOREIGN KEY (customer_id) REFERENCES fact_sa |

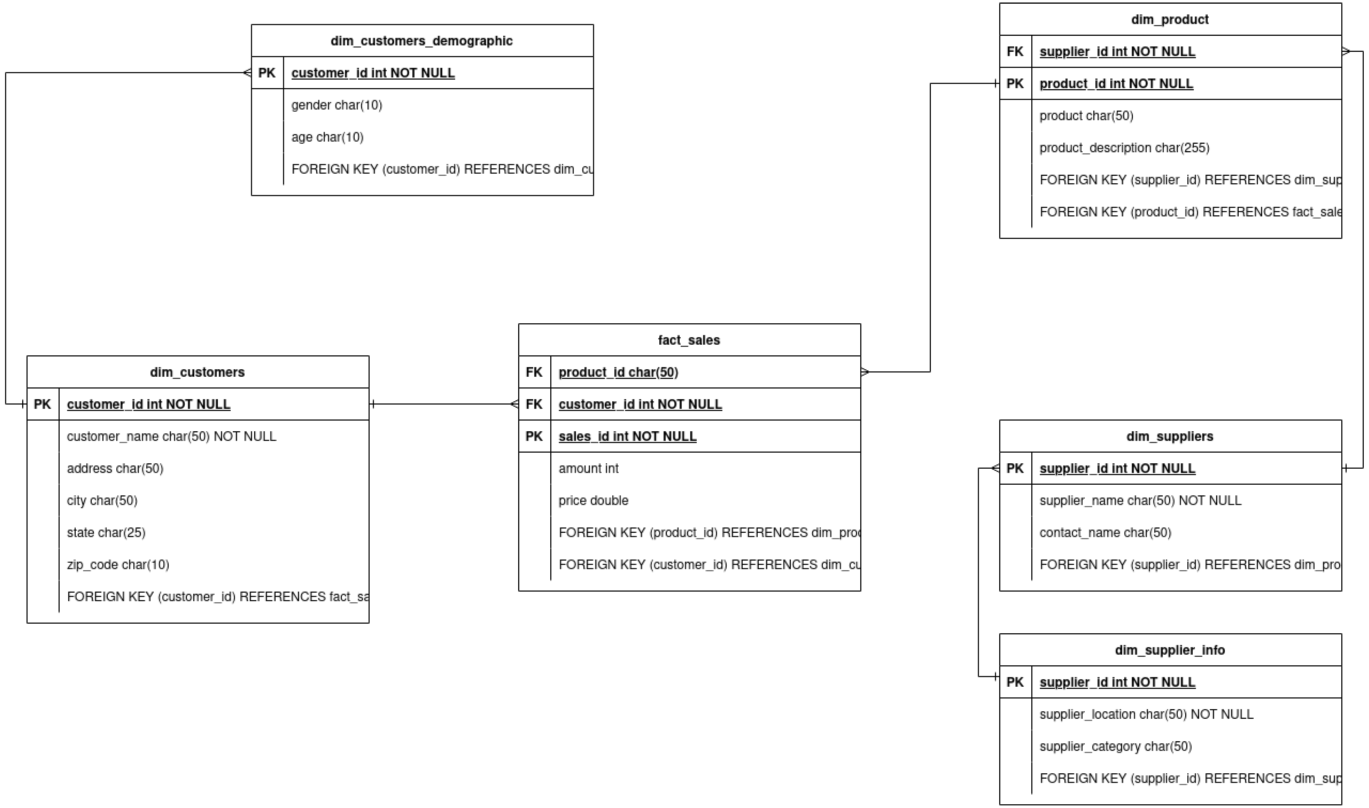
| dim_product | |
|---|---|
| FK | supplier_id int NOT NULL |
| PK | product_id int NOT NULL |
| | product char(50) |
| | product_description char(255) |
| | FOREIGN KEY (supplier_id) REFERENCES dim_sup |
| | FOREIGN KEY (product_id) REFERENCES fact_sale |

| fact_sales | |
|---|---|
| FK | product_id char(50) |
| FK | customer_id int NOT NULL |
| PK | sales_id int NOT NULL |
| | amount int |
| | price double |
| | FOREIGN KEY (product_id) REFERENCES dim_prod |
| | FOREIGN KEY (customer_id) REFERENCES dim_cu |

| dim_suppliers | |
|---|---|
| PK | supplier_id int NOT NULL |
| | supplier_name char(50) NOT NULL |
| | contact_name char(50) |
| | FOREIGN KEY (supplier_id) REFERENCES dim_pro |

# Different database structures: Snowflake Schema

> Snowflake schema design extends the Star Schema by only *one* fact table, multiple dimension tables, each with their own dimension tables. This adds another layer of abstration to the dimension tables and could contain additional information about attributes not in use every day.



| | dim_customers_demographic |
|---|---|
| PK | customer_id int NOT NULL |
| | gender char(10) |
| | age char(10) |
| | FOREIGN KEY (customer_id) REFERENCES dim_cu |

| | dim_product |
|---|---|
| FK | supplier_id int NOT NULL |
| PK | product_id int NOT NULL |
| | product char(50) |
| | product_description char(255) |
| | FOREIGN KEY (supplier_id) REFERENCES dim_sup |
| | FOREIGN KEY (product_id) REFERENCES fact_sale |

| | dim_customers |
|---|---|
| PK | customer_id int NOT NULL |
| | customer_name char(50) NOT NULL |
| | address char(50) |
| | city char(50) |
| | state char(25) |
| | zip_code char(10) |
| | FOREIGN KEY (customer_id) REFERENCES fact_sa |

| | fact_sales |
|---|---|
| FK | product_id char(50) |
| FK | customer_id int NOT NULL |
| PK | sales_id int NOT NULL |
| | amount int |
| | price double |
| | FOREIGN KEY (product_id) REFERENCES dim_pro |
| | FOREIGN KEY (customer_id) REFERENCES dim_cu |

| | dim_suppliers |
|---|---|
| PK | supplier_id int NOT NULL |
| | supplier_name char(50) NOT NULL |
| | contact_name char(50) |
| | FOREIGN KEY (supplier_id) REFERENCES dim_pro |

| | dim_supplier_info |
|---|---|
| PK | supplier_id int NOT NULL |
| | supplier_location char(50) NOT NULL |
| | supplier_category char(50) |
| | FOREIGN KEY (supplier_id) REFERENCES dim_sup |

Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

# Basic SQL ⚒

# Jumping in with both feet

> In an advanced Database course you should cover databases, its setup and optimization in much more depth. But for now, lets focus on getting you writing SQL queries and leave the database setup to the DBA.

To learn SQL, one needs to have a database. I have partitioned a DB for each of you! We are going to need mostly functions for this exercise: `db_query` and `db_write`. These functions are going to have the same format:

1) Connect to DB using variables from the `Environment`

2) Either `read` or `write` and then

3) …make sure that if the function exits, we disconnect

… dont worry if you feel like this …

# Jumping in with both feet

💡 Tip: In copy paste this! No need to reinvent the wheel 💡

**Step 0:** Specify your project variables in `.Renviron`, `usethis::edit_r_environ()`

```
gp_data = datascience
gp_user = datascience
gp_pass = f5VPEC8nsU01QKbSxSfv
gp_host = localhost
gp_port = 3141
```

**Step 1:** Connect to DB using variables from the `Environment` using `dbbasic`

```r
remotes::install_github("HanjoStudy/dbbasic")
library(dbbasic)
conn <- db_connect(db = "psql_datascience")
DBI::dbDisconnect(conn)
```

# What is dbbasic?

`dbbasic` is a little passion project from when I worked at 71point4. You can find it HERE

