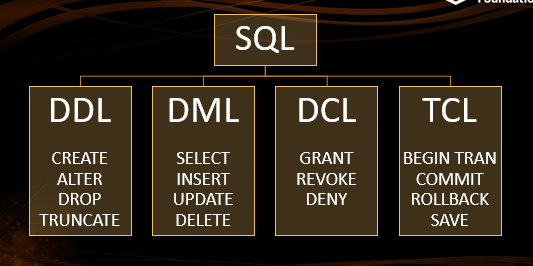
**PostgreSQL master password – 1234**

**PostgreSQL db password – TyCWJVDYP123**

**Forgotten password issue:** [**http://www.homebrewandtechnology.com/blog/graphicallychangepostgresadminpassword#TOC-Using-pgAdmin-to-Graphically-Reset-the-Password**](http://www.homebrewandtechnology.com/blog/graphicallychangepostgresadminpassword#TOC-Using-pgAdmin-to-Graphically-Reset-the-Password)

**SQL is a very large and powerful language, which is logically divided into 4 sections/categories:**



CRUD – Create (CREATE, INSERT), Read/Retrieve (SELECT), Update (UPDATE), Delete (DELETE, DROP)

Data Definition Language – for creating and dropping a DB

Data Manipulation Language – for accessing a DB

Data Control Language – for administering a DB

Transaction Control Language – for managing and controlling the transactions in a database to maintain consistency

**Database Table Elements/ SQL Terminology:**

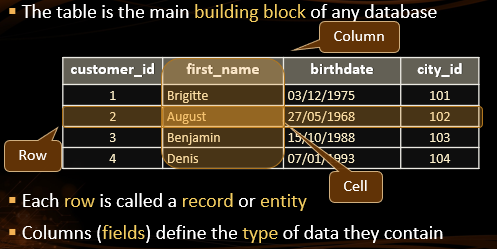


Table = relation

Columns = attributes/fields/set

Rows = Records/Entities/Tuples

Foreign key = Relationship (can be repeated, can be NULL)

Candidate Key/Primary Key (uniquely identify rows and cannot be NULL)

All values belonging to a particular attribute/column are from the same **domain**

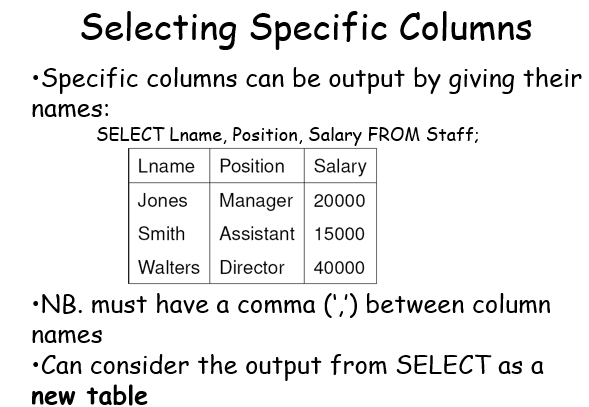
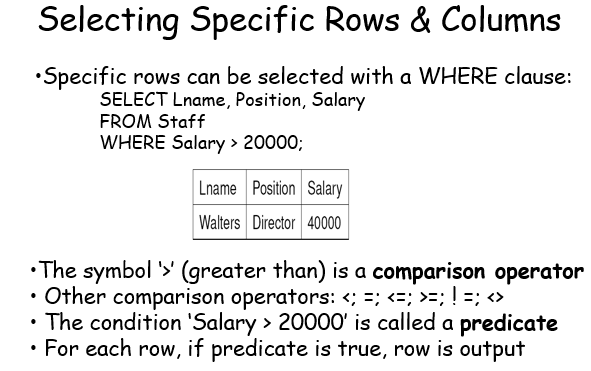
**SQL Syntax:**

* By convention, keywords are upper-case
* Text data is enclosed using single quotes (" ' ")
* Round brackets ( " ( " ) are used to group related items
* Commas (" , ") separate items in a list
* Statements are terminated with a semicolon (" ; ")

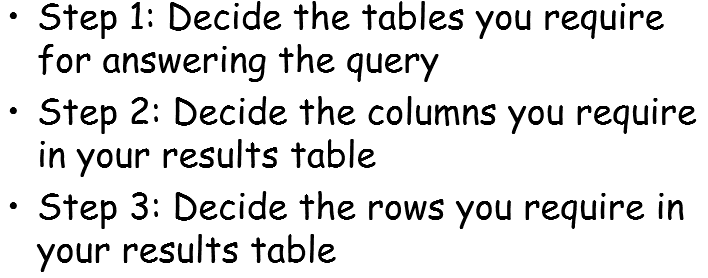
SELECT *target-list* (selecting the columns)

FROM *relation-list*  ( table name)

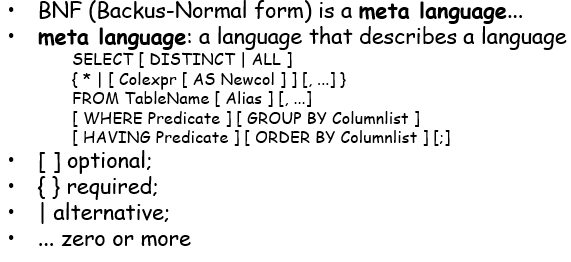
WHERE *qualification;* (selecting the rows)



**Write SELECT Queries in 3 Septs:**



**Describing SQL Syntax Using BNF Notation:**



* Sometimes we want to re-format the output from SELECT:
  + E.g. for reports...
* Examples of additional processing:
  + Eliminate duplicates
  + Sort and/or group the results
  + Rename column headings
  + Calculate totals, averages, etc. (often called aggregates)
  + Combine results from different tables

The following are the clauses that appear in the ​ SELECT ​ statement:

● Select distinct rows by using DISTINCT operator.

● Sort rows by using the ORDER BY clause.

You can limit the number of rows returned using LIMIT (select the 5 most recent payments)

*SELECT \* FROM payment*

*WHERE amount > 0*

*ORDER BY payment\_date DESC*

*LIMIT 5;*

● Filter rows by using WHERE clause.

● Select rows based on various operators such as BETWEEN, IN and LIKE.

