Udemy Course

The Complete SQL Boot camp: From zero to hero

Database: Systems that allow users to store and organize data

Typical User: Analysts, Technical, Data scientist, etc

Advantages of Data base

1. Data Integrity
2. Can handle massive amounts of data
3. Quickly combine different datasets
4. Automate steps for re-use
5. Can support data for websites and applications

PostGreSQL

SQL (Structured Query Language): Programming language used to communicate with our Database

Pgadmin4

SELECT statement: it allows us to retrieve information from a table

SELECT column\_name FROM table\_name

Table 1

Table 3

Table 2

Database

Not good practice to use an asterisk(\*) in the select statement. It will automatically query everything, which increases traffic between the application, which can show down the retrieval of results

SELECT first\_name,last\_name FROM actor;

Challenge Structure:

1. Business Situation
2. Challenge Question
3. Expected Answer
4. Hints
5. Solution

Situation: We want to send out a promotional email to our existing customers

Challenge: Use SELECT statement to grab the first and last names of every customer and their email address

Expected Answer:

|  |  |  |  |
| --- | --- | --- | --- |
|  | First\_name | Last\_name | email |
| 1 | Jared | Ely | Jared.ely@gmail.com |
| 2 |  |  |  |

Hints:

1. Use the customer table
2. You can use the table dorp-down to view what columns are available
3. You could also use “SELECT \* FROM customer” to see all the columns

Solution: SELECT first\_name,last\_name,email FROM customer;

DISTINCT keyword: returns unique elements of column

SELECT DISTINCT column FROM table

SELECT DISTINCT(release\_year) FROM film;

SELECT DISTINCT(rental\_rate) FROM film;

Situation:

1. An Australian visitor is not familiar with MPAA movie ratings (e.g. PG,PG-13,R,etc..)
2. We want to know the types of ratings, we have in our database
3. What ratings do we have available?

SQL Challenge: use what you have learned about SELECT DISTINCT to retrieve the distinct rating types our films could have in our database.

Expected Result:

|  |  |
| --- | --- |
|  | rating |
| 1 | NC-17 |
| 2 | G |
| 3 | PG |

Hints:

1. Use film table
2. Use SELECT \* FROM film; to see what columns are available
3. Or use drop down table menu in pgadmin.

Solution:

SELECT DISTINCT rating FROM film;

COUNT: returns the number of input rows that match a specific condition of a query

We can apply COUNT on a specific column or just pass COUNT(\*)

SELECT COUNT(name) FROM table;

How many unique names in the table

SELECT COUNT(DISTINCT name) FROM table;

SELECT COUNT(\*) FROM payment;

SELECT COUNT(amount) FROM payment;

SELECT COUNT(DISTINCT amount) FROM payment;

SELECT WHERE : WHERE statement allows us to specify conditions on columns for the rows to be returned

SELECT col1,col2 FROM table1 WHERE conditions;

Comparison Operators: Compare a column value to something (e.g. is the price greater than $3.00?)

|  |  |
| --- | --- |
| Operator | Description |
| = | Equal |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| <> or != | Not equal to |

Logical Operators: AND, OR, NOT

SELECT name,choice FROM table WHERE name=’David’

SELECT name,choice FROM table WHERE name=’David’ AND choice=’Red’

SELECT \* FROM customer WHERE first\_name = 'Jared';

SELECT \* FROM film WHERE rental\_rate > 4.00;

SELECT COUNT(title) FROM film WHERE rental\_rate > 4.00 AND replacement\_cost >= 19.99 AND rating='R';

SELECT COUNT(\*) FROM film WHERE rating = 'R' OR rating= 'PG-13';

Challenge 1 : A customer forgot their wallet at our store! We need to track down their email to inform them. What is the email for the customer with the name Nancy Thomas?

Solution: SELECT email FROM customer WHERE first\_name = ‘Nancy’ AND last\_name = ‘Thomas’

Challenge 2: A customer wants to know what the moive ‘Outlaw Hanky’ is about.

