**SQL**

**database** is a collection of data about different objects, also called entities, each entity with a common set of properties.  
Examples:

* Drivers (name, licence number, licence expiration);
* Cars (model, year);
* Clients (name, address, phone, e-mail);

**DBMS** (Database Management System) is a software responsible to assist the maintenance and usage of databases (e.g., PostgreSQL, MS Access, MS SQL Server, MySQL, etc...)

Managing data is:

* 0. **storing it**
* a. **keep track** of the relationship between the data entities;
* b. **avoid redundancies**;
* c. manage **security** issues to protect private and sensitive information;
* d. have an error **recovery system** in place;
* e. be able to **retrieve data** in an efficient manner;
* f. have well **controlled** concurrent access;
* g. keep the data in a **consistent state**;
* h. enforce data **integrity**

Structure the data: **relational Model** (tabular data) based on set theory

Difference between **database** vs. **DBMS**

Database es a group of tables and DBMS is a program that manage that tables.

**What does in means:**

**Relational model**

Based on set theory. It represents the database as a collection of relations.

**Database**

It is a collection of tables and it’s relational.

**Tables**

Are relations that describe elements in rows and their attributes in columns.

Any operation you perform in a database will result in a table.

We will be always dealing with tables; (keep this in mind as it will be helpful when dealing with subqueries).

**Row**

An element of the table

**Column**

Represent a specific attribute of an entity.

**Domain**

The set of values a column can take. Essential to guaranty data integrity.

**Primary key**

Is a column (or smallest group of columns) that can identify each row as unique.

**Candidate key**

If there are more than one column that can identify the rows as unique, the manager of the database should choose one of them as the primary key, and the others would be known as candidate keys.

**Foreign key**

It is a column (or group of columns) of one table that refers to a primary key from another table.

It assists you to relate different entities.

**Aggregate functions**

It is a function where the value of multiple rows are grouped to calculate a statistic; it is similar to `summarise` in Tidyverse in R.

Most common aggregation functions: min(), max(), sum(), avg(), count(); also stddev(), variance(), corr()

Aggregation functions requires the processing of the whole table to obtain the desired aggregate value.

**Difference of WHERE vs. HAVING**

We use `where` as a condition to filter the rows from the raw data before doing a group by; while we use `having` to filter groups after we aggregate the data using group by.

*SELECT country*

*FROM projects*

***WHERE country <> ‘US’***

*GROUP BY country*

***HAVING COUNT(\*) > 10000;***

**Use parameter as query**

In a query, we can specify an identifier as a parameter to be read by using `**:**`

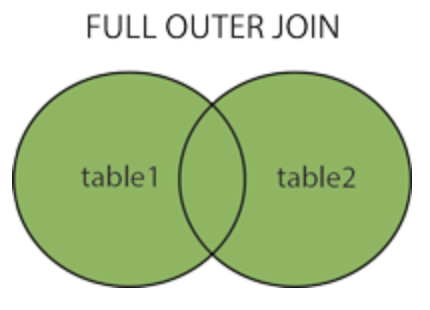
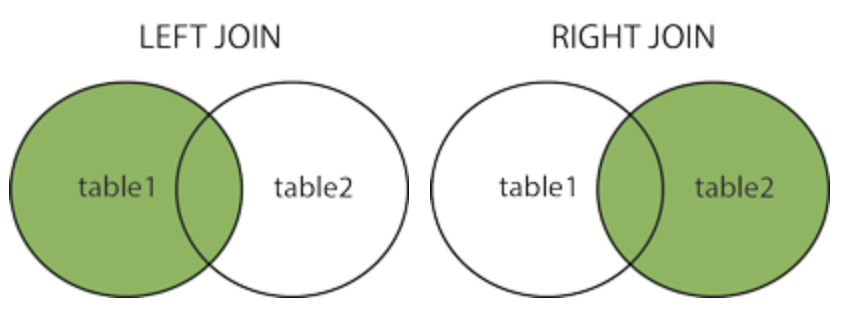
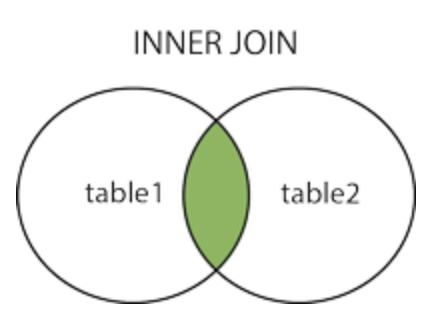
**Joins**

We use join to retrieve information stored across different tables that are connected by primary and/or foreign keys.