*SELECT \**

*FROM Staff*

*WHERE (Salary BETWEEN 10000 AND 20000) AND (Position IN ('Manager', 'Assistant')) AND*

*(Lname LIKE 'S%' OR Lname LIKE 'W\_\_\_\_');*

*'%' matches zero or more characters*

*'\_' matches exactly one character*

[*https://mariadb.com/kb/en/like/*](https://mariadb.com/kb/en/like/)

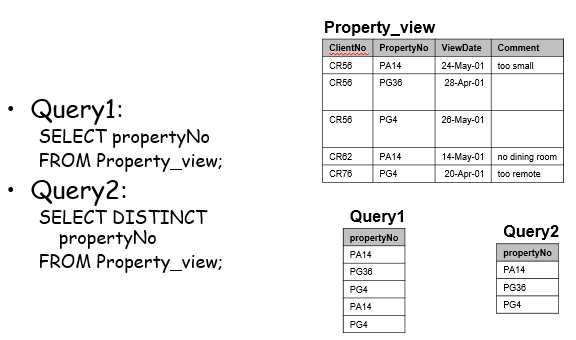
**NOTE: LIKE performs case-insensitive substring matching while WHERE column = “something” is giving a literal answer**

● Group rows into groups by using GROUP BY clause

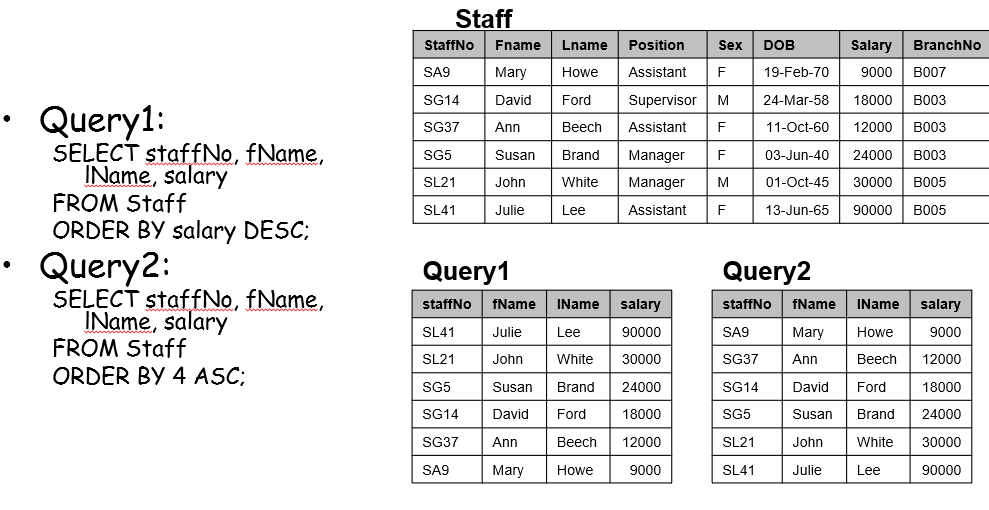
● Apply condition for groups by using HAVING clause.

● Join to another table by using INNER JOIN, LEFT JOIN, RIGHT JOIN clauses.

1. The keyword DISTINCT is used for avoiding duplicates:

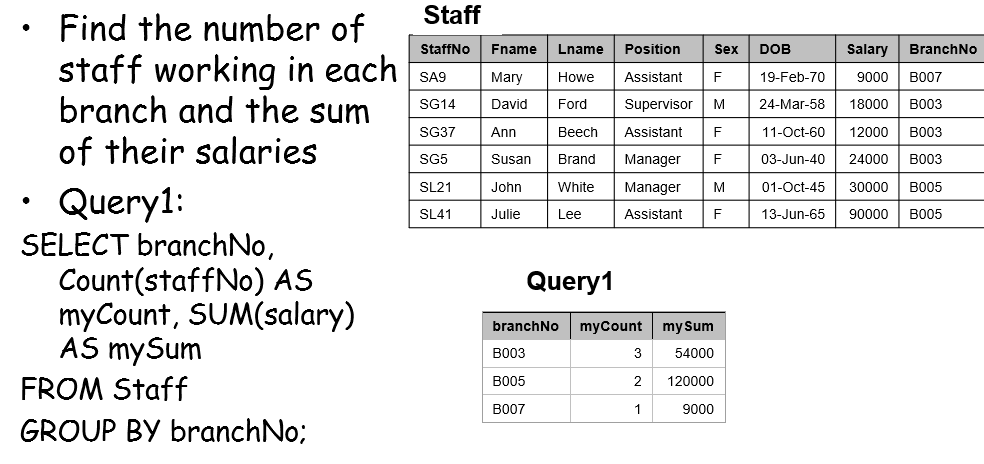


1. ORDER BY



! Notice 4 here – We start counting from 1 in Database Tables.

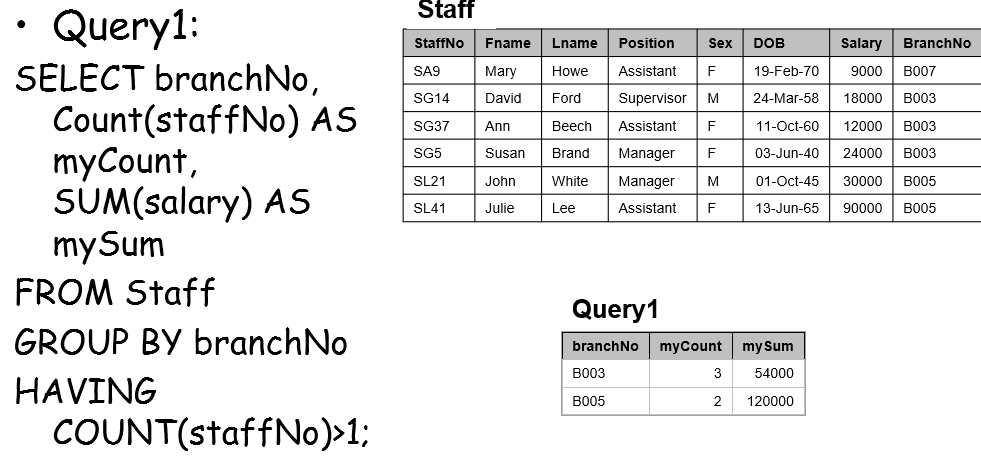
1. GROUP BY



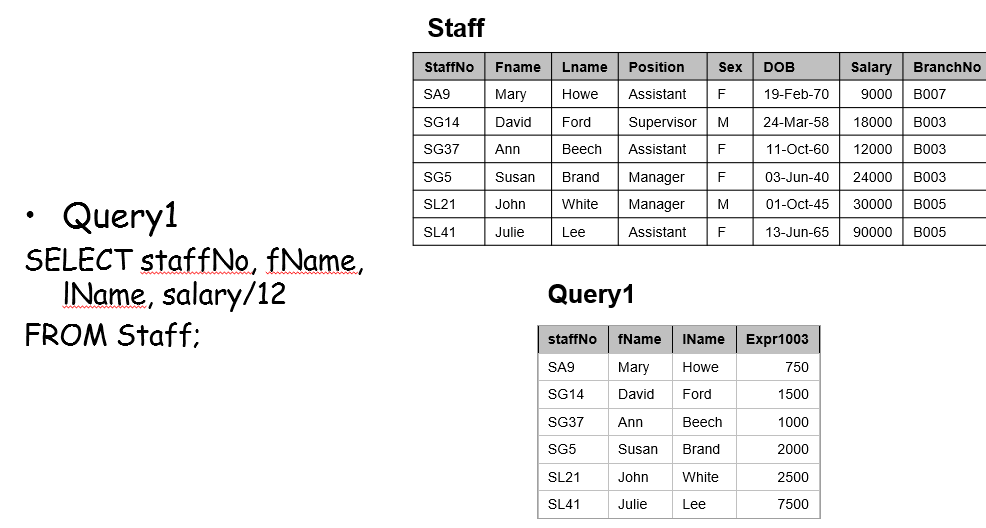
Firstly, grouping by branchNO and then calculating according the row.