Solution: SELECT description FROM film WHERE tiltle = ‘Outlaw Hanky’;

ORDER BY: To sort rows based on a column value, in either ascending or descending order

SELECT col1,col2 FROM table ORDER BY col1,col2 ASC/DESC

SELECT \* FROM customer ORDER BY first\_name ASC;

SELECT store\_id, first\_name, last\_name FROM customer ORDER BY store\_id DESC, first\_name ASC;

LIMIT: allows us to limit the number of rows returned for a query

Limits also becomes useful in combination with ORDER BY

select \* from payment ORDER BY payment\_date DESC LIMIT 5;

select \* from payment WHERE amount > 0.00 ORDER BY payment\_date DESC LIMIT 5;

Challenge: we want to reward our first 10 paying customers. What are the customer ids of the first 10 customers who created a payment?

Solution: SELECT customer\_id FROM payment ORDER BY payment\_date ASC LIMIT 10;

Challenge: A customer wants to quickly rent a video to watch over their short lunch break. What are the titles of the 5 shortest (in length of runtime) moives?

Solution: SELECT title, length FROM film ORDER BY length ASC LIMIT 5;

BETWEEN: can be used to match a value against a range of values

NOT BETWEEN

* Value BETWEEN low AND high
* Date BETWEEN ‘2007-01-01’ AND ‘2007-02-01’

IN :

SELECT color FROM table WHERE color IN (‘red’, ‘blue’)

SELECT color FROM table WHERE color NOT IN (‘red’, ‘blue’)

SELECT \* FROM payment WHERE amount IN (0.99,1.98,1.99);

SELECT \* FROM customer WHERE first\_name IN ('John','Jake','Julie');

LIKE & ILIKE: Pattern matching using string data. If we want to match against a general pattern in a string. Example:

* All emails ending in ‘@gmail.com’
* All names that begin with an ‘A’

The LIKE operator allows us to perform pattern matching against string data with the use of wildcard characters:

* Percent %: Matches any sequence of characters
  + All names that begin with an ‘A’ = WHERE name LIKE ‘A%’
  + All names that end with an ‘a’ = WHERE name LIKE ’%a’

# NOTE: LIKE is case-sensitive, we can use ILIKE which is case-insensitive

* Underscore \_: Matches any single character
  + Using underscore allows us to replace just a single character
  + Get all Missing Impossible films WHERE title LIKE ‘Mission Impossible \_’
  + We can use multiple underscore
* We can combine pattern matching operator to create more complex patterns

Example: WHERE name LIKE ‘\_her%’

#PostgreSQL does support full regex capabilities

SELECT \* FROM customer WHERE first\_name LIKE 'J%' AND last\_name LIKE 'S%';

SELECT \* FROM customer WHERE first\_name LIKE 'A%' AND last\_name NOT LIKE 'B%' ORDER BY last\_name;

Challenge: How many payment transactions were greater than $5.00?

Solution: SELECT COUNT(amount) FROM payment WHERE amount > 5.00;

Challenge: How many actors have a first name that starts with the letter P?

Solution: SELECT COUNT(\*) FROM actor WHERE first\_name LIKE 'P%';

Challenge: How many unique districts are our customers from?

Solution: SELECT COUNT(DISTINCT district) FROM address;

Challenge: Retrieve the list of name for those distinct districts from the previous question

Solution: SELECT DISTINCT district FROM address;

Challenge: How many films have a rating of R and a replacement cost between $5 and $15 ?

Solution: SELECT COUNT(\*) FROM film WHERE rating = 'R' AND (replacement\_cost BETWEEN 5 AND 15);

Challenge: How many films have the word Truman somewhere in the title?

Solution: SELECT COUNT(\*) FROM film WHERE title LIKE '%Truman%';

GROUP BY: Allow us to aggregate data and apply functions to better understand how data is distributed per category

The Main idea behind an aggregate function is to multiple inputs and return a single output.

Most common Aggregate Funtions:

* AVG() – returns average value
* COUNT() – returns number of values
* MAX() – returns maximum value
* MIN() – returns minimum value
* SUM() – returns the sum of all values

# Aggregate function calls happen only in the SELECT clause or the HAVING clause.

# AVG() returns a floating point value many decimal places (e.g. 2.343354), we can use ROUND() to specify precision after the decimal.