There are different types of joins that you might want to do.

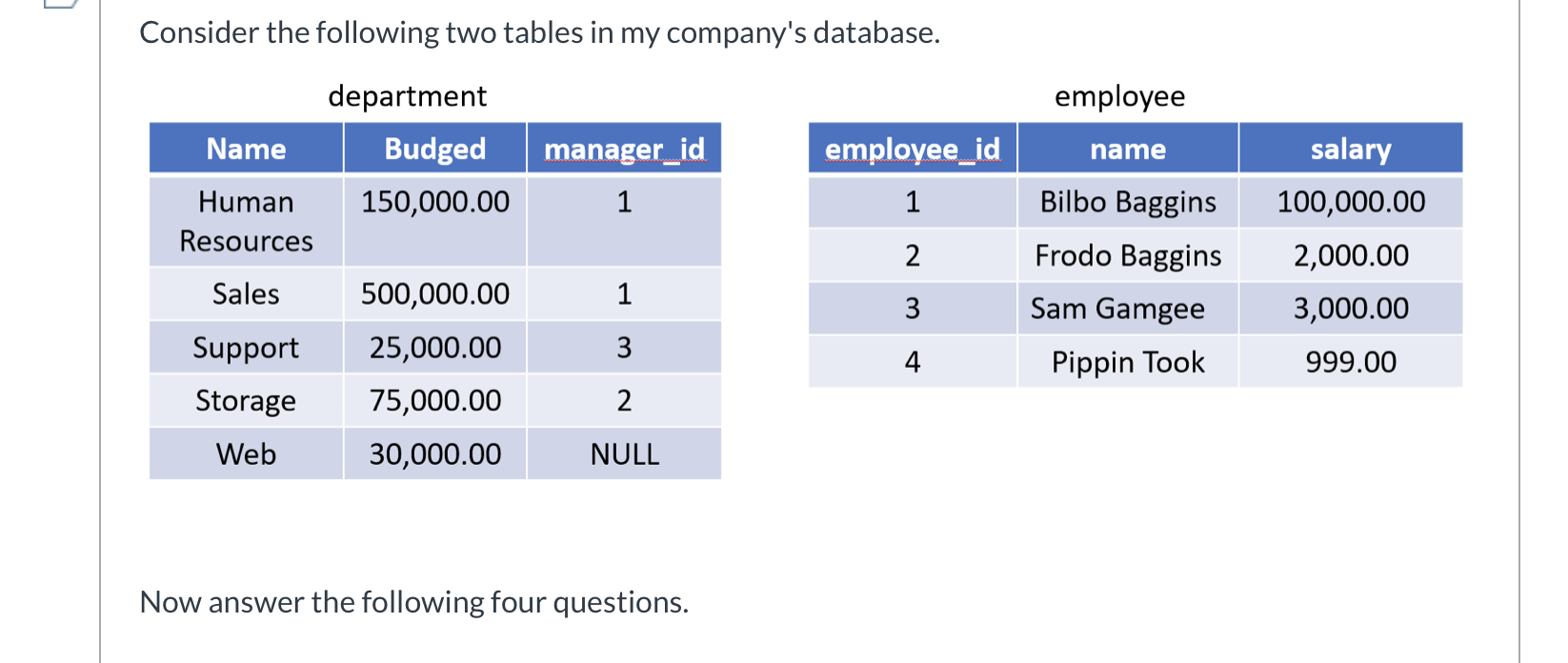
* *Cross join;*
* *Inner join;*
  + *Natural join;*
  + *Self join;*
* *Outer join*
  + *Left outer join;*
  + *Right outer join;*
  + *Full outer join;*

**Difference between joins**



**PostgreSQL RDBMS**

\l : all datasets  
\c database\_name : to connect to a database you can use  
\dt : Listing all tables in a database  
\d table\_name : Table descriptions  
\! command : To run a shell command  
\? : Want to know more?