* Aggregate functions help us to summarise the whole column(s) of data into one row.
* Sometimes we want to group data before applying aggregate functions
  + This gives us ‘subtotals’ rather than ‘overall total’
* GROUP BY is used to achieve that aggregating of columns per some category
* GROUP BY must appear right after a WHERE or FROM statement
* In the SELECT statement columns must either have an aggregate function or be in the GROUP BY call
* The simplest GROUP BY can be used as an alternative of SELECT DISTINCT

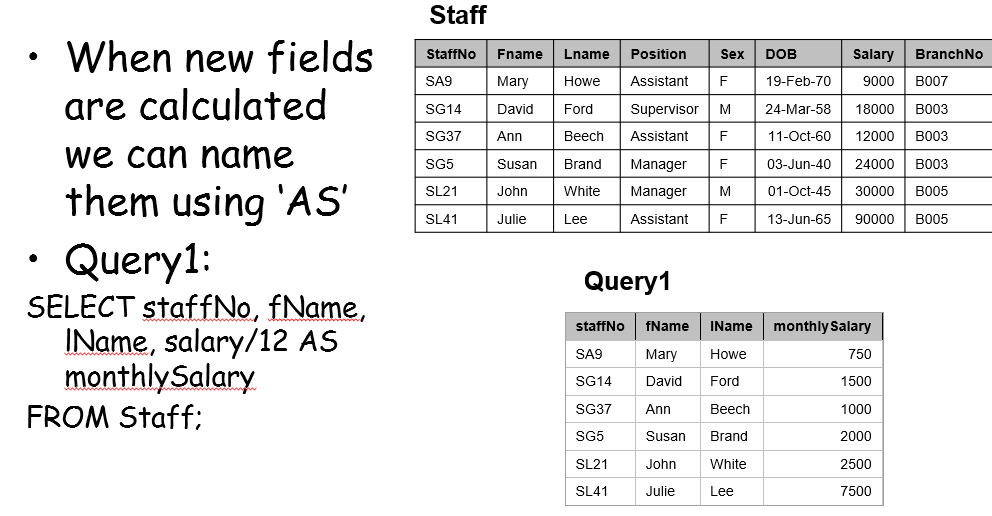
1. GROUP BY with HAVING - used to filter based on aggregate values (cannot use it without grouping before that, i.e. it appears just after GROUP BY)



1. Calculated Fields

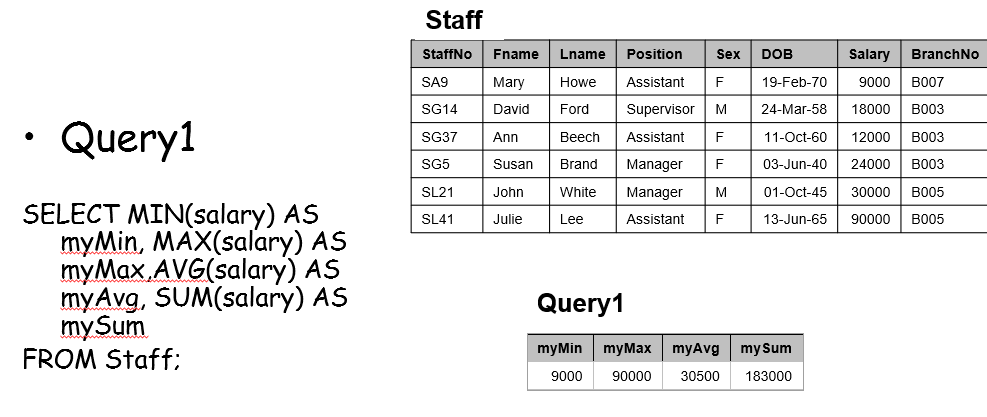


1. Renaming columns/attributes



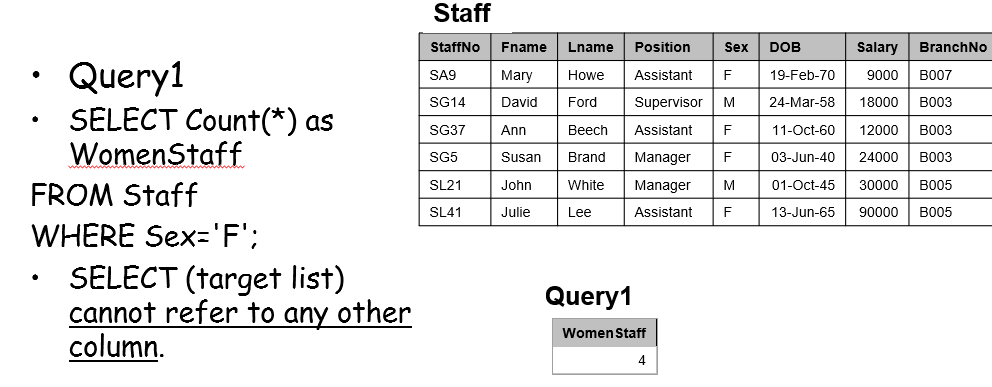
1. SQL Aggregate Functions

* We do not want to just retrieve data, but we also want to summarise data
* KEEP IN MIND THAT AGG.FUNCTIONS HAPPEN ONLY IN THE SELECT CLAUSE OR THE HAVING CLUASE
* Aggregate functions compute summarization (or aggregation) of data
* Aggregate functions - (SUM, AVG, MIN, MAX, COUNT)
* Note that AVG returns a float number, so use ROUND(AVG(), 2)

