SQL

Ruonan Ji

August 13, 2021

Abstract

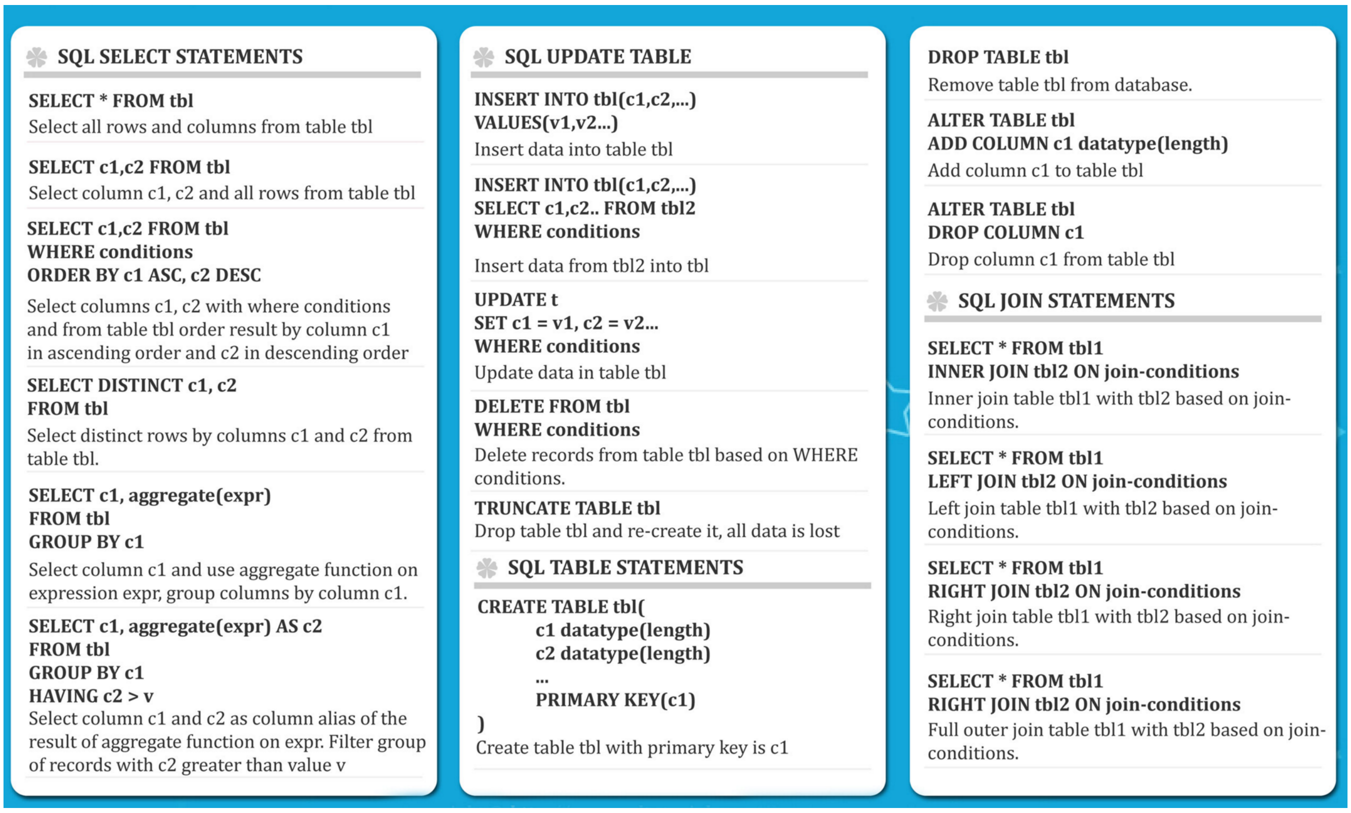
This is the notes that I take when study SQL, I am taking the course "The Complete SQL Bootcamp 2021: Go from Zero to Hero" from Udemy using pgAdmin 4.