TO CONECT TO SQL SHELL

|  |
| --- |
| r7k3-mds1.stat.ubc.ca  5432  NONE OR ANY DATABASE: drinks, kickstarter, imdb, etc.  cuspi  or use in case of emergency:  mds.ct6ghoz7smhy.us-east-1.rds.amazonaws.com  NONE OR ANY DATABASE: drinks, kickstarter, imdb, etc.  user: dsci513  pssw: dsci513 |

TO CONECT IN PYTHON

|  |
| --- |
| import os  import pandas as pd  from sqlalchemy import create\_engine |
| def execute\_print\_query(sql\_statement):  """  Returns the query formatted as a Pandas' dataframe.    Parameters  -----------  sql\_statement : string  The string containing the desired SQL statement to run.    Returns  --------  pandas.DataFrame  """  query = engine.execute(sql\_statement)  return pd.DataFrame(query.fetchall(), columns=query.keys()) |
| # THIS IS TO RUN FROM THE UNIVERSITY DATA BASES  #engine\_setup = 'postgres://user\_name:password@r7k3-mds1.stat.ubc.ca:5432/drinks'  engine\_setup = 'postgres://cuspi:' + os.getenv('POSTGRES\_PAT') + '@r7k3-mds1.stat.ubc.ca:5432/drinks'  engine = create\_engine(engine\_setup)  # ONLY USE IN CASE OF EMERGENCY  # EMERGENCY SERVER RUN THIS CELL IF YOU LOOSE CONNECTION  engine\_setup = 'postgres://dsci513:dsci513@mds.ct6ghoz7smhy.us-east-1.rds.amazonaws.com:5432/kickstarter'  engine = create\_engine(engine\_setup) |