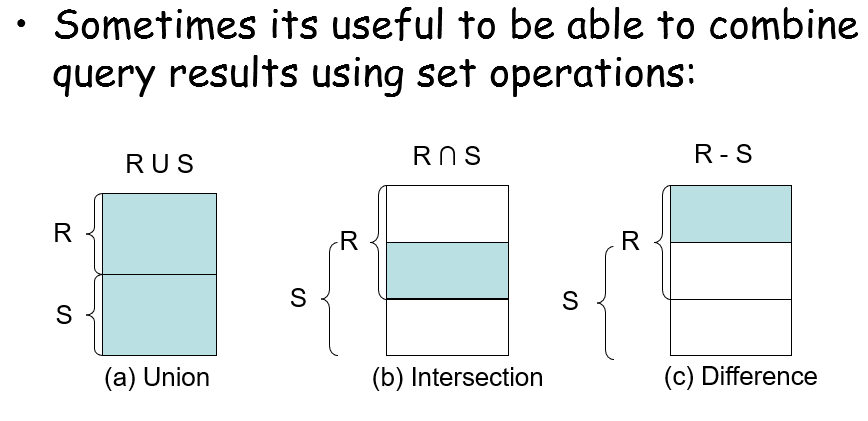


1. Count(\*) - counts the number of rows in a table

(Including the rows that have duplicates and nulls)



1. Combining Results Tables and Set Operations in SQL - UNION, INTERSECT, EXCEPT



* Syntax:

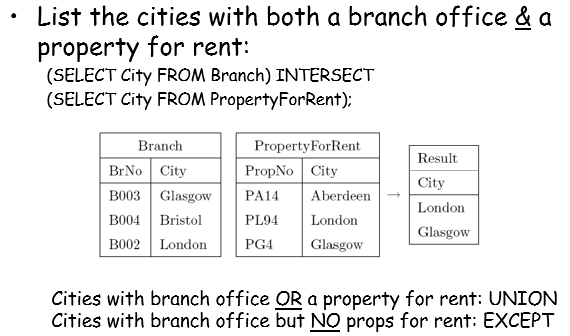
(SELECT ...) UNION (SELECT ...)

(SELECT ...) INTERSECT (SELECT ...)

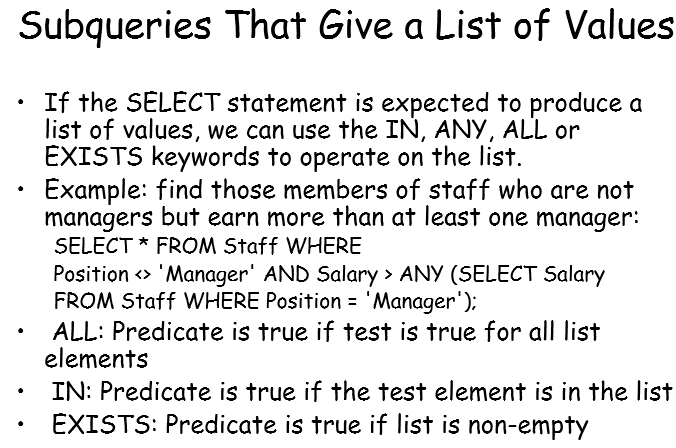
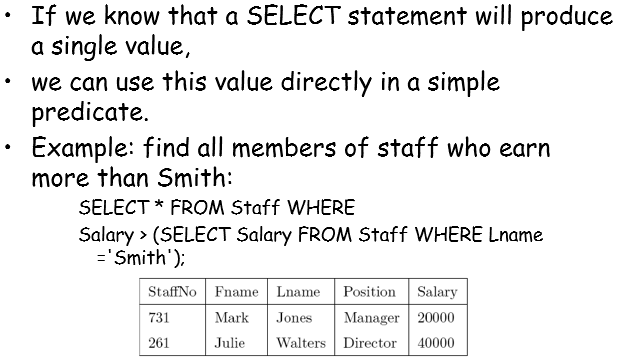
(SELECT ...) EXCEPT (SELECT ...)

* Some DBMSs use MINUS instead of EXCEPT
* For set operations, the tables must be union-compatible
  + i.e. have the same number and types of columns

Example:



1. Using Subqueries – Nested SELECTs (IN, ANY, ALL, EXISTS)

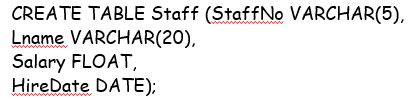
**Built-in Data Types (Domains)**

<https://www.postgresql.org/docs/9.5/datatype.html>

* CHARACTER (CHAR), CHARACTER VARYING (VARCHAR), TEXT
* NUMERIC, DECIMAL (DEC), INTEGER (INT), SMALLINT
* FLOAT, REAL, DOUBLE PRECISION
* DATE, TIME, TIMESTAMP
* BOOLEAN, BIT
* BINARY LARGE OBJECT, etc.
* Some types have an associated size - e.g. CHAR(5)

1. Create TABLE

* Used to define relational tables – defines the data type of each column
* Defines how data may be inserted and deleted



* VARCHAR, FLOAT, and DATE are examples of **domains**
* Domains specify type & range of allowed data values

1. Adding Data to a Table using INSERT

* General format:

INSERT INTO <TableName> (colname1, colname2, ...)

VALUES (value1, value2, ...);

* Example:

INSERT INTO Staff (StaffNo, Position, Salary, Lname)

VALUES (322, 'Assistant', 1500, 'Smith');

1. Modifying & Deleting Data (UPDATE / DELETE)

* Changing specific values in a table:
  + General Format:

UPDATE <TableName> SET <column> = <value>

WHERE <predicate>

* Example:

UPDATE Staff SET Salary = 1.05 \* Salary

WHERE Position = 'Director';

* Deleting specific rows:
  + General Format:

DELETE FROM <TableName>

WHERE <predicate>

* Example:

DELETE FROM Staff WHERE Fname IS NULL;

* Deleting all rows of a table:

DELETE FROM Staff;

* Completely removing a table is a DDL operation:

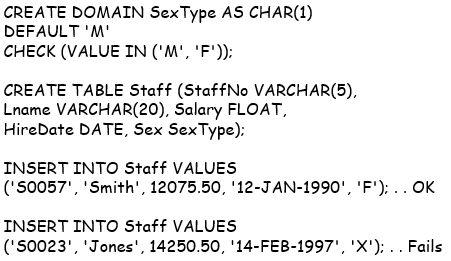
DROP TABLE Staff;

1. Deleting Domains

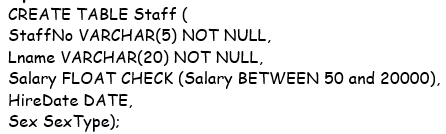
* DROP DOMAIN DomainName [RESTRICT | CASCADE]