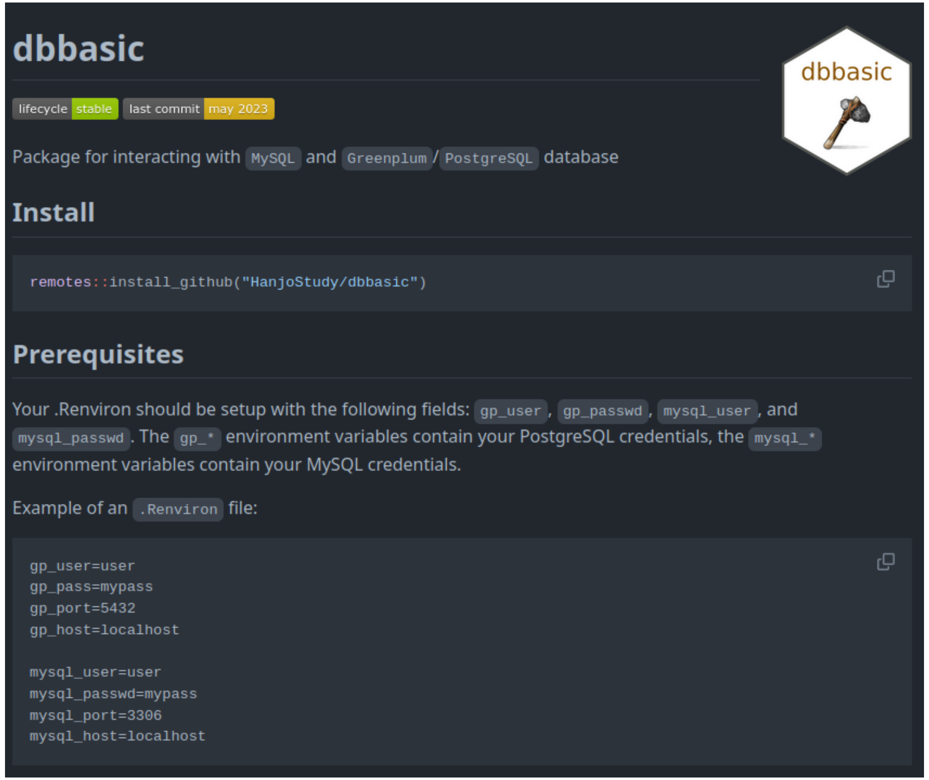
The aim is to assist and facilitate R programmers with interacting with databases without too much cognitive overhead:

- Read
- Write
- Connect

It also has some very nice utilities:

- `db_collapes` - if you need to collapse a vector for an `IN` style query
- `hash` - creates a SHA256 hash
- `tibble_to_sql` - helps with boilerplate

*https://github.com/HanjoStudy/dbbasic/tree/master*



## dbbasic

Package for interacting with `MySQL` and `Greenplum` / `PostgreSQL` database

### Install

```
remotes::install_github("HanjoStudy/dbbasic")
```

### Prerequisites

Your .Renviron should be setup with the following fields: `gp_user`, `gp_passwd`, `mysql_user`, and `mysql_passwd`. The `gp_*` environment variables contain your PostgreSQL credentials, the `mysql_*` environment variables contain your MySQL credentials.

Example of an `.Renviron` file:

```
gp_user=user
gp_pass=mypass
gp_port=5432
gp_host=localhost

mysql_user=user
mysql_passwd=mypass
mysql_port=3306
mysql_host=localhost
```

# Rstudio + dbbasic?

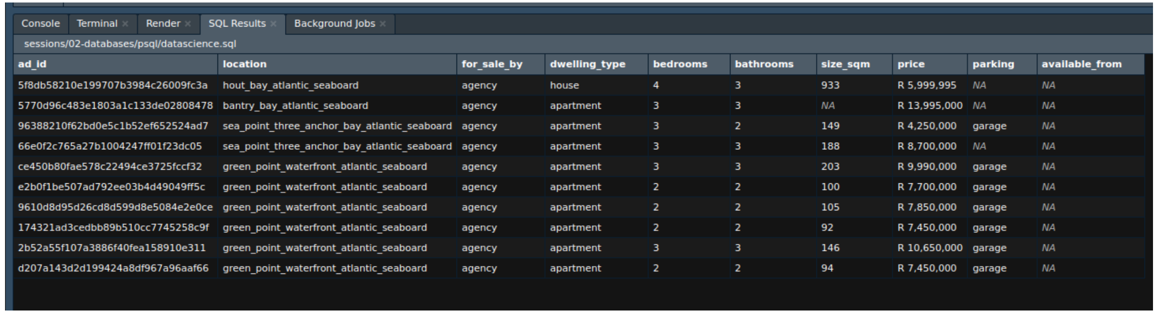
To use `dbbasic` in R, create a new `dev.sql` file in your project and add `-- !preview conn=db_connect(db = "psql_datascience")` to the top of the file.

⚠️ For this to work your `.Renviron` file in your project directory must be set up correctly!

```sql
-- !preview conn=db_connect(db = "psql_datascience")

SELECT * FROM gumtree LIMIT 10
```

And then press `CTRL + SHIFT + ENTER` for magic 🧙



sessions/02-databases/psql/datascience.sql

| ad_id | location | for_sale_by | dwelling_type | bedrooms | bathrooms | size_sqm | price | parking | available_from |
|---|---|---|---|---|---|---|---|---|---|
| 5f8db58210e199707b3984c26009fc3a | hout_bay_atlantic_seaboard | agency | house | 4 | 3 | 933 | R 5,999,995 | NA | NA |
| 5770d96c483e1803a1c133de02808478 | bantry_bay_atlantic_seaboard | agency | apartment | 3 | 3 | NA | R 13,995,000 | NA | NA |
| 96388210f62bd0e5c1b52ef652524ad7 | sea_point_three_anchor_bay_atlantic_seaboard | agency | apartment | 3 | 2 | 149 | R 4,250,000 | garage | NA |
| 66e0f2c765a27b1004247ff01f23dc05 | sea_point_three_anchor_bay_atlantic_seaboard | agency | apartment | 3 | 3 | 188 | R 8,700,000 | NA | NA |
| ce450b80fae578c22494ce3725fccf32 | green_point_waterfront_atlantic_seaboard | agency | apartment | 3 | 3 | 203 | R 9,990,000 | garage | NA |
| e2b0f1be507ad792ee03b4d49049ff5c | green_point_waterfront_atlantic_seaboard | agency | apartment | 2 | 2 | 100 | R 7,700,000 | garage | NA |
| 9610d8d95d26cd8d599d8e5084e2e0ce | green_point_waterfront_atlantic_seaboard | agency | apartment | 2 | 2 | 105 | R 7,850,000 | garage | NA |
| 174321ad3cedbb89b510cc7745258c9f | green_point_waterfront_atlantic_seaboard | agency | apartment | 2 | 2 | 92 | R 7,450,000 | garage | NA |
| 2b52a55f107a3886f40fea158910e311 | green_point_waterfront_atlantic_seaboard | agency | apartment | 3 | 3 | 146 | R 10,650,000 | garage | NA |
| d207a143d2d199424a8df967a96aaf66 | green_point_waterfront_atlantic_seaboard | agency | apartment | 2 | 2 | 94 | R 7,450,000 | garage | NA |

*You can also do some very clever things with `dbbasic` + `Quarto` + `SQL chunks`... but thats for another day.*

# The KING of all statements: SELECT

The way to think about SQL is in terms of english commands. Also, also start from the inside and work your way out (you will see what I mean). ⚠️ It is good practice to always end your statements with `LIMIT 10` until you are sure that the correct results is returned. Working on billion row tables and forgetting to limit your results can crash tables.

Lets start with the two statements you will most likely use every day:

- Counting how many rows there are

```sql
SELECT COUNT(*) FROM gumtree LIMIT 10;
```

```r
db_query("SELECT COUNT(*) FROM gumtree LIMIT 10;", db = "psql_datascience")
```

- Getting a 10 row sample

```sql
SELECT * FROM gumtree LIMIT 10;
```

```r
db_query("SELECT * FROM gumtree LIMIT 10;", db = "psql_datascience")
```

# The KING of all statements: SELECT

Previously I decided I wanted to return all the columns (`*`), but what if I only want to return one or two of the columns?

```
SELECT {column1}, {column2} FROM table LIMIT 10;
```

Lets only return `dwelling_type`, `size_sqm` and `price`:

```
SELECT dwelling_type, size_sqm, price FROM gumtree LIMIT 10;
```

👍 It is good practice to not have *long* SQL statements in one row.