COMANDS

|  |  |  |
| --- | --- | --- |
| SELECT | It is used to retrieve and manipulate data.  Never count on a pre-defined row order unless you specify it. | TYPE |
| FROM | Use distinct to remove duplicates. | ORDER |
| JOIN |  | ORDER |
| WHERE | It is used as a condition that retrieves only the rows for which this condition is true.   | **Operator** | **Consition** | | --- | --- | | [<, <=, =, >=, >](https://www.postgresql.org/docs/9.1/functions-comparison.html) | Ordinal comparisons | | [LIKE, ILIKE, SIMILAR TO](https://www.postgresql.org/docs/11/functions-matching.html#FUNCTIONS-LIKE) | Pattern matching | | [BETWEEN](https://www.postgresql.org/docs/9.1/functions-comparison.html) | Range filtering | | IN | Check belonginess (similar to %in% in R) | | IS NULL | Check if is NULL |   Also can use AND, OR, DISTINCT, NOT DISTINCT, BETWEEN, NOT BETWEEN | ORDER |
| GROUP BY | It is a filter that group the rows in base to a specific attribute of an entity.  GROUP BY groups all the NULL together. | ORDER |
| HAVING | It is a condition that retrieves the groups which the condition is true. | ORDER |
| ORDER BY | There is no particular row order in SELECT a select statement.  ORDER BY sort the results of your query in ascending (ASC) or descending (DESC) order.  ORDER BY NULL: treats NULL as the highest possible value.  We do not need to select the column we are using to order.  ORDER BY and DISTINCT are expensive operations (they rely on sorting), use only if you need to. | ORDER |
| LIMIT |  | ORDER |
| ; | ALWAYS END WITH SEMMICOLON | ORDER |
| ------------ | -------------------- | -------- |
| / | Default scape character | PATTERN |
| \_ and % | Pattern matching in LIKE, ILIKE and SIMILAR TO (~) \_ 1 any character, % for 0 or more any characters | PATTERN |
| || | String Concatenation | OPERATOR |
| SUBSTRING | *SELECT SUBSTRING('VIC\_CUSPINERA' FROM 5 FOR 10);* | OPERATOR |
| LENGTH | *SELECT LENGTH('VICTOR');* | OPERATOR |
| EXTRACT | *SELECT EXTRACT(year FROM to\_timestamp(deadline) )*  *FROM projects*  *LIMIT 10;*  It is used for dates and time: year, second, etc. | OPERATOR |
| CASE | With case we can check for several conditions and return only one value (the first true condition) | OPERATOR |
| COALESCE | Returns the first non-NULL argument. Usually used to "rename" NULL values for printing. | OPERATOR |
| NULLIF | NULLIF compares the arguments, if they are equal, returns NULL.  For example, you can change a user specified values for NULL to the actual NULL | OPERATOR |
| MAX()  MIN()  AVG()  SUM()  COUNT() | Most common aggregation functions.  Also: stddev(), variance(), corr()  Example:  *SELECT MIN(runtime), MAX(runtime)*  *FROM titles\_basics;*  You cannot use nested aggregate functions;  You cannot mix aggregate functions, with column names in the SELECT clause, unless you're using GROUP BY. | AGGREGATE FUNCTIONS |
|  |  |  |
|  |  |  |
|  |  |  |
| IS NOT NULL |  |  |
| PARTITION BY | AVG(salary) OVER (PARTITION BY person\_id) as avg\_salary |  |
| DISTINCT | SELECT can return duplicate rows, unless otherwise specified with DISTINCT  Treats NULL as if they were equal to each other.  DISTINCT eliminates only rows that have the same values in all the selected columns simultaneously.  ORDER BY and DISTINCT are expensive operations (they rely on sorting), use only if you need to.  You can use DISTINCT to remove duplicates when using aggregate functions, e.g., COUNT(DISTINCT id). |  |
| SELECT \* | is not recommended for production code (use for exploration only) |  |
| AS | variable\_name AS name: Creating a column alias with AS does not modify the column name in the table |  |
| IS NULL | Use IS NULL to match NULL, do not try using equality. A NULL is not equal to another NULL. |  |
| LIKE | You can use LIKE to match patterns. When using LIKE, \_ matches any character and % matches any sequence of 0 or more character.  IT IS **SENSITIVE** CASE (DIFFER FROM LOWER VS. UPPER CASE)  *WHERE name LIKE 'a%\_%\_%o'* | LOOK FOR |
| ILIKE | (not part of SQL standard) works similarly, however is case-insensitive.  IT IS **INSENSITIVE** TO LOWER VS. UPPER CASE | LOOK FOR |
| SIMILAR TO | It interprets the pattern using the SQL standard's definition of a regular expression. SQL regular expressions are a curious cross between LIKE notation and common regular expression notation... but it is not the normal regex. | LOOK FOR |
| ~ or POSIX | WORKS AS Real REGEX  *WHERE name LIKE 'a.\*?.\*?.\*o'* | LOOK FOR |
| BETWEEN | TRUE if the operand is **within the range** of comparisons  The values can be numbers, text, or dates.  INCLUDES BOUNDARIES | LOGICAL |
| ALL | TRUE if **all** of the subquery values **meet the condition** | LOGICAL |
| AND | TRUE if **all** **the conditions** separated by AND is TRUE | LOGICAL |
| ANY | TRUE **if any of the subquery** values **meet the condition** | LOGICAL |
| EXISTS | TRUE **if the subquery returns one or more records** | LOGICAL |
| IN | It **allows** you to **specify multiple values in a WHERE** clause.  TRUE if the operand is equal to one of a list of expressions  Like looking into a list in Python | LOGICAL |
| NOT | Displays a record if the **condition is NOT TRUE** | LOGICAL |
| OR | TRUE if **any of the conditions** separated by OR is TRUE | LOGICAL |
| SOME | TRUE if **any of the subquery values meet the condition** | LOGICAL |
| INNER JOIN  (also JOIN) | Return the rows that have matching values in two tables | JOINS |
| LEFT JOIN / RIGHT JOIN  (OUTER) | Return all the rows from the left table (right table), and the matched records from the right table (left table). If the rows in the right (left) table doesn’t match, it sets NULL values in all columns of the table that have no information. | JOINS |
| FULL JOIN (OUTER) | Return all the rows that match in either left or right table. For all the rows that doesn’t match between tables, the full join sets NULL values for all the attributes (columns) that lack of matching rows. It only save one row if it matches between tables. | JOINS |
| SELF JOIN | It is like an INNER JOIN which one table is matching itself, and it is used to compare rows within the same table. |  |
| CROSS JOIN | It combines all the rows of the first and second tables any matching condition in the join clause. | JOINS |
| NATURAL JOIN | It is a join that create an implicit join based on the same column names in the joined tables.  It is a join that returns the rows that have matching column names in two tables.  A natural join can be an inner join, left join, or right join. If you do not specify it uses the INNER JOIN by default.  *SELECT \**  *FROM T1*  *NATURAL [INNER, LEFT, RIGHT] JOIN T2;* |  |
| EXISTS | Returns TRUE if the following subquery is not empty |  |
| UNION | combines all rows from both selects into the same table. It removes duplicates. (use UNION ALL to keep duplicates).  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year <= 1964*  ***UNION***  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year >= 1964;*  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year <= 1964*  ***UNION ALL***  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year >= 1964;* |  |
| INTERSECT | combines only rows that are present in both tables. It removes duplicates. (use INTERSECT ALL to keep duplicates).  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year <= 1964*  *INTERSECT*  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year >= 1964;* |  |
| EXCEPT | only keeps the rows that are in the first table but are not in the second table.  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year <= 1964*  *EXCEPT*  *SELECT name, birth\_year*  *FROM people p*  *WHERE birth\_year >= 1964;* |  |
| WHIT… AS | Creates temporary **tables** that only **exist for one query** and stores partial results.  The partial tables generated in the WITH clause will **only exist until the main SELECT statement is executed.**  If you don't include the main SELECT statement with the WITH clause, you will get an error.  *WITH average\_runtime AS*  *(SELECT format, AVG(runtime)*  *FROM titles\_basics*  *GROUP BY format)*  *SELECT format, tconst, primary\_title title, runtime, ROUND(avg, 2)*  *FROM titles\_basics*  *NATURAL JOIN average\_runtime*  *WHERE runtime IS NOT NULL* |  |
| VIEWS | Are stored select statements (QUERIES);  Virtual tables which the **data is not physically stored but computed as needed**.  Does NOT return anything, just saves the SELECT statement.  *CREATE [TEMP] VIEW view\_name[(view\_columns\_names)] AS*  *SELECT STATEMENT;* |  |