1 Before Start

- Create database: right click Databases, Create, Database..., give a name to the new database, Save.
- Delete database: close query tool window, right click the corresponding database, Delete/Drop, Yes.
- Shortcut to run code in query tool: F5 to run all the code, select partial code and F5 to run specific chunk of code.
- Find tables: click the using database, click Schemas, under public there is Tables.

2 Statement Fundamentals

Cheat sheet for basic statements.



SQL SELECT STATEMENTS

```
SELECT * FROM tbl
```
Select all rows and columns from table tbl

```
SELECT c1,c2 FROM tbl
```
Select column c1, c2 and all rows from table tbl

```
SELECT c1,c2 FROM tbl
WHERE conditions
ORDER BY c1 ASC, c2 DESC
```
Select columns c1, c2 with where conditions and from table tbl order result by column c1 in ascending order and c2 in descending order

```
SELECT DISTINCT c1, c2
FROM tbl
```
Select distinct rows by columns c1 and c2 from table tbl.

```
SELECT c1, aggregate(expr)
FROM tbl
GROUP BY c1
```
Select column c1 and use aggregate function on expression expr, group columns by column c1.

```
SELECT c1, aggregate(expr) AS c2
FROM tbl
GROUP BY c1
HAVING c2 > v
```
Select column c1 and c2 as column alias of the result of aggregate function on expr. Filter group of records with c2 greater than value v

SQL UPDATE TABLE

```
INSERT INTO tbl(c1,c2,...)
VALUES(v1,v2...)
```
Insert data into table tbl

```
INSERT INTO tbl(c1,c2,...)
SELECT c1,c2.. FROM tbl2
WHERE conditions
```
Insert data from tbl2 into tbl

```
UPDATE t
SET c1 = v1, c2 = v2...
WHERE conditions
```
Update data in table tbl

```
DELETE FROM tbl
WHERE conditions
```
Delete records from table tbl based on WHERE conditions.

```
TRUNCATE TABLE tbl
```
Drop table tbl and re-create it, all data is lost

SQL TABLE STATEMENTS

```
CREATE TABLE tbl(
        c1 datatype(length)
        c2 datatype(length)
        ...
        PRIMARY KEY(c1)
)
```
Create table tbl with primary key is c1

```
DROP TABLE tbl
```
Remove table tbl from database.

```
ALTER TABLE tbl
ADD COLUMN c1 datatype(length)
```
Add column c1 to table tbl

```
ALTER TABLE tbl
DROP COLUMN c1
```
Drop column c1 from table tbl

SQL JOIN STATEMENTS

```
SELECT * FROM tbl1
INNER JOIN tbl2 ON join-conditions
```
Inner join table tbl1 with tbl2 based on join-conditions.

```
SELECT * FROM tbl1
LEFT JOIN tbl2 ON join-conditions
```
Left join table tbl1 with tbl2 based on join-conditions.

```
SELECT * FROM tbl1
RIGHT JOIN tbl2 ON join-conditions
```
Right join table tbl1 with tbl2 based on join-conditions.

```
SELECT * FROM tbl1
RIGHT JOIN tbl2 ON join-conditions
```
Full outer join table tbl1 with tbl2 based on join-conditions.

2.1 SELECT Statement

- Usage: select certain columns from a table.
- Format: SELECT c1, c2, c3 FROM table;

2.2 SELECT DISTINCT Statement

- Usage: only show distinct values in a certain column (no replication) from a table.
- Format: SELECT DISTINCT c1 FROM table;
- Extension: SELECT DISTINCT c1, c2 FROM table GROUP BY c1, c2;
- Extension note: extract multiple cols with distinct value, the length of distinct values for each col should be the same, otherwise, there would be some values replicated.)

2.3 COUNT Statement

- Usage: count the number of rows with the corresponding value from a table.
- Format: SELECT COUNT (c1) FROM table;
- Extension: SELECT COUNT (c1) as new name FROM table WHERE c1 = the selected value
- Extension note: count the number of rows / frequency when c1 equal a specific value (the selected value) from the table. (recommend to always add () after COUNT, not sure why but sometimes gives me syntax error when I don't, but the problem will be fixed if I add ().)