> Code Needs a lot of whitespace.
> That is how it breaths
>
> — Roger Peng, Jenny Bryan, useR 2018

# The KING of all statements: SELECT

Lets build a *bigger* `SELECT` statement ( I like 3 tab indentation ):

```
SELECT
      dwelling_type,
      bedrooms,
      bathrooms,
      parking,
      size_sqm,
      price
FROM gumtree
LIMIT 10;
```

# SELECT but with filter criteria

What happens if we only want to return an ad of a certain type?

Well, then we can employ the `WHERE` statement. We are going to collect the same columns as previously, but now we will specify the `WHERE` criteria on `dwelling_type` column where equal to `house`:

```sql
SELECT
      dwelling_type,
      bedrooms,
      bathrooms,
      parking,
      size_sqm,
      price
FROM gumtree
WHERE
      dwelling_type = 'house'
LIMIT 10;
```

**Quick Practice:**, write the code to bring back 100 examples where there is `parking` and `ORDER BY` price. TIP: Please google "psql filter NOT NULL on column"

# SELECT but with filter criteria and order

In certain circumstances, it is necessary to order your data to get the correct output. For instance if we want to get the top 10 largest `size_sqm` houses:

```
SELECT
      dwelling_type,
      bedrooms,
      bathrooms,
      parking,
      size_sqm,
      price
FROM gumtree
WHERE
      dwelling_type = 'house'
      AND size_sqm IS NOT NULL
ORDER BY
      size_sqm DESC
LIMIT 10
;
```

# Lets clean and upload a clean version!

By now you would have seen that price and `size_sqm` and `price` is not in the correct format... Lets fix that:

1) Pull whole database into R, 2) Fix `size_sqm` and `price`, 3) Upload to new table called `gumtree_clean`

**(1)** Pull whole database into R & **(2)** Fix `size_sqm` and `price`

```r
library(dplyr)
gumtree_clean <- db_query("SELECT * FROM gumtree;", db = "psql_datascience") %>%
  mutate(size_sqm = as.numeric(size_sqm)) %>%
  # remove all except numbers
  mutate(price = as.numeric(gsub("[^0-9]+", "", price)))
```

**(3)** Upload to new table called `gumtree_clean`

```r
db_write(gumtree_clean, "gumtree_clean", db = "psql_datascience")
db_query("DROP TABLE gumtree_clean", db = "psql_datascience")
db_write(gumtree_clean, "gumtree_clean", db = "psql_datascience")

db_query("SELECT * FROM gumtree LIMIT 10;", db = "psql_datascience")
```

# WELL DONE!!!

# Time for security 🔒

I have set your password for you, but given that you are a super user, you can now make your password whatever you want!

- This will also ensure that someone doesn't mess with your sever

```sql
ALTER USER datascience WITH PASSWORD 'XXX';
```

# Aggregations (Pivoting) in SQL

Pivoting forms part of the aggregation function of SQL. This helps us answer questions like:

- What is the average price of houses by `dwelling_type`?
- Total value and volume per location?

As you can see, aggregations or `GROUP BY` clauses gets used OFTEN, so learn it well and get comfortable with it.

# Aggregations (Pivoting) in SQL

> What is the average price of houses by `dwelling_type`?

```sql
SELECT
      dwelling_type,
      AVG(price) as mean_price
FROM gumtree_clean
WHERE
      price IS NOT NULL
GROUP BY
      dwelling_type
ORDER BY
      AVG(price) DESC
LIMIT 10
;
```

```r
db_query("
          SELECT
                dwelling_type,
                AVG(price) as mean_price
          FROM gumtree_clean
          WHERE
                price IS NOT NULL
          GROUP BY
                dwelling_type
          ORDER BY
                AVG(price) DESC
          LIMIT 10
          ; ", db = "psql_datascience")
```

Once you are comfortable that you have the query correctly specified, drop the `LIMIT` and scroll through your magnificent piece of work!

# Aggregations (Pivoting) in SQL

> Total value and volume per location?

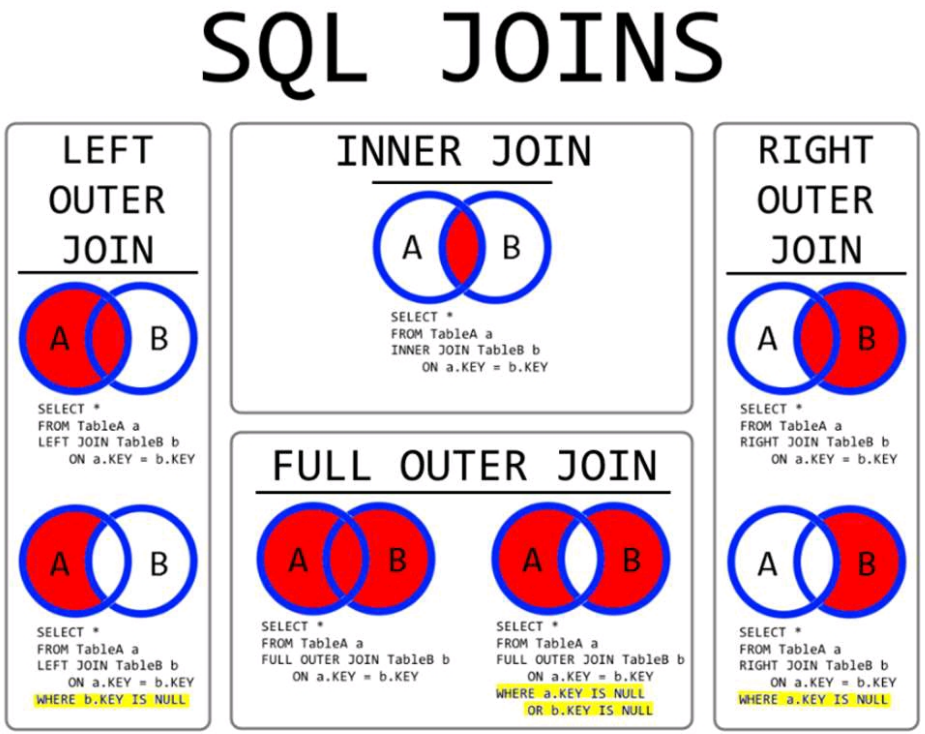
```
SELECT
    location,
    COUNT(*) AS volume,
    SUM(price) AS value
FROM
    gumtree
GROUP BY
    location
LIMIT 10
;
```

Notice how I `ALIAS` my aggregations as {aggregation} then {name}. This will make your life a lot easier and in some case it is mandatory... as in joins.

```
db_query("
     SELECT
        location,
        COUNT() AS volume,
        SUM(price) AS value,
        SUM(price)/COUNT() avg_price
     FROM
        gumtree_clean
     GROUP BY
        location
     ORDER BY
        SUM(price) DESC
     LIMIT 10
     ;", db = "psql_datascience")
```

# Last but not least: JOINS

By now you are asking yourself, if we designed our database in the beautiful star schema that we talked about earlier, how do we *join* all the information together again? This is where `JOINS` come in and there are a multitude of them. Most important one is `LEFT JOIN` and `INNER JOIN`:



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
    ON a.KEY = b.KEY
WHERE b.KEY IS NULL
```

```
SELECT *
FROM TableA a
INNER JOIN TableB b
    ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
    OR b.KEY IS NULL
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
    ON a.KEY = b.KEY
WHERE a.KEY IS NULL
```

# Last but not least: JOINS

Lets attempt a basic join before we combine joins with aggregations. To start off we will `JOIN` the `gumtree_descriptions` table onto the `gumtree_clean` table:

- First check what tables are available:

```
db_query("SELECT * FROM pg_catalog.pg_tables WHERE schemaname = 'public'", db = "psql_datascience")
```

- Next get a sample of rows from `gumtree_descriptions`:

```
db_query("SELECT * FROM gumtree_description LIMIT 10", db = "psql_datascience")
```

> From this we see that `gumtree_clean.ad_id` = `gumtree_descriptions.ad_id`!

# Last but not least: JOINS

Now that we have the specific keys from the different tables, there are two ways to join: (1) When name is the same in both table, (2) When they are different:

### When name is the same in both table

```
SELECT
  *
FROM gumtree_description
LEFT JOIN(
  SELECT * FROM gumtree_clean
) tbl_clean
USING(ad_id)
;
```

### When they are different

```
SELECT
  *
FROM gumtree_descriptions
LEFT JOIN(
  SELECT * FROM gumtree_clean
) tbl_clean
ON gumtree_clean.ad_id = gumtree_descriptions.ad_id
;
```

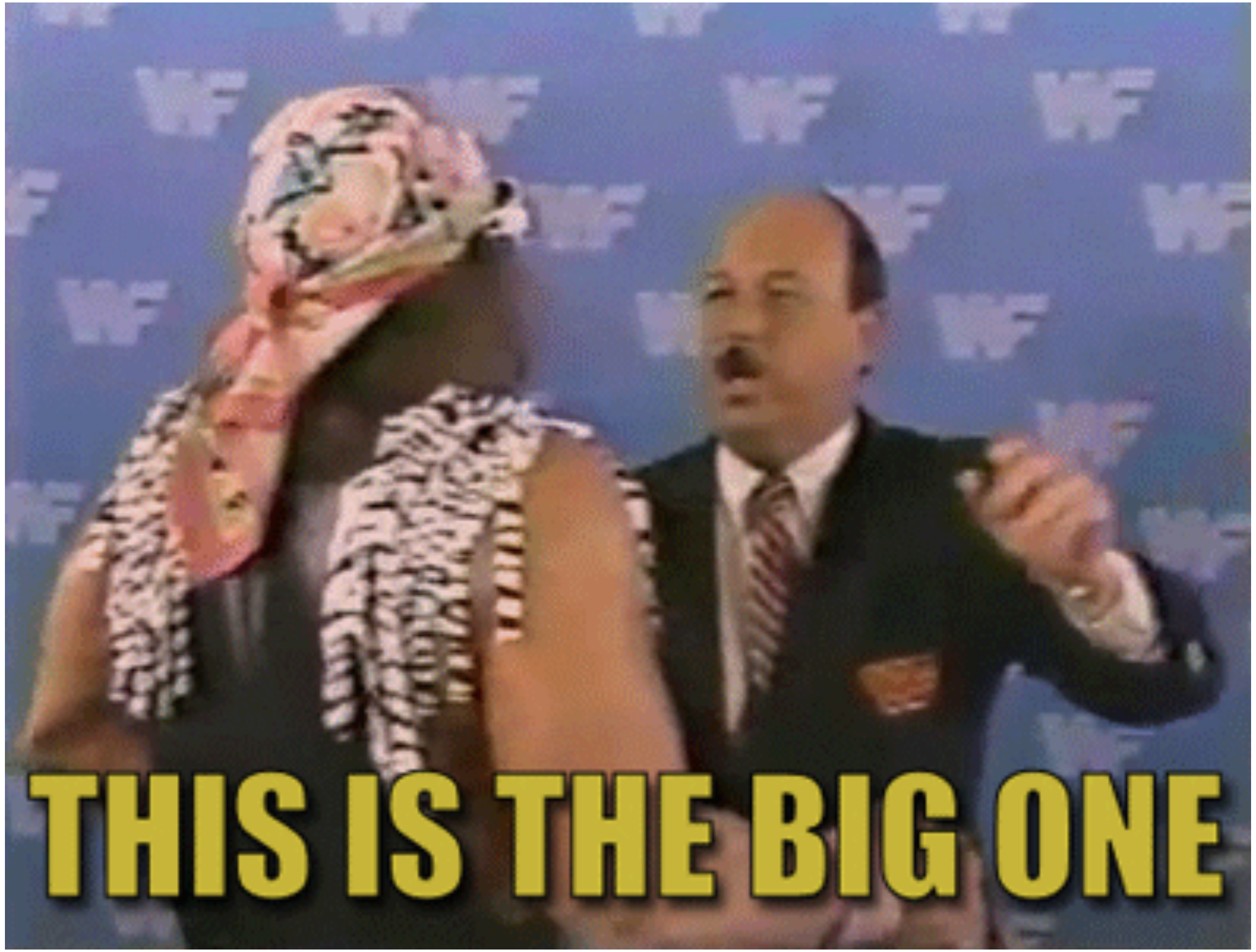
- Important to note the `ALIAS` of the inner table called `tbl_clean`.

# Last but not least: JOINS

There are ways to optimize your joins to be extremely fast. Although we do not cover these in this course, it is interesting and worth knowing none the less.

- One is keys (which is why we use primary and muli keys in tables).
- Another is query optimization through column selection and subqueries.
    - Although we do not cover these in this course, having knowledge of advance backend mechanics can sometimes take your execution time from days to minutes.

# Bonus Round: CTEs



# Common Table Expressions

This is the `%>%` equivalent for SQL. The structure is quite simple:

```
WITH cte_one AS(
... some SQL ...
), cte_two AS(
... do something with cte_one ...
)
SELECT * FROM cte_two
```

These are quite 'advanced' topics, but can promise you once you start using them you wont go back to subqueries[1].

Now lets move on to asking practical questions

> What are the top words associated with a property advert? 🤓

[1] *If you want to know why I say that, go read up on subqueries.*

Stellenbosch
UNIVERSITY
IYUNIVESITHI
UNIVERSITEIT

# Case Study

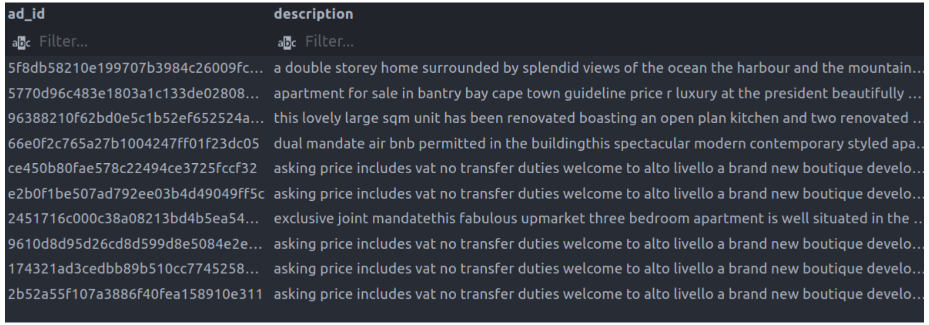
# Case Study: Property

**Step 1:** I know we are going to need to remove useless words (stopwords) such as "a", "the" etc. So lets upload a stopwords table to our DB:

```
db_write(tidytext::stop_words, table_name = "stopwords", db = "psql_datascience")
```

**Step 2:** Cleaning the text of puncuation:

```
WITH cte_one AS(
    SELECT ad_id,
        REGEXP_REPLACE(LOWER(description), '[^A-Za-
    FROM gumtree_description
)
SELECT * FROM cte_one LIMIT 10;
```

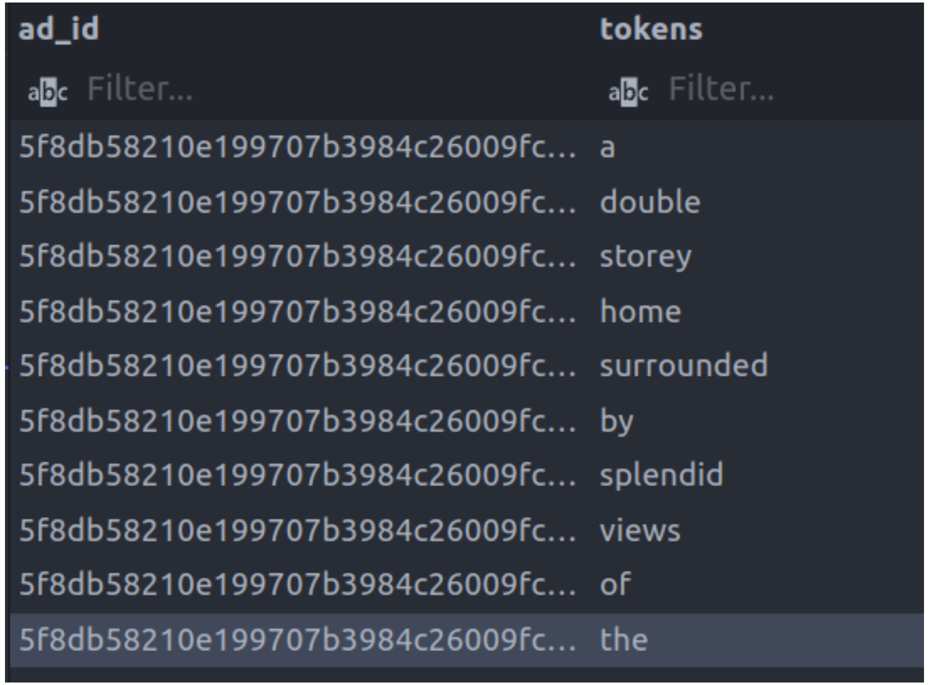


| ad_id | description |
|---|---|
| Filter... | Filter... |
| 5f8db58210e199707b3984c26009fc... | a double storey home surrounded by splendid views of the ocean the harbour and the mountain... |
| 5770d96c483e1803a1c133de02808... | apartment for sale in bantry bay cape town guideline price r luxury at the president beautifully ... |
| 96388210f62bd0e5c1b52ef652524a... | this lovely large sqm unit has been renovated boasting an open plan kitchen and two renovated ... |
| 66e0f2c765a27b1004247ff01f23dc05 | dual mandate air bnb permitted in the buildingthis spectacular modern contemporary styled apa... |
| ce450b80fae578c22494ce3725fccf32 | asking price includes vat no transfer duties welcome to alto livello a brand new boutique develo... |
| e2b0f1be507ad792ee03b4d49049ff5c | asking price includes vat no transfer duties welcome to alto livello a brand new boutique develo... |
| 2451716c000c38a08213bd4b5ea54... | exclusive joint mandatethis fabulous upmarket three bedroom apartment is well situated in the ... |
| 9610d8d95d26cd8d599d8e5084e2e... | asking price includes vat no transfer duties welcome to alto livello a brand new boutique develo... |
| 174321ad3cedbb89b510cc7745258... | asking price includes vat no transfer duties welcome to alto livello a brand new boutique develo... |
| 2b52a55f107a3886f40fea158910e311 | asking price includes vat no transfer duties welcome to alto livello a brand new boutique develo... |

# Case Study: Property

**Step 3:** Tokenise the long text into words

```
WITH cte_one AS(
    SELECT ad_id,
        REGEXP_REPLACE(LOWER(description), '[^A-Za-
    FROM gumtree_description
), cte_two AS(
    SELECT ad_id,
        TRIM(REGEXP_SPLIT_TO_TABLE(LOWER(descriptio
    FROM cte_one
)
SELECT * FROM cte_two LIMIT 10
```

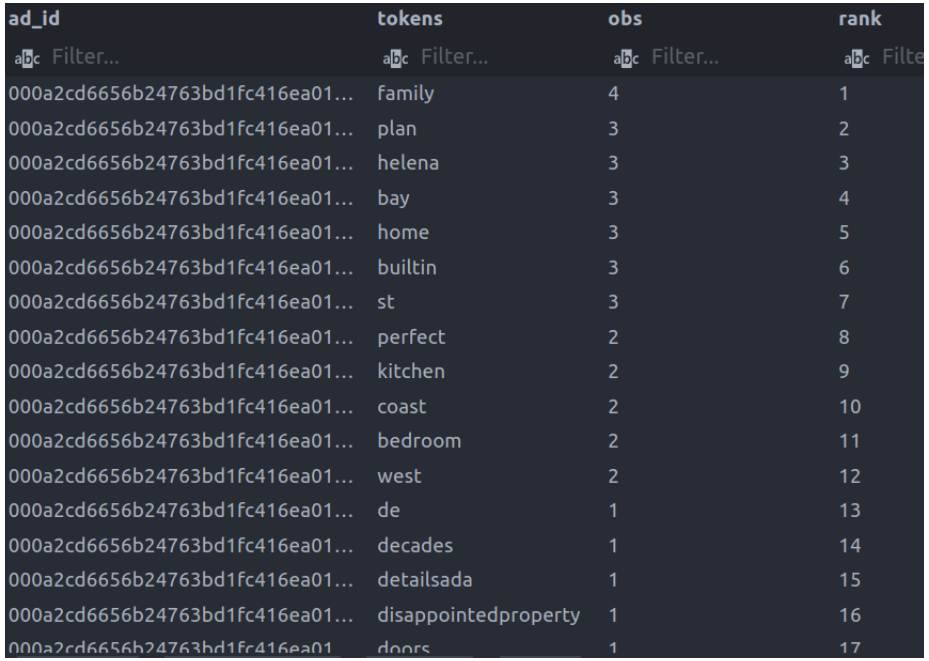


| ad_id | tokens |
|---|---|
| 5f8db58210e199707b3984c26009fc... | a |
| 5f8db58210e199707b3984c26009fc... | double |
| 5f8db58210e199707b3984c26009fc... | storey |
| 5f8db58210e199707b3984c26009fc... | home |
| 5f8db58210e199707b3984c26009fc... | surrounded |
| 5f8db58210e199707b3984c26009fc... | by |
| 5f8db58210e199707b3984c26009fc... | splendid |
| 5f8db58210e199707b3984c26009fc... | views |
| 5f8db58210e199707b3984c26009fc... | of |
| 5f8db58210e199707b3984c26009fc... | the |

# Case Study: Property

**Step 4:** Remove stopwords and `GROUP BY` count words per `ad_id`