SELECT DISTINCT n.name, \*

FROM names\_basics AS n

INNER JOIN names\_professions AS p ON n.nconst = p.nconst

WHERE n.name SIMILAR TO '%Pitt%' AND n.birth\_year IS NOT NULL AND n.death\_year IS NULL AND p.profession ~ 'act.\*'

ORDER BY n.name DESC

LIMIT 10;

**SELECT**

It is used to retrieve and manipulate data;

|  |
| --- |
| SELECT id, name, country, usd\_pledged, goal, state  FROM projects  LIMIT 4; |
| # TO RUN UN PYTHON YOU SHOULD USE THE NEXT STRUCTURE  sql\_statement = \  """  SELECT id, name, country, usd\_pledged, goal, state  FROM projects  LIMIT 4;  """  execute\_print\_query(sql\_statement) |

**DISTINCT**

Use distinct to remove duplicates.

|  |
| --- |
| --Database: Kickstarter  SELECT DISTINCT state  FROM projects;  SELECT \*  FROM projects  WHERE usd\_pledged >= 6500  AND state ='successful'; |

**WHERE clause**

A (boolean) condition that retrieves only the rows for which the condition is true.

| **Operator** | **Consition** |
| --- | --- |
| [<, <=, =, >=, >](https://www.postgresql.org/docs/9.1/functions-comparison.html) | Ordinal comparisons |
| [LIKE, ILIKE, SIMILAR TO](https://www.postgresql.org/docs/11/functions-matching.html#FUNCTIONS-LIKE) | Pattern matching |
| [BETWEEN](https://www.postgresql.org/docs/9.1/functions-comparison.html) | Range filtering |
| IN | Check belonginess (similar to %in% in R) |
| IS NULL | Check if is NULL |

Also can use AND, OR, DISTINCT, NOT DISTINCT, BETWEEN, NOT BETWEEN

|  |
| --- |
| SELECT name, birth\_year  FROM names\_basics  WHERE name = 'Brad Pitt'; |

**LIKE**

You can use LIKE to match patterns. When using LIKE, \_ matches any character and % matches any sequence of 0 or more character.

IT IS SENSITIVE CASE (DIFFER FROM LOWER VS. UPPER CASE)

**ILIKE**

(not part of SQL standard) works similarly, however is case-insensitive.