2.4 SELECT WHERE Statement

- Usage: select certain columns from a table.
- Format: SELECT c1, c2, c3 FROM table WHERE condition;
- Under condition: comparison operators(>=,!=...), logical operators(AND/OR/NOT)

2.5 ORDER BY Statement

- Usage: order the rows by columns in descending or ascending order.
- Format: SELECT c1,c2,c3 FROM table ORDER BY c1 DESC;
- Extension: SELECT c1,c2,c3 FROM table ORDER BY c1 DESC, c2 ASC;
- Extension note: order the rows first by c1 in descending order, then order by c2 in ascending order. (default order is ASC)

2.6 LIMIT Statement

- Usage: only extract certain number of rows from a table.
- Format: SELECT c1,c2 FROM table WHERE condition ORDER BY c1 LIMIT 5;
- Extension note: the order is very pivotal, I need to remember. I have tried to switch the order of the statements, but it shows syntax error.

2.7 BETWEEN AND Statement

- Usage: under WHERE statement, use between statement to set the range. It is same as <= value <= (inclusive). It can be used for date, time, number.
- Format: SELECT c1 FROM table WHERE c1 BETWEEN value1 AND value2;
- Extension: SELECT c1 FROM table WHERE c1 NOT BETWEEN value1 AND value2;
- Extension note: not between mean select the rows that have values smaller than value1 or greater than value2. Be careful about the date with time (2020-08-08 22:23:11 is not included when BETWEEN '2020-08-01' AND '2020-08-08' because the hour is over 00:00:00. So always double check the output date!)

2.8 IN Statement

- Usage: only extract rows that have values which match the selected values.
- Format: SELECT * FROM table WHERE c1 IN (value1,value2,value3);
- Extension: SELECT * FROM table WHERE c1 NOT IN (value1,value2,value3);
- Extension note: select the rows that have values which do not match the selected values.

2.9 LIKE / ILIKE Statement

- Usage: use the wildcard(%, _) to write some general patterns in a string to find the corresponding values.
- Wildcard: % matches any sequence of characters, _ matches any single character
- Statement: LIKE: case-sensitive, ILIKE: case-insensitive.
- Format: SELECT * FROM table WHERE c1 LIKE 'her_'; (herb)
- Format: SELECT * FROM table WHERE c1 LIKE '_her%'; (Whether)
- Format: SELECT * FROM table WHERE c1 NOT LIKE 'her_'; (blablabla)
- Extension note: select the rows that have values which do not match the pattern.

3 GROUP BY Statements

3.1 Aggregate Functions

- Usage: aggregate functions by using simple calculation.
- Common functions: AVG(), COUNT(), MAX(), MIN(), SUM().
- Format: SELECT ROUND(AVG(c1),2) FROM table;
- Explanation: round up the average of values in c1 with 2 decimal places.
- Note: aggregate function calls happen only in the SELECT clause or the HAVING clause. If I want to select other cols in SELECT clause, I should use GROUP BY.

3.2 GROUP BY Functions

- Usage: group rows that have same value, the column should be categorical.
- Format: SELECT category col, AGG(data col) FROM table WHERE category col != 'A' GROUP BY category col ORDER BY AGG(data col);
- Note 1: the GROUP BY clause must appear right after a FROM or WHERE statement.
- Note 2: columns in the SELECT statement must be mentioned in GROUP BY (aggregate functions are the exceptions, no need to be mentioned in group by).
- Note 3: WHERE statement should not mention aggregate functions.
- Note 4: if I want to ORDER BY aggregate functions, I have to reference the entire function like what I wrote in the format section.
- Note 5: the order of columns in GROUP BY does not matter, but the order of columns in SELECT matters.