```
WITH cte_one AS(
    SELECT ad_id,
        REGEXP_REPLACE(LOWER(description), '[^A-Za-z ]', '', 'g') AS
    FROM gumtree_description
), cte_two AS(
    SELECT ad_id,
        TRIM(REGEXP_SPLIT_TO_TABLE(LOWER(description), '\s+')) AS to
    FROM cte_one
), cte_three AS(
    SELECT *
    FROM cte_two
    WHERE tokens NOT IN (SELECT DISTINCT word FROM stopwords)
), cte_four AS(
    SELECT ad_id,
        tokens,
        COUNT(*) as obs,
        ROW_NUMBER() OVER (
            PARTITION BY ad_id
            ORDER BY COUNT(*) DESC
        ) as rank
    FROM cte_three
    GROUP BY ad_id,
        tokens
)
SELECT * FROM cte_four LIMIT 10
```

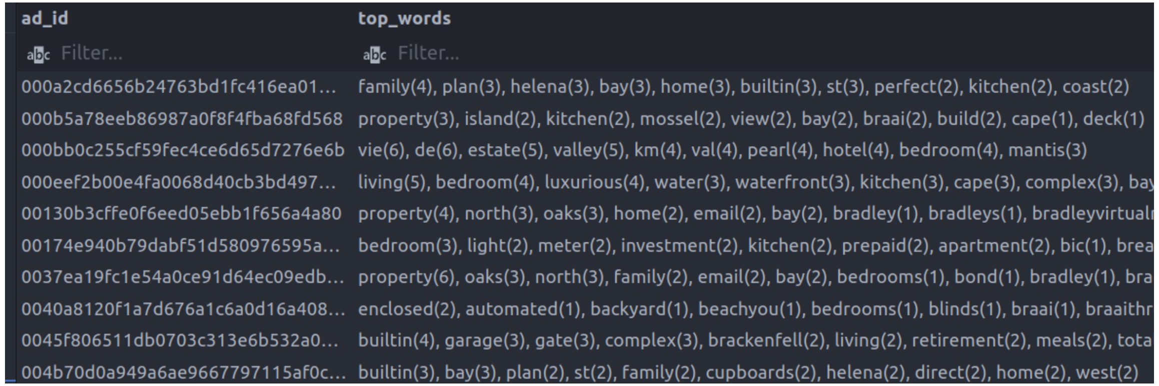


| ad_id | tokens | obs | rank |
|---|---|---|---|
| 000a2cd6656b24763bd1fc416ea01... | family | 4 | 1 |
| 000a2cd6656b24763bd1fc416ea01... | plan | 3 | 2 |
| 000a2cd6656b24763bd1fc416ea01... | helena | 3 | 3 |
| 000a2cd6656b24763bd1fc416ea01... | bay | 3 | 4 |
| 000a2cd6656b24763bd1fc416ea01... | home | 3 | 5 |
| 000a2cd6656b24763bd1fc416ea01... | builtin | 3 | 6 |
| 000a2cd6656b24763bd1fc416ea01... | st | 3 | 7 |
| 000a2cd6656b24763bd1fc416ea01... | perfect | 2 | 8 |
| 000a2cd6656b24763bd1fc416ea01... | kitchen | 2 | 9 |
| 000a2cd6656b24763bd1fc416ea01... | coast | 2 | 10 |
| 000a2cd6656b24763bd1fc416ea01... | bedroom | 2 | 11 |
| 000a2cd6656b24763bd1fc416ea01... | west | 2 | 12 |
| 000a2cd6656b24763bd1fc416ea01... | de | 1 | 13 |
| 000a2cd6656b24763bd1fc416ea01... | decades | 1 | 14 |
| 000a2cd6656b24763bd1fc416ea01... | detailsada | 1 | 15 |
| 000a2cd6656b24763bd1fc416ea01... | disappointedproperty | 1 | 16 |
| 000a2cd6656b24763bd1fc416ea01... | doors | 1 | 17 |

# Case Study: Property

**Final Step:** Concatenate the token and observation and filter where less than 10