1. User-Defined Domains

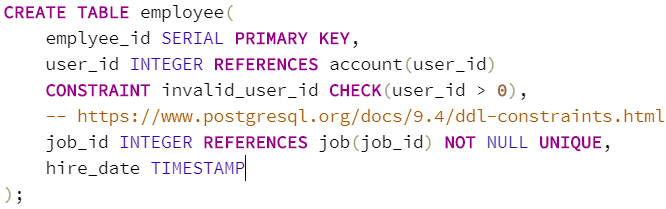
* Act as a constraint on allowed range of values



1. Required Data and Domain Constraints



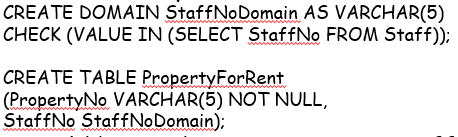
* StaffNo & Lname are **required -** may not be NULL
* The CHECK clause gives a **domain constraint** for Salary
* Updates & insertions **will fail** if constraints not satisfied



* NOT NULL constraint essentially ensures that a column cannot have a null value
* UNIQUE constraint ensures all values in a column are different
* <https://www.postgresql.org/docs/9.4/ddl-constraints.html>

1. Dynamic Domain Constraints

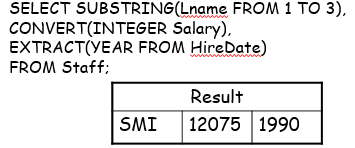
* Domains may be defined ‘dynamically’ using values that already exist in the database



* This could be used to ensure every StaffNo in PropertyForRent is valid

1. Scalar Functions

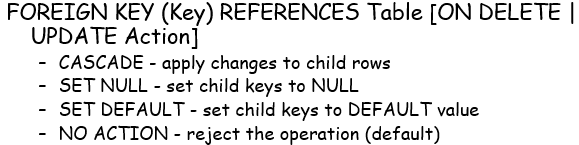
* Scalar functions may be used to convert/manipulate data values (something like MIN, MAX, etc.)
* ANSI SQL supports many scalar functions, here is one of them:



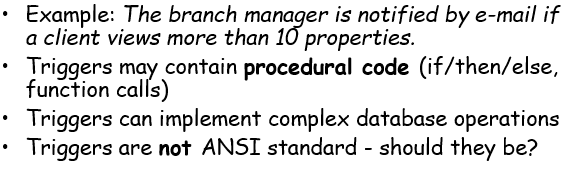
1. ON Delete | ON Update

<https://www.youtube.com/watch?v=UcbJNJdr_6A>

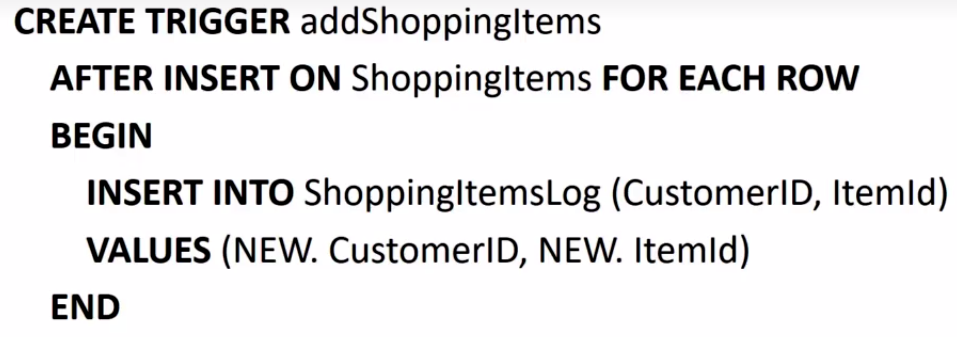
* Question: if a key attribute in the **parent** table is modified, what should happen in the child table ? - SQL provides 4 alternative referential actions:



1. Triggers



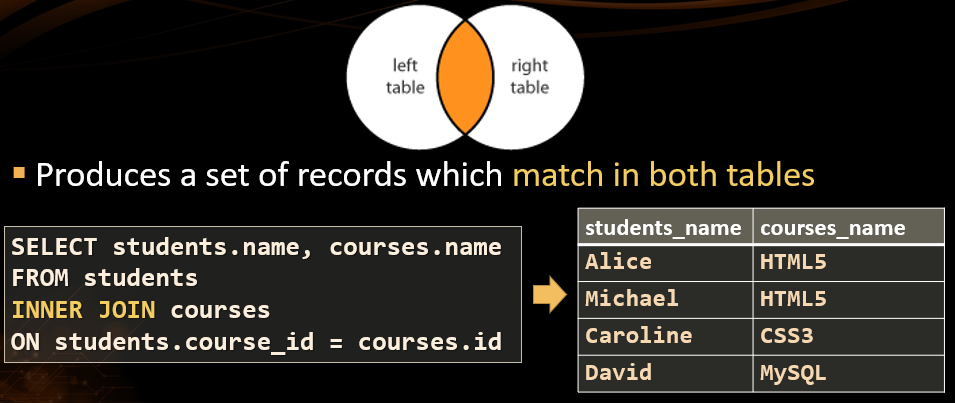
* Example:



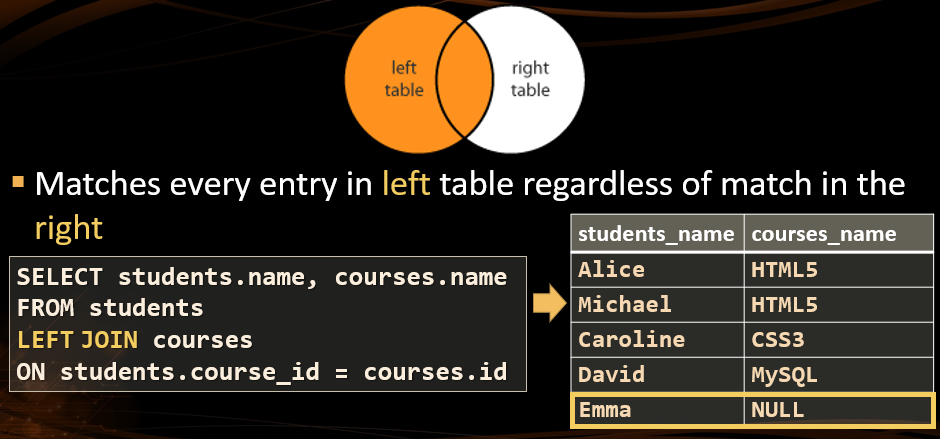
**SQL JOINS –** used to collect data from 2 or more tables