SELECT MIN(replacement\_cost) FROM film;

SELECT MAX(replacement\_cost) FROM film;

SELECT MAX(replacement\_cost), MIN(replacement\_cost) FROM film;

SELECT ROUND(AVG(replacement\_cost),2) FROM film;

SELECT SUM(replacement\_cost) FROM film;

We need to choose a categorical column to GROUP BY

Categorical columns are non-continuous.

SELECT category\_col, AGG(data\_col) FROM table GROUP BY category\_col

#The GROUP BY clause must appear right after a FROM or WHERE statement.

In the SELECT statement, columns must either have an aggregate function or be in the GROUP BY call

SELECT company, division, SUM(sales) FROM finance\_table GROUP BY company, division;

# In the SELECT statement, columns must either have an aggregate function or be in the GROUP BY call

SELECT company, division, SUM(sales) FROM finance\_table WHERE division IN (‘marketing’,’transport’) GROUP BY company ,division;

# WHERE statements should not refer to the aggregation result, later on we will learn to use HAVING to filter on those results.

SELECT company, SUM(sales) FROM finance\_table GROUP BY company ORDER BY SUM(sales);

If you want to sort results based on the aggregate, make sure to reference the entire function

SELECT customer\_id,SUM(amount) FROM payment GROUP BY customer\_id ORDER BY SUM(amount) DESC

SELECT staff\_id,customer\_id,SUM(amount) FROM payment GROUP BY staff\_id, customer\_id ORDER BY customer\_id

SELECT DATE(payment\_date),SUM(amount) FROM payment GROUP BY DATE(payment\_date) # use DATE function to remove timestamp and only show date value

Challenge: we have two staff members, with staff IDs 1 and 2. We want to give a bonus to the staff member that handled the most payments. (Most in terms of number of payments processed, not total dollar amount). How many payments did each staff member handle and who gets the bonus?

Solution: SELECT staff\_id, COUNT(payment\_id) FROM payment GROUP BY staff\_id ORDER BY COUNT(payment\_id) DESC

Challenge: Corporate HQ is conducting a study on the relationship between replacement cost and a moive MPAA rating (e.g. G,PG,R.etc). What is the average replacement cost per MPAA rating?

Solution: SELECT rating,AVG(replacement\_cost) FROM film GROUP BY rating

Challenge: We are running a promotion to reward our top 5 customers with coupons. What are the customer ids of the top 5 customers by total spend?

Solution: SELECT customer\_id, SUM(amount) FROM payment GROUP BY customer\_id ORDER BY SUM(amount) DESC LIMIT 5

HAVING: Allows us to filter after an aggregation has already taken place.

SELECT company, SUM(sales) FROM finance\_table WHERE company != ‘Google’ GROUP BY company HAVING SUM(sales)>1000

# HAVING allows us to use the aggregate result as a filter along with a GROUP BY

SELECT customer\_id, SUM(amount) FROM payment GROUP BY customer\_id HAVING SUM(amount)>100;

SELECT store\_id, COUNT(\*) FROM customer GROUP BY store\_id HAVING COUNT(\*) > 300;

Challenge: What customer\_ids are eligible for platinum status (40 or more transaction payment)

Solution: SELECT customer\_id,count(\*) FROM payment GROUP BY customer\_id HAVING count(\*) > 39;

Challenge: What are the customer ids of customers who have spent more than $100 in payment transactions with our staff\_id member 2

Solution: SELECT customer\_id, SUM(amount) FROM payment WHERE staff\_id = 2 GROUP BY customer\_id HAVING SUM(amount)>100;

**ASSESSMENT TEST 1**

**COMPLETE THE FOLLOWING TASKS!**

1. Return the customer IDs of customers who have spent at least $110 with the staff member who has an ID of 2.

The answer should be customers 187 and 148.

Solution: SELECT customer\_id,SUM(amount) FROM payment WHERE staff\_id = 2 GROUP BY customer\_id HAVING SUM(amount)>=110

2. How many films begin with the letter J?

The answer should be 20.