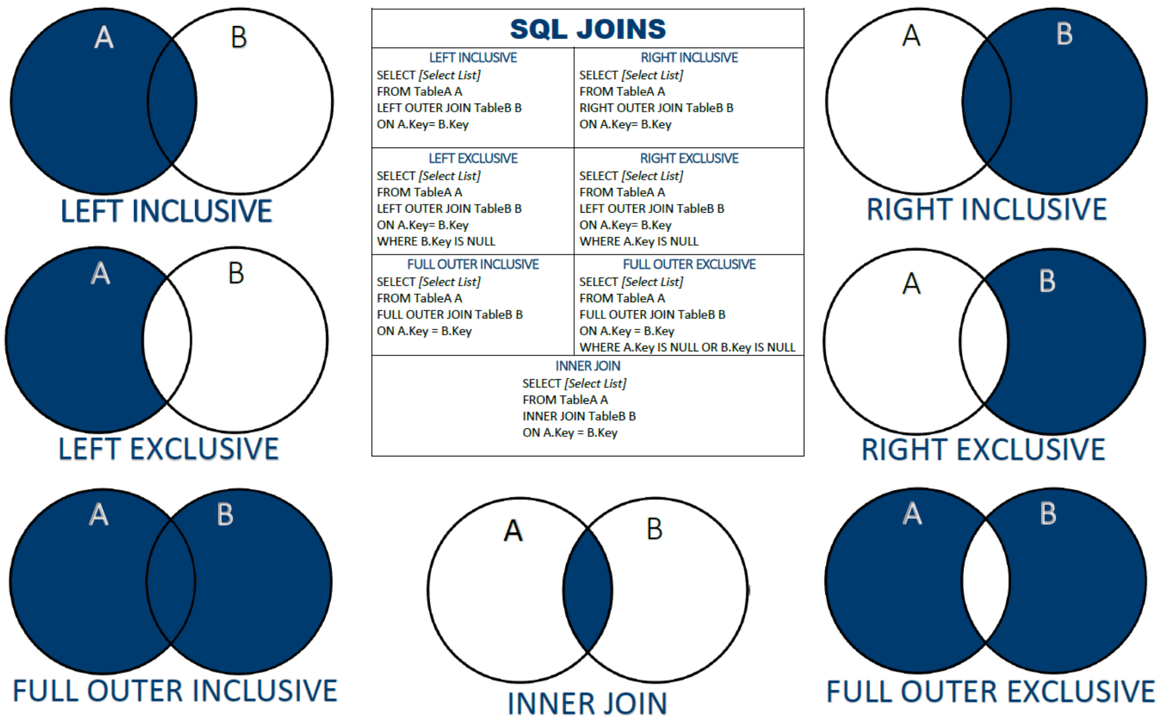
3.3 HAVING Functions

- Usage: filter aggregate functions.
- Format: SELECT c1, SUM(c2) FROM table WHERE c1 != 'value' GROUP BY c1 HAVING SUM(c2) > number;
- Note: I cannot use WHERE to filter aggregate functions. Instead, I should use HAVING right after GROUP BY to condition aggregate functions.

3.4 AS Functions

- Usage: output the variable name with a new name.
- Format: SELECT c1, SUM(c2) AS total amount FROM table WHERE c1 != 'value' GROUP BY c1 HAVING SUM(c2) > number;
- Note: the new name only functions as an output name, it cannot be called as the original variable in any statement because it gets executed at the very end of a query. I have to call SUM(c2) in HAVING statement (Or c1 in WHERE statement).

4 JOINS



SQL JOINS

LEFT INCLUSIVE
```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
```

RIGHT INCLUSIVE
```
SELECT [Select List]
FROM TableA A
RIGHT OUTER JOIN TableB B
ON A.Key= B.Key
```

LEFT EXCLUSIVE
```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
WHERE B.Key IS NULL
```

RIGHT EXCLUSIVE
```
SELECT [Select List]
FROM TableA A
LEFT OUTER JOIN TableB B
ON A.Key= B.Key
WHERE A.Key IS NULL
```

FULL OUTER INCLUSIVE
```
SELECT [Select List]
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```

FULL OUTER EXCLUSIVE
```
SELECT [Select List]
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL
```

INNER JOIN
```
SELECT [Select List]
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```

4.1 INNER JOINS

- Usage: get a new table that contains values which match in both tables.
- Format: SELECT c1,tableA.c2,c3 FROM tableA INNER JOIN tableB ON tableA.c2 = tableB.c2
- Note: SELECT c1,tableA.c2,c3 can eliminate duplicaion. Of course, I can just use SELECT * if I do not care about duplication.