```sql
WITH cte_one AS(
    SELECT ad_id,
        REGEXP_REPLACE(LOWER(description), '[^A-Za-z ]', '', 'g') AS description
    FROM gumtree_description
), cte_two AS(
    SELECT ad_id,
        TRIM(REGEXP_SPLIT_TO_TABLE(LOWER(description), '\s+')) AS tokens
    FROM cte_one
), cte_three AS(
    SELECT *
    FROM cte_two
    WHERE tokens NOT IN (SELECT DISTINCT word FROM stopwords)
), cte_four AS(
    SELECT ad_id,
        tokens,
        COUNT(*) as obs,
        ROW_NUMBER() OVER (
            PARTITION BY ad_id
            ORDER BY COUNT(*) DESC
        ) as rank
    FROM cte_three
    GROUP BY ad_id,
        tokens
), cte_five AS(
    SELECT ad_id,
        STRING_AGG(tokens || '(' || CAST(obs AS VARCHAR) || ')',', ') AS top_words
    FROM cte_four
    WHERE rank <= 10
    GROUP BY ad_id
)
SELECT * FROM cte_five LIMIT 10;
```



| ad_id | top_words |
|---|---|
| Filter... | Filter... |
| 000a2cd6656b24763bd1fc416ea01… | family(4), plan(3), helena(3), bay(3), home(3), builtin(3), st(3), perfect(2), kitchen(2), coast(2) |
| 000b5a78eeb86987a0f8f4fba68fd568 | property(3), island(2), kitchen(2), mossel(2), view(2), bay(2), braai(2), build(2), cape(1), deck(1) |
| 000bb0c255cf59fec4ce6d65d7276e6b | vie(6), de(6), estate(5), valley(5), km(4), val(4), pearl(4), hotel(4), bedroom(4), mantis(3) |
| 000eef2b00e4fa0068d40cb3bd497… | living(5), bedroom(4), luxurious(4), water(3), waterfront(3), kitchen(3), cape(3), complex(3), bay |
| 00130b3cffe0f6eed05ebb1f656a4a80 | property(4), north(3), oaks(3), home(2), email(2), bay(2), bradley(1), bradleys(1), bradleyvirtual |
| 00174e940b79dabf51d580976595a… | bedroom(3), light(2), meter(2), investment(2), kitchen(2), prepaid(2), apartment(2), bic(1), brea |
| 0037ea19fc1e54a0ce91d64ec09edb… | property(6), oaks(3), north(3), family(2), email(2), bay(2), bedrooms(1), bond(1), bradley(1), bra |
| 0040a8120f1a7d676a1c6a0d16a408… | enclosed(2), automated(1), backyard(1), beachyou(1), bedrooms(1), blinds(1), braai(1), braaithr |
| 0045f806511db0703c313e6b532a0… | builtin(4), garage(3), gate(3), complex(3), brackenfell(2), living(2), retirement(2), meals(2), tota |
| 004b70d0a949a6ae9667797115af0c… | builtin(3), bay(3), plan(2), st(2), family(2), cupboards(2), helena(2), direct(2), home(2), west(2) |

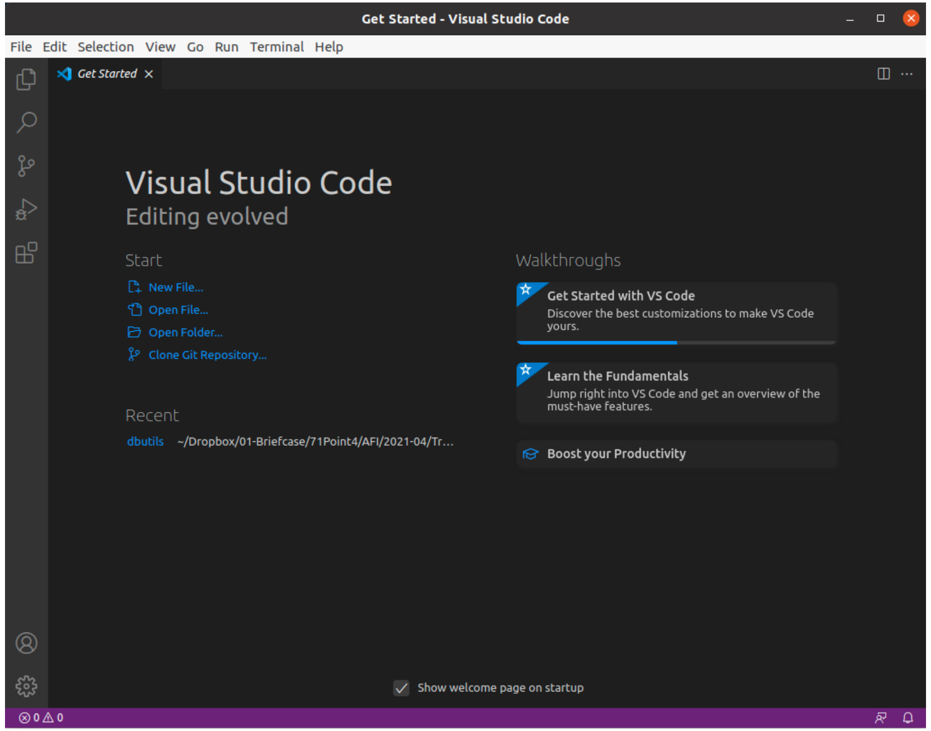
# VSCode 💻

# Learning to code in VSCode

> Why switch from RStudio to VSCode for SQL development?

The first few things we are gonna do in VSCode is:

- Interact with a remote server
- Connect to database on remote server
- Execute code and download results


Get Started - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Get Started

Visual Studio Code

Editing evolved

Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Recent
dbutils ~/Dropbox/01-Briefcase/71Point4/AFI/2021-04/Tr...
Walkthroughs

Get Started with VS Code

Discover the best customizations to make VS Code yours.

Learn the Fundamentals

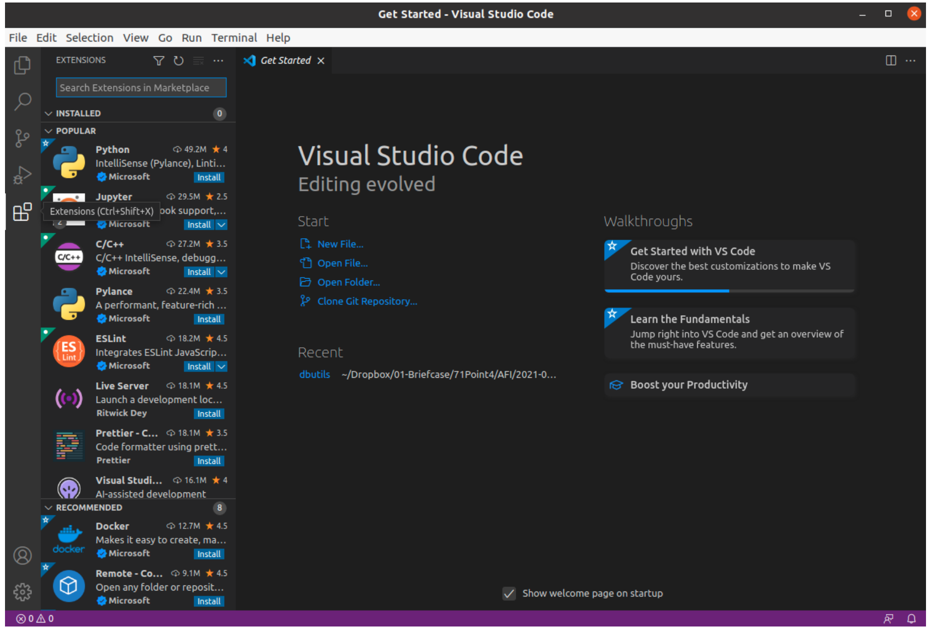
Jump right into VS Code and get an overview of the must-have features.

Boost your Productivity
Show welcome page on startup
0 0

# Installing the recommended Extension

Installing *Extensions* in VSCode is pretty straight forward. Just navigation to the search tab using GUI. Then search and install the following:

- R Extension for Visual Studio Code
- Spelling Checker for Visual Studio Code
- SQLTools