IT IS **IN**SENSITIVE TO LOWER VS. UPPER CASE

|  |
| --- |
| SELECT name, birth\_year  FROM names\_basics  WHERE name ILIKE '%Pitt%'; |

**CASE**

|  |
| --- |
| SELECT  runtime,  primary\_title,  CASE  WHEN runtime < 30 THEN 'very short'  WHEN runtime > 120 THEN 'very long'  ELSE 'usual length'  END AS "length\_class"  FROM titles\_basics  WHERE runtime IS NOT NULL  LIMIT 50; |

## **COALESCE**

Returns the first non-NULL argument. Usually used to "rename" NULL values for printing.

|  |
| --- |
| SELECT  primary\_title,  **COALESCE(runtime, -1) AS "runtime"**  FROM titles\_basics  LIMIT 10; |

## **NULLIF**

NULLIF compares the arguments, if they are equal, returns NULL.  
For example, you can change a user specified values for NULL to the actual NULL

|  |
| --- |
| SELECT  primary\_title,  COALESCE(runtime, -1) AS "runtime",  **NULLIF(runtime, -1) AS "back\_to\_null"**  FROM titles\_basics  LIMIT 10; |

**Using arguments for queries**

In a query, we can specify an identifier as a parameter to be read by using `**:**`

*SELECT :my\_column*

*FROM titles\_basics*

*LIMIT 10;*

| **#** | **Date** | **Day** | **Topic** | **Resource materials** |
| --- | --- | --- | --- | --- |
| 1 | 2019-11-19 | Tue | **Course Overview:**  - Define database  - DBMS  - Why database? (vs file system)   **Familiarizing with PostgreSQL and SQL:**  - Create a database;  - List databases and tables;  - The SELECT, FROM and WHERE clauses;  - AS: Aliases (tables and columns);  - DISTINCT: removing duplicates;   - ORDER BY: sorting;  - SQL's logical operators;  - Scripts (including passing parameters);   - **The Relational Model:**  - Entity, Schemas and Tables;  - Primary key;  - Foreign Key; | - [Some useful POSTGRES commands](http://www.postgresqltutorial.com/psql-commands/)  -SELECT statement:[[1]](http://www.postgresqltutorial.com/postgresql-select/) [[2]](https://www.w3schools.com/sql/sql_select.asp)  - WHERE clause: [[1]](http://www.postgresqltutorial.com/postgresql-where/) [[2]](https://www.w3schools.com/sql/sql_where.asp)  - POSTGRES Logical operators:[[1]](https://www.postgresql.org/docs/11/functions-logical.html)  - Aliases: [[1]](http://www.postgresqltutorial.com/postgresql-alias/) [[2]](https://www.w3schools.com/sql/sql_alias.asp)  - ORDER BY clause: [[1]](http://www.postgresqltutorial.com/postgresql-order-by/) [[2]](https://www.w3schools.com/sql/sql_orderby.asp)  - SELECT DISTINCT clause: [[1]](http://www.postgresqltutorial.com/postgresql-select-distinct/)[[2]](https://www.w3schools.com/sql/sql_distinct.asp)  - Queries with parameters (see [SQL Interpolation](https://www.postgresql.org/docs/11/app-psql.html#APP-PSQL-INTERPOLATION)) |
| 2 | 2019-11-21 | Thur | **More on basic querying:**   - LIKE, BETWEEN and IN for filtering;   - Common operations on numbers, string and dates;  - CASE, COALESCE, NULLIF;   **Aggregation functions:**  - MIN(), MAX(), SUM(), AVG(), COUNT(), [and others](https://www.postgresql.org/docs/11/functions-aggregate.html);   - GROUP BY and HAVING;    **Accesing database from applications:**  - Performing queries from python and R; | - LIKE: [[1]](http://www.postgresqltutorial.com/postgresql-like/) [[2]](https://www.w3schools.com/sql/sql_like.asp) and more on [pattern matching](https://www.postgresql.org/docs/9.1/functions-matching.html)  - BETWEEN: [[1]](http://www.postgresqltutorial.com/postgresql-between/) [[2]](https://www.w3schools.com/sql/sql_between.asp)  - IN: [[1]](http://www.postgresqltutorial.com/postgresql-in/) [[2]](https://www.w3schools.com/sql/sql_in.asp)  - CASE [[1]](http://www.postgresqltutorial.com/postgresql-case/) [[2]](https://www.w3schools.com/sql/sql_case.asp), COALESCE [[1]](http://www.postgresqltutorial.com/postgresql-coalesce/), and NULLIF [[1]](http://www.postgresqltutorial.com/postgresql-nullif/): [docs](http://www.postgresqltutorial.com/postgresql-nullif/)   - Aggregate Functions: [[1]](http://www.postgresqltutorial.com/postgresql-aggregate-functions/) [[2]](https://www.w3schools.com/sql/sql_count_avg_sum.asp) [[3]](https://www.w3schools.com/sql/sql_min_max.asp) [[docs]](https://www.postgresql.org/docs/11/functions-aggregate.html)  - GROUP BY: [[1]](http://www.postgresqltutorial.com/postgresql-group-by/) [[2]](https://www.w3schools.com/sql/sql_groupby.asp)  - HAVING: [[1]](http://www.postgresqltutorial.com/postgresql-group-by/) [[2]](https://www.w3schools.com/sql/sql_groupby.asp)   - [Intro to dbplyr](https://cran.r-project.org/web/packages/dbplyr/vignettes/dbplyr.html)  - [Basic Usage SQLAlchemy](https://docs.sqlalchemy.org/en/13/core/connections.html#basic-usage) |
| 3 | 2019-12-03 | Tue | **Joins:**  - CROSS;  - NATURAL;  - INNER;  - OUTER;   **Set operations:**  - UNION;  - INTERSECT;  - EXCEPT; | - CROSS JOIN: [[1]](http://www.postgresqltutorial.com/postgresql-cross-join/)  - INNER JOIN: [[1]](http://www.postgresqltutorial.com/postgresql-inner-join/) [[2]](https://www.w3schools.com/sql/sql_join_inner.asp)  - SELF-JOIN: [[1]](http://www.postgresqltutorial.com/postgresql-self-join/) [[2]](https://www.w3schools.com/sql/sql_join_self.asp)  - NATURAL JOINS: [[1]](http://www.postgresqltutorial.com/postgresql-natural-join/)  - LEFT JOIN: [[1]](http://www.postgresqltutorial.com/postgresql-left-join/) [[2]](https://www.w3schools.com/sql/sql_join_left.asp)  - RIGHT JOIN: [[1]](http://www.sqlservertutorial.net/sql-server-basics/sql-server-right-join/) [[2]](https://www.w3schools.com/sql/sql_join_right.asp)  - FULL JOIN: [[1]](http://www.postgresqltutorial.com/postgresql-full-outer-join/) [[2]](https://www.w3schools.com/sql/sql_join_full.asp)  - [More on JOINS](http://www.postgresqltutorial.com/postgresql-joins/)   - UNION: [[1]](http://www.postgresqltutorial.com/postgresql-union/) [[2]](https://www.w3schools.com/sql/sql_union.asp)  - INTERSECT: [[1]](http://www.postgresqltutorial.com/postgresql-intersect/)  - EXCEPT: [[1]](http://www.postgresqltutorial.com/postgresql-tutorial/postgresql-except/)  - More on combining queries: [docs](https://www.postgresql.org/docs/9.4/queries-union.html) |
| 4 | 2019-12-05 | Thur | **Subqueries:**  - IN, ALL, ANY, and EXISTS;   - Correlated subqueries;  - Aggregate expressions in the WHERE clause (nested aggregation);  - WITH clause;   **Views:**  - Creating views;  - Retrieving data through views;  - Manipulating data through views;  - Deleting views;   **Window functions;** | -- ALL, ANY: [[1]](http://www.postgresqltutorial.com/postgresql-all/) [[2]](http://www.postgresqltutorial.com/postgresql-any/) [[3]](https://www.w3schools.com/sql/sql_any_all.asp)  - EXISTS: [[1]](http://www.postgresqltutorial.com/postgresql-exists/) [[2]](https://www.w3schools.com/sql/sql_exists.asp)  - WITH clause (recursive WITH is out of the scope): [[1]](https://www.tutorialspoint.com/postgresql/postgresql_with_clause.htm) [[docs]](https://www.postgresql.org/docs/9.1/queries-with.html)  - [Subqueries expressions docs](https://www.postgresql.org/docs/11/functions-subquery.html)  - [More on subqueries](https://www.w3resource.com/PostgreSQL/postgresql-subqueries.php)   Views:[[1]](http://www.postgresqltutorial.com/postgresql-views/)[[2]](http://www.postgresqltutorial.com/managing-postgresql-views/) [docs](https://www.postgresql.org/docs/11/sql-createview.html)    [Window funtions](http://www.postgresqltutorial.com/postgresql-window-function/) |