4.2 FULL OUTER JOINS

- Usage: get a new table that contains all values from both tables.
- Format: SELECT * FROM tableA FULL OUTER JOIN tableB ON tableA.c2 = tableB.c2
- Note: this is is for full outer inclusive graph. One important thing is if the values do not match, it will appear nulls!!!
- Extension: full outer exclusive graph excludes the common values from two tables, it always has nulls.
- Format: SELECT * FROM tableA FULL OUTER JOIN tableB ON tableA.c2 = tableB.c2 WHERE tableA.c2 IS null OR tableB.c2 IS null

4.3 LEFT OUTER JOINS

- Usage: get a new table that contains all values from one table.
- Format: SELECT * FROM tableA LEFT OUTER JOIN tableB ON tableA.c2 = tableB.c2
- Note: this is is for left inclusive graph. One important thing is if the values in tableB do not match values in tableA, the new table will appear nulls in tableB.
- Extension: left exclusive graph excludes the common values from two tables, it always has nulls.

- Format: SELECT * FROM tableA LEFT OUTER JOIN tableB ON tableA.c2 = tableB.c2 WHERE tableB.c2 IS null

4.4 RIGHT OUTER JOINS

- Usage: get a new table that contains all values from one table.
- Format: SELECT * FROM tableA RIGHT OUTER JOIN tableB ON tableA.c2 = tableB.c2
- Note: this is is for right inclusive graph. One important thing is if the values in tableA do not match values in tableB, the new table will appear nulls in tableA.
- Extension: right exclusive graph excludes the common values from two tables, it always has nulls.
- Format: SELECT * FROM tableA RIGHT OUTER JOIN tableB ON tableA.c2 = tableB.c2 WHERE tableA.c2 IS null

4.5 UNION

- Usage: combine two or more SELECT statements.
- Format: SELECT c1 FROM tableA UNION SELECT c1 FROM tableB
- Note: the cols from tableA and tableB should match.

5 Advanced SQL Commands

5.1 TIMESTAMPS and EXTRACT

- Usage: different forms of time.
- Format: SHOW TIMEZONE / SELECT NOW() / SELECT TIMEOFDAY() / SELECT CURRENT_TIME / SELECT CURRENT_DAY
- Extension: date and time order: TIME, DATE, TIMESTAMP, TIMESTAMPTZ.
- Note: be careful when choosing level of TIMESTAMPTZ, I can always remove it but cannot add it.
- functions 1: EXTRACT() extracts subcomponent from a col.
- Format 1: SELECT EXTRACT (YEAR/MONTH/DAY/WEEK/QUARTER FROM c1) FROM table
- functions 2: AGE() automatically calculates the current age given a timestamp.
- Format 2: SELECT AGE (c1) FROM table
- functions 3: TO_CHAR converts data types to certain form of text.
- Format 3: SELECT TO_CHAR (c1, 'MONTH-YYYY') FROM table
- link for data type formatting: https://www.postgresql.org/docs/12/functions-formatting.html

5.2 Mathematical Functions and Operations

- Documentation for various signs: https://www.postgresql.org/docs/9.5/functions-math.html
- Format: SELECT c1 % c2 AS deposit FROM table

5.3 String Functions and Operations

- Documentation for various signs: https://www.postgresql.org/docs/9.1/functions-string.html
- Format 1: SELECT first_name || ‘ ’ || last_name AS fullname FROM table
- Note 1: to concatenate first name and last name into one string with space between them.
- Format 2: SELECT LOWER(LEFT(first_name,1)) || LOWER(last_name) || ‘@gmail.com’ AS custom_email FROM table
- Note 2: to build a email address by picking the first letter of first name and concatenate with last name, lower the letters and adding @gmail.com at the end of the string.