Get Started - Visual Studio Code
File
Edit
Selection
View
Go
Run
Terminal
Help
EXTENSIONS
Search Extensions in Marketplace
Extensions (Ctrl+Shift+X)
Get Started
Install
0
8

INSTALLED 0

POPULAR

Python 49.2M ★ 4
IntelliSense (Pylance), Linti…
Microsoft

Jupyter 29.5M ★ 2.5
…ook support,…
Microsoft

C/C++ 27.2M ★ 3.5
C/C++ IntelliSense, debugg…
Microsoft

Pylance 22.4M ★ 3.5
A performant, feature-rich …
Microsoft

ESLint 18.2M ★ 4.5
Integrates ESLint JavaScrip…
Microsoft

Live Server 18.1M ★ 4.5
Launch a development loc…
Ritwick Dey

Prettier - C… 18.1M ★ 3.5
Code formatter using prett…
Prettier

Visual Studi… 16.1M ★ 4
AI-assisted development

RECOMMENDED 8

Docker 12.7M ★ 4.5
Makes it easy to create, ma…
Microsoft

Remote - Co… 9.1M ★ 4.5
Open any folder or reposit…
Microsoft

Visual Studio Code
Editing evolved

Start
New File…
Open File…
Open Folder…
Clone Git Repository…

Recent
dbutils ~/Dropbox/01-Briefcase/71Point4/AFI/2021-0…

Walkthroughs

Get Started with VS Code
Discover the best customizations to make VS Code yours.

Learn the Fundamentals
Jump right into VS Code and get an overview of the must-have features.

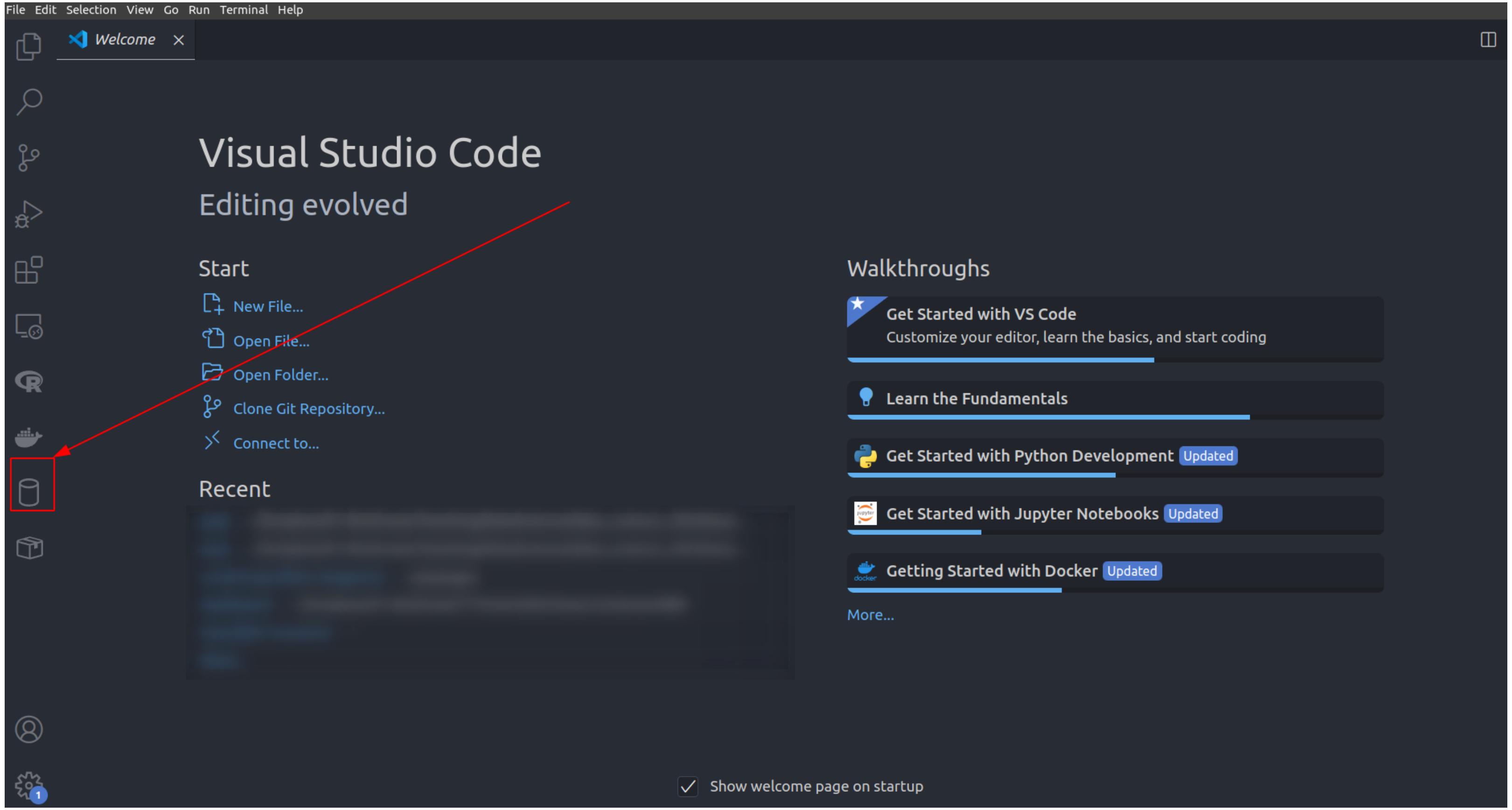
Boost your Productivity

☑ Show welcome page on startup

⊗ 0 ⚠ 0

# Connecting to DB

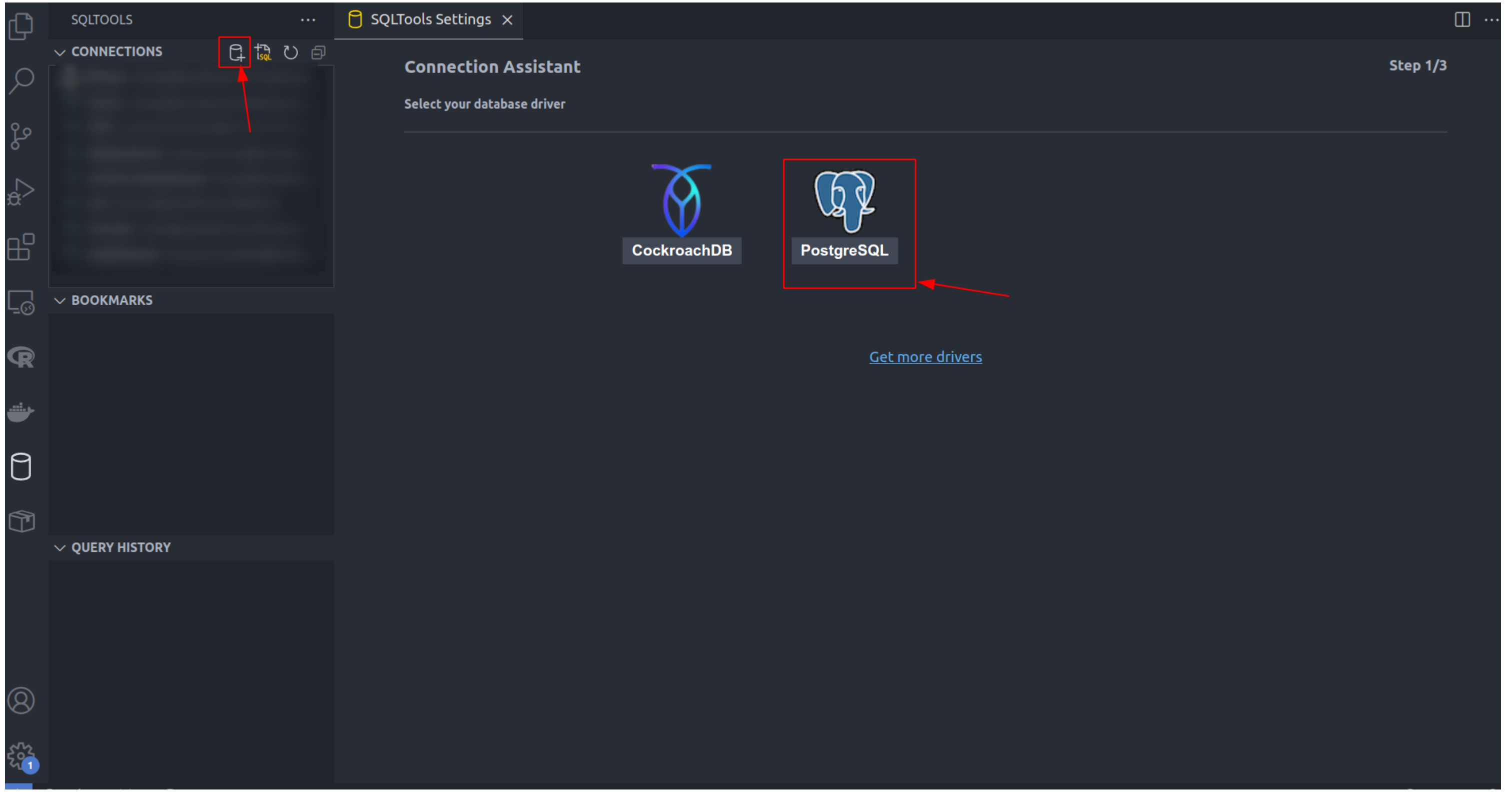
After installing VScode, you should see a `SQLTools` icon in the left-hand bar:


File Edit Selection View Go Run Terminal Help
Welcome
Visual Studio Code
Editing evolved
Start
New File...
Open File...
Open Folder...
Clone Git Repository...
Connect to...
Recent
Walkthroughs
Get Started with VS Code
Customize your editor, learn the basics, and start coding
Learn the Fundamentals
Get Started with Python Development Updated
Get Started with Jupyter Notebooks Updated
Getting Started with Docker Updated
More...
Show welcome page on startup

# Connecting to DB

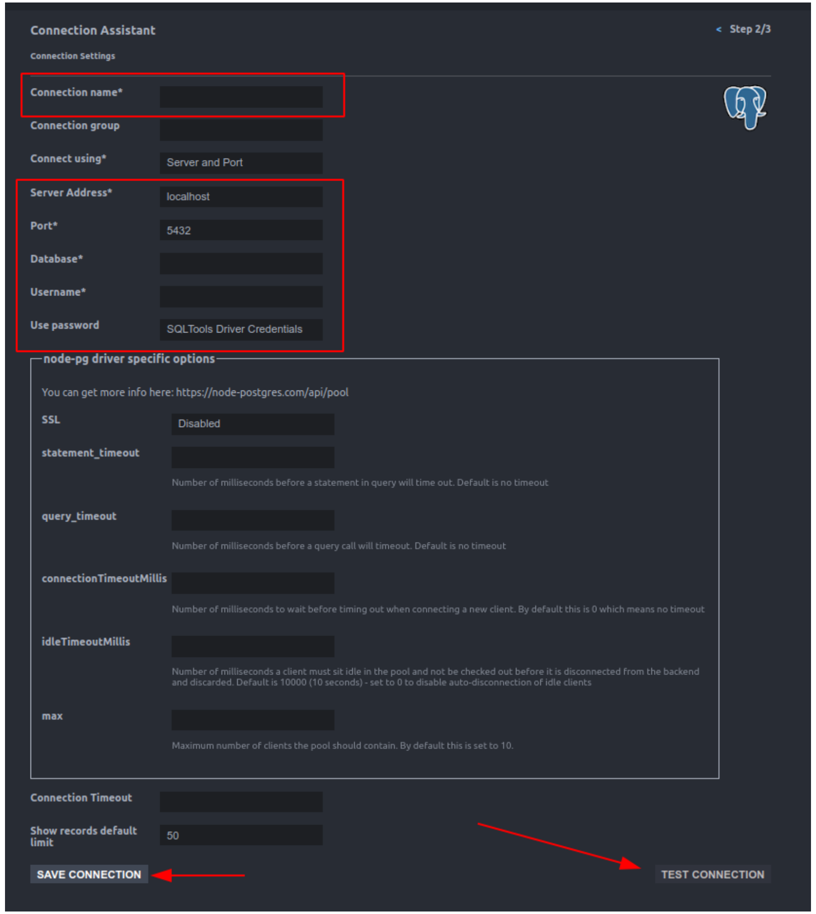
We can now add a new connection to a DB.

⚠️ If you do not see PostgreSQL, then just click on 'Get More Drivers' and search for 'SQLTools PostgreSQL/Cockroach Driver'.



# Connecting to DB

In the last step we going to fill in the connection strings (On the next slide):



Connection Assistant

Connection Settings

< Step 2/3

| | |
|---|---|
| Connection name* | |
| Connection group | |
| Connect using* | Server and Port |
| Server Address* | localhost |
| Port* | 5432 |
| Database* | |
| Username* | |
| Use password | SQLTools Driver Credentials |

node-pg driver specific options

You can get more info here: https://node-postgres.com/api/pool

| | |
|---|---|
| SSL | Disabled |
| statement_timeout | Number of milliseconds before a statement in query will time out. Default is no timeout |
| query_timeout | Number of milliseconds before a query call will timeout. Default is no timeout |
| connectionTimeoutMillis | Number of milliseconds to wait before timing out when connecting a new client. By default this is 0 which means no timeout |
| idleTimeoutMillis | Number of milliseconds a client must sit idle in the pool and not be checked out before it is disconnected from the backend and discarded. Default is 10000 (10 seconds) - set to 0 to disable auto-disconnection of idle clients |
| max | Maximum number of clients the pool should contain. By default this is set to 10. |

| | |
|---|---|
| Connection Timeout | |
| Show records default limit | 50 |