Solution: SELECT COUNT(\*) FROM film WHERE title LIKE 'J%'

3. What customer has the highest customer ID number whose name starts **with** an 'E' **and**has an address ID lower than 500?

The answer is Eddie Tomlin

Solution: SELECT first\_name,last\_name FROM customer WHERE first\_name LIKE 'E%' AND address\_id < 500 ORDER BY customer\_id DESC LIMIT 1;

JOIN

AS: which allows us to create an ‘alias’ for a column result

SELECT column AS new\_name FROM table

Alias cannot be use in WHERE and HAVING clause

INNER JOIN: will result with the set of records that match in both tables

SELECT \* FROM TableA INNER JOIN TableB ON TableA.col\_match = TableB.col\_match

SELECT payment\_id,payment.customer\_id, first\_name FROM payment INNER JOIN customer ON payment.customer\_id = customer.customer\_id

FULL OUTER JOIN: select everything

SELECT \* FROM Registration FULL OUTER JOIN Logins ON Registrations.name = Logins.name

FULL OUTER JOIN with WHERE: Get rows unique to either table(rows not found in both tables)

SELECT \* FROM TableA FULL OUTER JOIN TableB ON TableAcol\_match = TableB.col\_match WHERE TableA.id IS null OR TableB.id IS null

LEFT OUTER JOIN: results in the set of records that are in the left table, if there is no match with the right table, the results are null

SELECT \* FROM TableA LEFT OUTER JOIN TableB ON TableA.col\_match = TableB.col\_match

SELECT film.film\_id, title, inventory\_id FROM film LEFT JOIN inventory ON inventory.film\_id = film.film\_id

LEFT OUTER JOIN with WHERE:

SELECT \* FROM TableA LEFT OUTER JOIN TableB ON TableA.col\_match = Table.col\_match WHERE TableB.id IS null

RIGHT OUTER JOIN: SELECT \* FROM TableA RIGHT OUTER JOIN TableB ON TableA.col\_match = Table B.col\_match

RIGHT OUTER JOIN with WHERE: SELECT \* FROM TableA RIGHT OUTER JOIN TableB ON TableA.col\_match = Table B.col\_match WHERE TableA.id IS null

UNION: operator is used to combine the result-set of two or more SELECT statement

It basically serves to directly concatenate two results together, essentially “pasting” them together.

SELECT column\_name(s) FROM table1 UNION SELECT column\_name(s) FROM table2;

SELECT \* FROM Sales2021\_Q1 UNION SELECT \* FROM Sales2021\_Q2

JOIN Challenges

Challenges01: What are the emails of the customers who live in California

* SELECT email FROM customer

JOIN address ON customer.address\_id = address.address\_id

WHERE district = 'California';

Challenge02: Get list of all movies “Nick Wahlberg” has been in.

* SELECT title,actor.first\_name,actor.last\_name FROM film
* JOIN film\_actor ON film.film\_id = film\_actor.film\_id
* JOIN actor ON film\_actor.actor\_id = actor.actor\_id
* WHERE actor.first\_name = 'Nick' AND actor.last\_name = 'Wahlberg'

TimeStamp and Extract

Functions

* TIMEZONE
* NOW
* TIMEOFDAY
* CURRENT\_TIME
* CURRENT\_DATE
* SHOW TIMEZONE # to get time zone (Asia/Culcutta)
* SELECT NOW() # to get timestamp with time zone
* SELECT TIMEOFDAY() # returns string
* SELECT CURRENT\_TIME
* SELECT CURRENT\_DATE

Lets explore extracting information form a time based data type using

EXTRACT() : Allows to “extract” or obtain a sub-component of a date value

* EXTRACT(YEAR FROM date\_col)

AGE(date\_col) : returns the current age given a timestamp

TO\_CHAR(): Convert data type to text

* To\_CHAR(date\_col,’mm-dd-yyyy’)
* SELECT EXTRACT(YEAR FROM payment\_date) AS year FROM payment
* SELECT EXTRACT(MONTH FROM payment\_date) AS month FROM payment
* SELECT EXTRACT(QUARTER FROM payment\_date) AS pay\_month FROM payment
* SELECT AGE(payment\_date) FROM payment
* SELECT TO\_CHAR(payment\_date,'MONTH-YYYY') FROM payment
* SELECT TO\_CHAR(payment\_date,'MM/dd/YYYY') FROM payment

<https://www.sqlines.com/oracle-to-sql-server/to_char_datetime>

Challenges: During which month did payments occur? Format your answer to return back the full month

* SELECT DISTINCT(TO\_CHAR(payment\_date,'MONTH')) AS Month FROM payment

How may payment occurred on a Monday?