5.4 SubQuery

- Usage: performing a query on the results of another query (two SELECT statements)
- Format 1: SELECT c1 FROM table.a WHERE c1 IN (SELECT c1 FROM table.b)
- Note 1: the subquery is inside the parenthesis and will be runed first. I can also use comparison operators and other commands instead of IN.
- Extension 1: I can use JOINS in the subquery.
- Format 2: SELECT c1,c2 FROM tablea AS a WHERE EXISTS (SELECT * FROM tableb AS b WHERE b.c3 = a.c3)

6 Creating Databases and Tables

6.1 Data Types

- Documentation for data types: https://www.postgresql.org/docs/current/datatype.html
- Data types: Boolean (T/F), Character (char/varchar/text), Numeric (integer, floating-point number), Temporal(date, time, timestamp, interval), UUID, Array, JSON, Hstore key-value pair, special types such as network address and geometric data.
- Note: Take time to plan for long term storage!

6.2 Primary Keys

- Define: a primary key is a column or a group of columns used to identify a row uniquely in a table.
- Usage: primary keys are important since they allow us to easily discern what columns should be used for joining tables together.
- Note: when the table it shown, under a column name, if it says [PK], then the column is primary key. Moreover, the primary key contains values that are unique and non-null! (example: ID numbers, each person will have different ID to identify themselves from others.)

6.3 Foreign Keys

- Define: a foreign key is a field or group of fields in a table that uniquely identifies a row in another table.
- Note: the table that contains the foreign key is called referencing table or child table. The table to which the foreign key references is called referenced table or parent table.
- Note: I can see which columns are primary keys and foreign keys by clicking "constraints" below "schemas"

6.4 Constraints

- Define: constraints are the rules enforced on data columns on table.
- Usage: it improves the accuracy and reliability of the data in the database by preventing invalid data from being entered into the database.
- Usage: column constraints: constrains the data in a columns to adhere to certain conditions; table constraints: applied to the entire table rather than to an individual column
- Note: common constraints: NOT NULL constraint, UNIQUE constraint, PRIMARY Key, FOREIGN Key, CHECK constraint, EXCLUSION constraint, REFERENCES,

6.5 CREATE Tables

- Format: CREATE TABLE tbl_name(user_id SERIAL PRIMARY KEY, username VARCHAR(50) UNIQUE NOT NULL, created_on TIMESTAMP NOT NULL)
- Note: tbl_name is the created table's name, user_id, username and created_on are the three columns, SERIAL, VARCHAR and TIMESTAMP are the data type from the column, PRIMARY KEY, UNIQUE and NOT NULL are the column constraints.
- Extension: CREATE TABLE account_job(user_id INTEGER REFERENCES account(user_id), job_id INTEGER REFERENCES job(job_id), hire_date TIMESTAMP)
- Note: account_job is a reference table, user_id and job_id are foreign keys because they are originally from account table and job table.

6.6 INSERT

- Define: INSERT helps me add in rows to a table
- Note: the inserted row values must match up for the table, including constraints.
- Format: INSERT INTO account(username, created_on) VALUES ('Jenna', CURRENT_TIMESTAMP)
- Note: account is the table name. username and created_on are column names from the table.

6.7 UPDATE

- Define: UPDATE helps me change values of the columns in a table
- Format 1: UPDATE account SET c1 = CURRENT_TIMESTAMP WHERE c2 = 5
- Note 1: account is the table name. c1 is the column name where values will be updated if meets the WHERE condition.
- Format 2: UPDATE account SET original_col = tableB.new_col FROM tableB WHERE account.id = tableB.id
- Note 2: use another table's values by applying UPDATE join method.
- Format 3: UPDATE account SET last_login = created_on returning account_id, last_login
- Note 3: only output affected rows

6.8 DELETE

- Define: delete rows from a table
- Format: DELETE FROM account WHERE row_id = 1
- Note: similar to UPDATE command, I can also add in a RETURNING call to output rows that have been deleted.