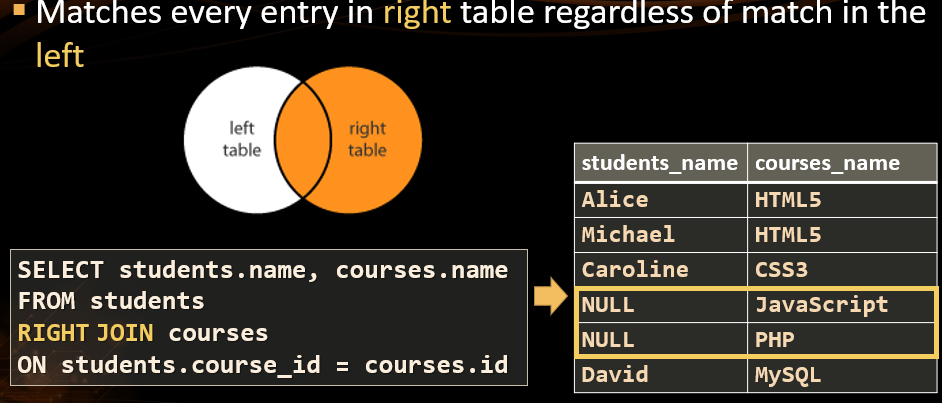
1. Inner Join

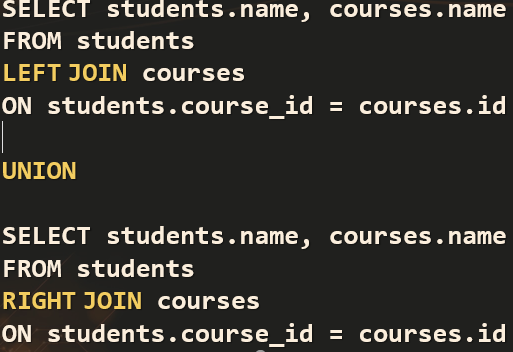


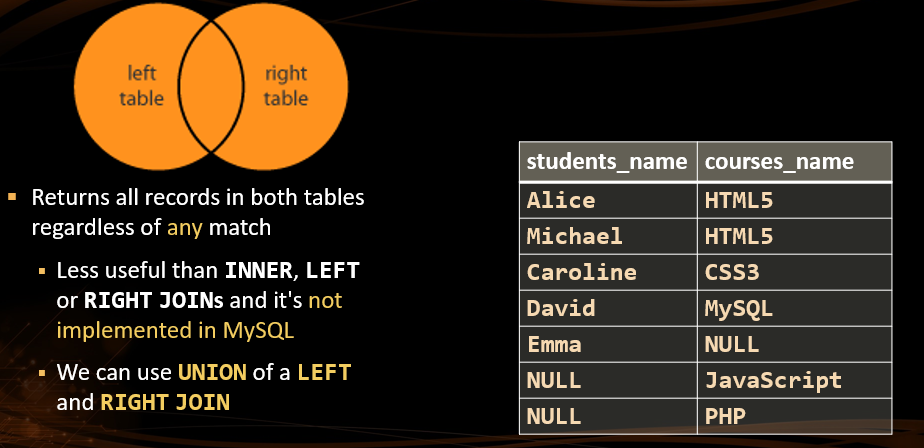
2. Left (Outer) Join



3. Right (Outer) Join

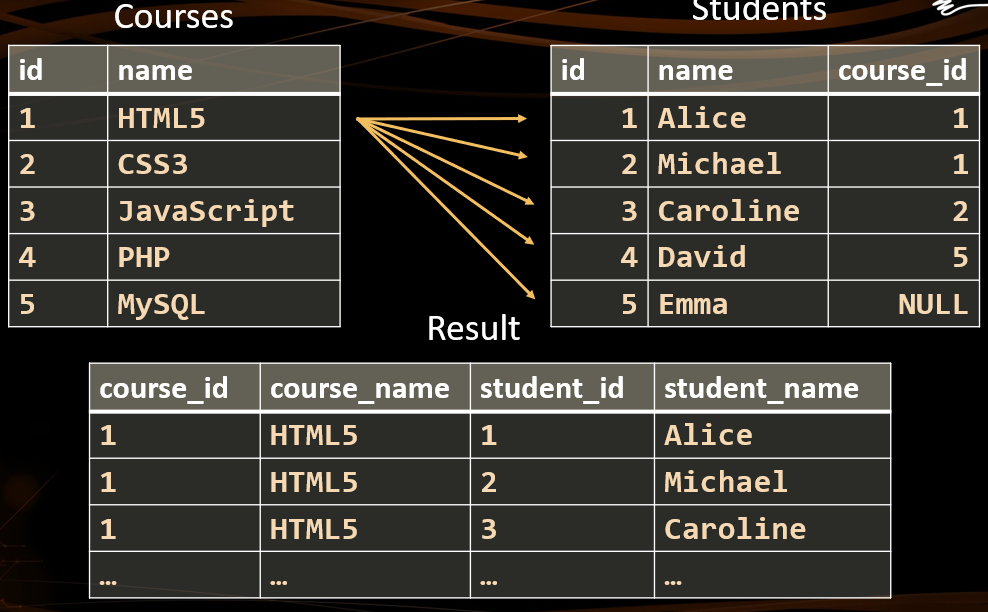
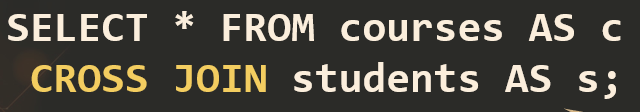


4. Full (Outer) Join + UNION



5. Cross Join / Cartesian Product

Each row in the first table is paired with all the rows in the second table

<https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/> Symmetric Difference/Complement included

<https://www.studytonight.com/dbms/joining-in-sql.php> Structure of the join syntax + Natural Join

<https://en.wikipedia.org/wiki/Join_(SQL)> Different names of the joins

*SELECT S.Fname, S.Lname, B.Street, B.City*

*FROM Staff AS S, Branch AS B*

*WHERE S.BranchNo = B.BranchNO*

*ORDER BY B.City, S.Lname;*

*/\**

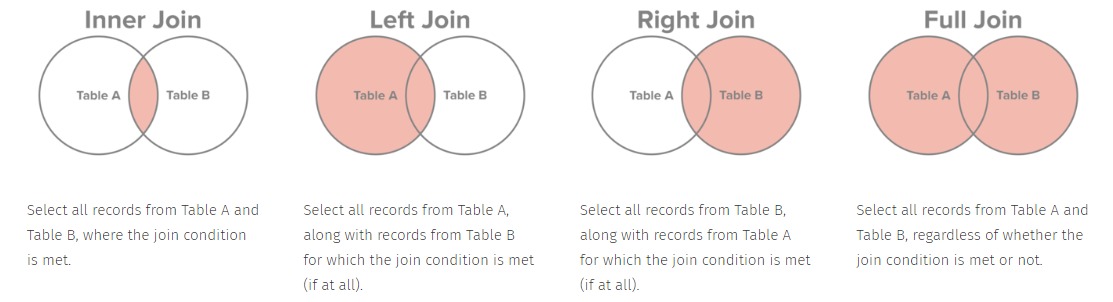
*WHICH IS THE SAME AS*

*SELECT Staff.Fname, Staff.Lname, Branch.Street, Branch.City*

*FROM Staff INNER JOIN Branch ON Staff.BranchNo = Branch.BranchNO*

*ORDER BY Branch.City, Staff.Lname;*

*\*/*



**SELECT \* (i.e. select everything) is not efficient!!! However, if you use it with a combination with a LIMIT the query will be more efficient.**