* SELECT COUNT(\*) FROM payment WHERE EXTRACT(dow FROM payment\_date) = 1

Mathematical Functions:

+,-,\*,/,%,^,^,|/,||/,

<https://www.postgresql.org/docs/9.5/functions-math.html>

* SELECT ROUND(rental\_rate/replacement\_cost,2)\*100 AS percent\_cost FROM film

String Function and Operator:

<https://www.postgresql.org/docs/9.1/functions-string.html>

* SELECT LENGTH(first\_name) FROM customer
* SELECT UPPER(first\_name) ||' '|| UPPER(last\_name) AS full\_name FROM customer

SUB-QUERY : allows you to construct complex queries, essentially peroforming a query on the results of another query. The syntax is straight forword and involves two SELECT statements.

EXISTS operator is used to test for existence of rows in a subquery. Typically a subquery is passed in the EXISTS() function to check if any rows are returned with the subquery.

* SELECT column\_name FROM table\_name WHERE EXISTS (SELECT column\_name FROM table\_name WHERE condition);
* SELECT title,rental\_rate FROM film WHERE rental\_rate > (SELECT AVG(rental\_rate) FROM film

IF Subquery is returning multiple values use IN command

* SELECT film\_id, title FROM film WHERE film\_id IN (SELECT inventory.film\_id FROM rental
* INNER JOIN inventory ON inventory.inventory\_id = rental.inventory\_id
* WHERE return\_date BETWEEN '2005-05-29' AND '2005-05-30')
* ORDER BY film\_id
* SELECT first\_name, last\_name
* FROM customer AS c
* WHERE EXISTS
* (SELECT \* FROM payment as p
* WHERE p.customer\_id = c.customer\_id AND amount >11)

SELF-JOIN:

A self-join is a query in which a table is joined to itself. Self joins are useful for comparing values in a column of rows within the same table

* SELECT tableA.col, tableB.col FROM table AS tableA JOIN table AS tableB ON tableA.some\_col = tableB.other\_col

Find films of same lenth:

* SELECT f1.title, f2.title, f1.length FROM film f1
* INNER JOIN film AS f2 ON f1.film\_id != f2.film\_id
* AND f1.length = f2.length

Assessment Test #2:

1. How can you retrieve all the information from the cd.facilities table?

* SELECT \* FROM cd.facilities

1. You want to print out a list of all of the facilities and their cost to members. How would you retrieve a list of only facility names and costs?

* SELECT name, membercost FROM cd.facilities

1. How can you produce a list of facilities that charge a fee to members?

* SELECT \* FROM cd.facilities
* WHERE membercost != 0
* LIMIT 5

1. How can you produce a list of facilities that charge a fee to members, and that fee is less than 1/50th of the monthly maintenance cost? Return the facid, facility name, member cost, and monthly maintenance of the facilities in question.

* SELECT facid, name, membercost, monthlymaintenance FROM cd.facilities
* WHERE membercost > 0 AND (membercost < monthlymaintenance/50.0)

1. How can you produce a list of all facilities with the word 'Tennis' in their name?

* SELECT \* FROM cd.facilities
* WHERE name LIKE '%Tennis%'

1. How can you retrieve the details of facilities with ID 1 and 5? Try to do it without using the OR operator.

* SELECT \* FROM cd.facilities
* WHERE facid IN (1,5)

1. How can you produce a list of members who joined after the start of September 2012? Return the memid, surname, firstname, and joindate of the members in question.

* SELECT memid,surname,firstname,joindate FROM cd.members
* WHERE joindate > '2012-09-01';

1. How can you produce an ordered list of the first 10 surnames in the members table? The list must not contain duplicates.