6.9 ALTER

- Define: adding, dropping or renaming columns; changing a column's data type; set DEFAULT values for a column; add CHECK constraints; rename table
- Format: ALTER TABLE account ADD COLUMN new_col TYPE
- Note: the above format is just a simple example of adding a column. There are so many different ALTER commands. This website can give some basic ideas of what I can do with ALTER: https://www.java67.com/2013/01/how-to-use-alter-command-in-sql-examples.html.

6.10 DROP

- Define: allows for the complete removal of a column in a table.
- Format 1: ALTER TABLE account DROP COLUMN c1
- Format 2: ALTER TABLE account DROP COLUMN c1 CASCADE
- Note 2: remove all dependencies: CASCADE clause is needed because DROP will not remove cols used in views, triggers or stored procedures.
- Format 3: ALTER TABLE account DROP COLUMN IF EXISTS c1
- Note 3: I consider this format as a useful, important tool. It is always safe to check for existence to avoid error.
- Format 4: ALTER TABLE account DROP COLUMN c1, DROP COLUMN c2, DROP COLUMN c3
- Note 4: drop multiple cols.

6.11 CHECK

- Define: the CHECK constraints are added when creating a table. The constraints allow me to create more customized constraints that adhere to a certain condition.
- Format: CREATE TABLE account (id SERIAL PRIMARY KEY, birthdate DATE CHECK (birthdate > '1900-01-01'), hiredate DATE CHECK (hiredate > birthdate))
- Note: when INSERT values into the created table, if the values violate the CHECK constraints, it will produce an error that warns me the values are not appropriate.

7 Conditional Expressions and Procedures

7.1 Case

- Define: CASE statement works like if...else... statement.
- Note: there are two main ways: a general CASE; a CASE expression.
- Format 1:
  ```
  SELECT c1,
  CASE
  WHEN (c1 <= 100) THEN 'a'
  WHEN (c1 BETWEEN 1 AND 10 THEN 'b')
  ELSE 'c'
  END
  FROM tbl
  ```
- Note 1: (I failed to find a way to indent. It should be(select...from(case...end)(when...when...else...)))
  The above format is an example of a general CASE which for WHEN function can check conditions.

- Format 2:
  ```
  SELECT c1,
  CASE c1
  WHEN 100 THEN ‘a’
  WHEN 10 THEN ‘b’
  ELSE ‘c’
  END
  FROM tbl
  ```
- Note 2: The above format is an example of a CASE expression which for WHEN function can check values’ equality.

7.2 COALESCE

- Define: the COALESCE function accepts an unlimited number of arguments and returns the first argument that is not null.
- Format: SELECT c1, (c3-COALESCE(c2,0)) AS new_col FROM tbl
- Note: null values from c2 will be replaced with 0s. This function is very useful when substitute null values with another value.

7.3 CAST

- Define: the CAST function allows to convert from one data type into another, but the converting should be reasonable. (‘5’ to 5)
- Format 1: SELECT CAST (‘5’AS INTEGER)
  Another way of doing it: SELECT ‘5’:: INTEGER
- Note 1: two ways of converting a string into an integer.
- Format 2: SELECT CAST (c1AS INTEGER) FROM tbl

7.4 NULLIF

- Define: it takes in 2 arguments and return NULL if the arguments are equal to each other; return the first argument if the arguments are not equal.
- Format: SELECT c1/ NULLIF (c2,0) FROM tbl
- Note: use NULLIF will prevent producing error when the denominator is 0. If c2 is 0, it will output ‘NULL’, if c2 is not 0, it will output a normal division result.

7.5 Views

- Define: a view acts more like defining a function in other coding languages. More specifically, stores a bunch of queries that I use frequently and create a temporary table that helps me extract certain data quickly.
- Format:
  ```
  CREATE VIEW tbl AS
  SELECT c1,c2,c3 FROM tblAA
  INNER JOIN tblBB
  ON tblAA.c1 = tblBB.c1
  ```
- Note: There are many ways to edit a view like changing its name, adding columns, deleting it. Explore them using google.

7.6 Import and Export

- Note: Explore them using google.