<https://stackoverflow.com/questions/28423687/is-it-necessary-to-use-limit-1-for-sql-select-for-performance>

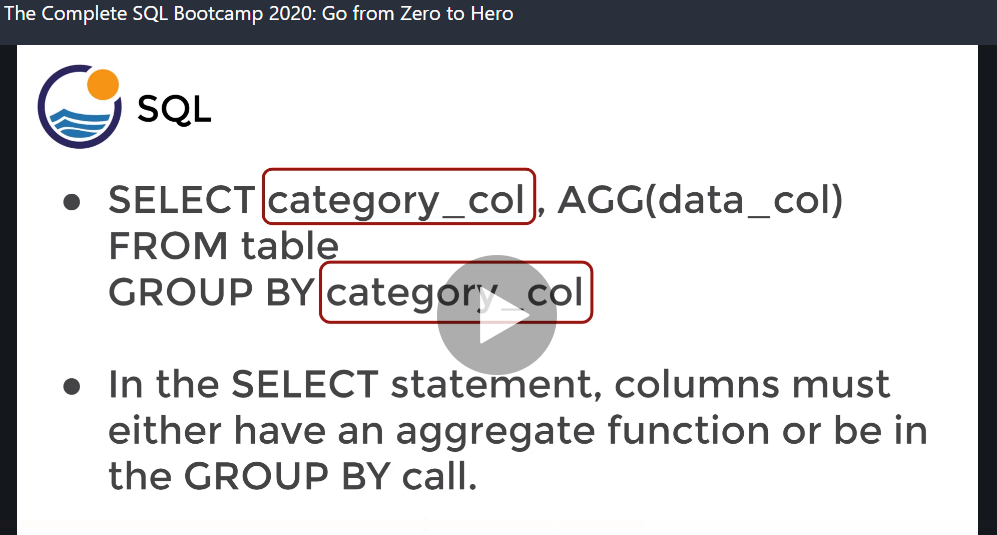
**GROUP BY with multiple columns**

<https://stackoverflow.com/questions/2421388/using-group-by-on-multiple-columns>

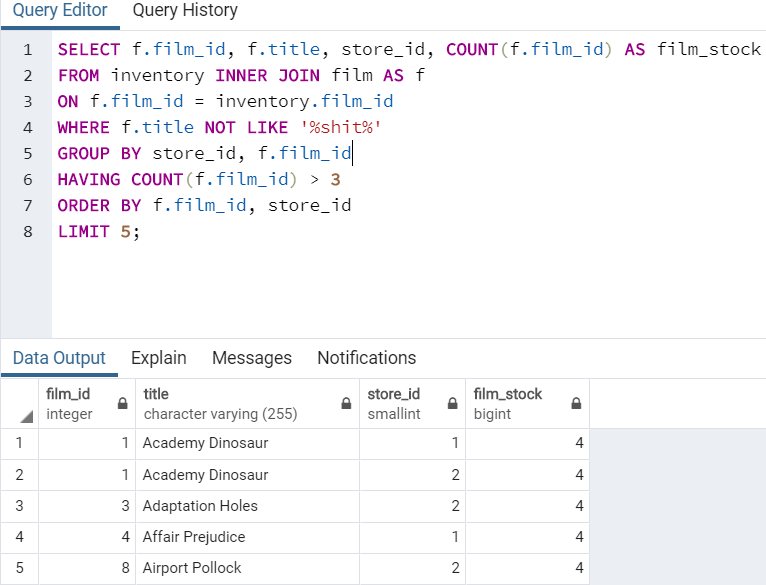
<https://stackoverflow.com/questions/28228133/sql-count-multiple-columns>

**GROUP BY Confusion**

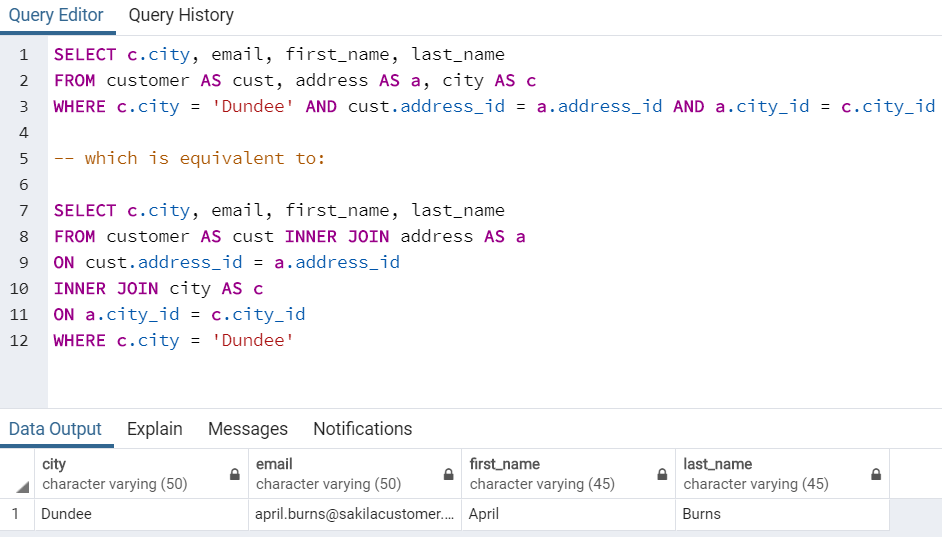
Functions allow only columns with matching values (note that aggregate functions return a single result)



**USEFUL COMPLEX QUERY:**



**Multiple Join Query (Email customers who live in Dundee)**



**Note: / is an integer division in postgresql. To get floor division cast:**

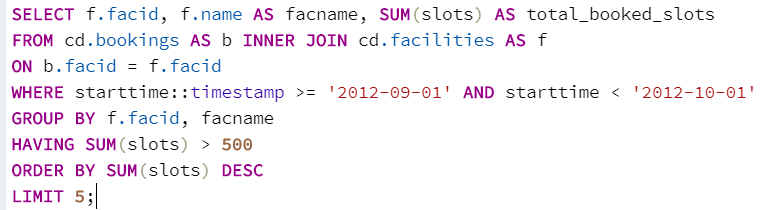
* MATH functions and operators: <https://www.postgresql.org/docs/9.5/functions-math.html>

*select dev\_cost::decimal / sell\_cost from software;*

*select cast(dev\_cost as decimal) / sell\_cost from software ;*

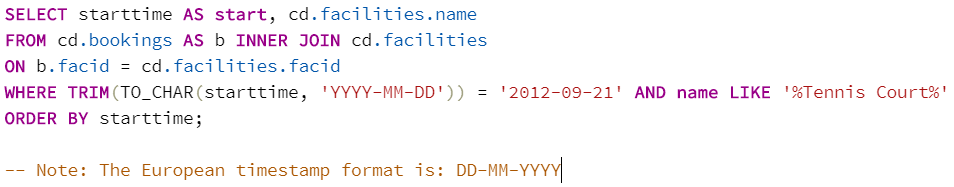
**USEFUL COMPLEX QUERY (GROUP BY with INNER JOIN):**

* <https://www.sqlshack.com/a-step-by-step-walkthrough-of-sql-inner-join/>
* Note: This task is taken from Assessment 2 – ex.11



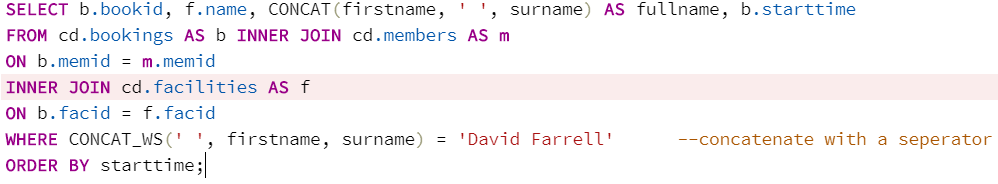
**Timestamps and data type formatting functions:**

* <https://www.postgresql.org/docs/12/functions-formatting.html>
* Note: This task is taken from Assessment 2 – ex.13

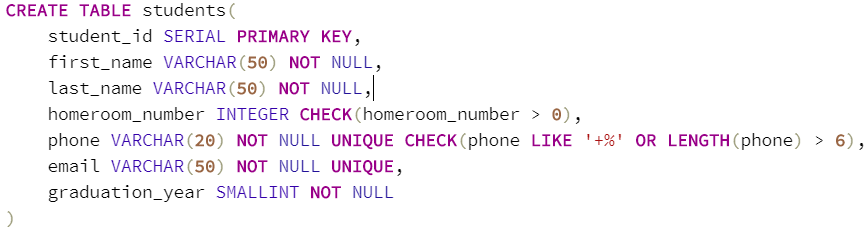


**USEFUL COMPLEX QUERY (String Functions and Multiple Joins):**

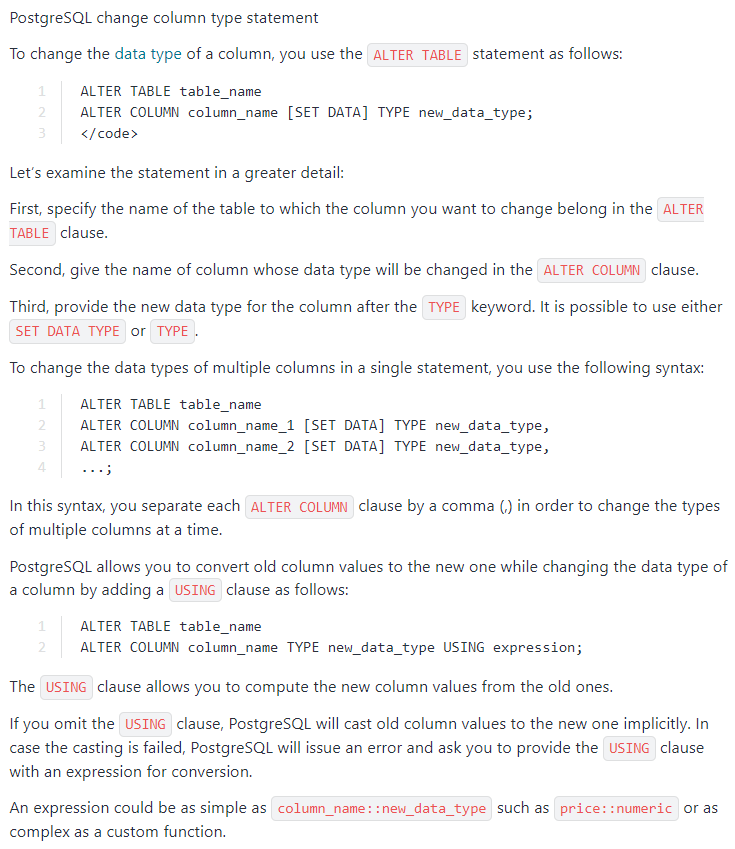
* <http://www.postgresql.cn/docs/9.5/functions-string.html> - CONCAT to concatenate names
* <https://learnsql.com/blog/illustrated-guide-multiple-join/>
* <https://www.sqlshack.com/sql-multiple-joins-for-beginners-with-examples/>
* Note: This task is taken from Assessment 2 – ex.14



**Creating Tables and Databases**

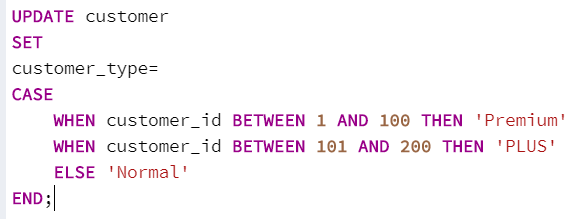


**How do you ALTER/change a column data type?**

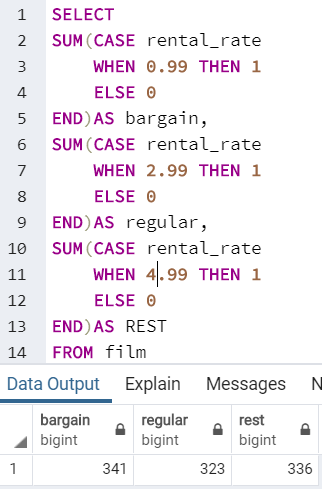


**Updating existing multiple rows in a table using the CASE keyword:**

<https://stackoverflow.com/questions/25674737/mysql-update-multiple-rows-with-different-values-in-one-query/25674827>



**Format groups in the output (GROUP BY alternative)**



**NOTE: for information on VIEWS go to the DB file from the Aberdeen course – Year2**