* SELECT DISTINCT(surname) FROM cd.members
* ORDER BY surname
* LIMIT 10

1. You'd like to get the signup date of your last member. How can you retrieve this information?

* SELECT MAX(joindate) FROM cd.members

1. Produce a count of the number of facilities that have a cost to guests of 10 or more.

* SELECT COUNT(\*) FROM cd.facilities
* WHERE guestcost > 9

1. Produce a list of the total number of slots booked per facility in the month of September 2012. Produce an output table consisting of facility id and slots, sorted by the number of slots.

* SELECT facid, sum(slots) AS "Total Slots" FROM cd.bookings b
* WHERE starttime BETWEEN '2012-09-01' AND '2012-10-01'
* GROUP BY facid
* ORDER BY sum(slots)

1. Produce a list of facilities with more than 1000 slots booked. Produce an output table consisting of facility id and total slots, sorted by facility id.

* SELECT facid, sum(slots) AS "Total Slots" FROM cd.bookings
* GROUP BY facid
* HAVING SUM(slots) > 1000 ORDER BY facid;

13.How can you produce a list of the start times for bookings for tennis courts, for the date '2012-09-21'? Return a list of start time and facility name pairings, ordered by the time.

=> SELECT cd.bookings.starttime AS start, cd.facilities.name

AS name

FROM cd.facilities

INNER JOIN cd.bookings

ON cd.facilities.facid = cd.bookings.facid

WHERE cd.facilities.facid IN (0,1)

AND cd.bookings.starttime >= '2012-09-21'

AND cd.bookings.starttime < '2012-09-22'

ORDER BY cd.bookings.starttime;

14. How can you produce a list of the start times for bookings by members named 'David Farrell'?

=> SELECT cd.bookings.starttime

FROM cd.bookings

INNER JOIN cd.members ON

cd.members.memid = cd.bookings.memid

WHERE cd.members.firstname='David'

AND cd.members.surname='Farrell';

CREATING DATABASES AND TABLES

Data Types

1. Boolean: True or Flase
2. Character: char, varchar and text
3. Numeric: integer and floating-point number
4. Temporal: date, time, timestamp, and interval
5. UUDI (Universally Unique Identifiers)
6. Array: Stores an array of strings, numbers, etc
7. JSON
8. Hstore key-vlaue pair
9. Special types such as network address and geometric data
10. SERIAL: a sequence is a special kind of database object that generates sequence of intergers

# Save phone number in text based data type

A Primary key: is a column or a group of columns or a group of columns used to identify a row uniquely in a table. Unique and not null

Foreign key: referencing table, Helps to join tables

Constraints are the rules enforced on data columns on table. These prevent invalid data from being entered into the database

1. Column Constraints
2. Table Constraints
3. NOT NUL
4. UNIQUE
5. PRIMARY KEY
6. FOREIGN KEY
7. CHECK: ensures that all values in a column satisfy certain conditions
8. EXCLUSION: ensures that if any two rows are compared on the specified column or expression using the specified operator, not all of these comparisons will return TRUE
9. REFERENCES: to constrain the value stored in the column that must exist in a column another table

CREATE:

CREATE TABLE table\_name( col\_name TYPE col\_constraint, col\_name TYPE col\_constraint, table\_constraint) INHERITS existing\_table\_name

* CREATE TABLE players(
* player\_id SERIAL PRIMARY KEY,
* age SMALLINT NOT NULL);
* CREATE TABLE account(
* user\_id SERIAL PRIMARY KEY,
* username VARCHAR(50) UNIQUE NOT NULL,
* password VARCHAR(50) NOT NULL,
* email VARCHAR(250) UNIQUE NOT NULL,
* created\_on TIMESTAMP NOT NULL,
* last\_login TIMESTAMP
* )
* CREATE TABLE account\_job(
* user\_id INTEGER REFERENCES account(user\_id),
* job\_id INTEGER REFERENCES job(job\_id),
* hire\_date TIMESTAMP
* )
* CREATE TABLE job(
* job\_id SERIAL PRIMARY KEY,
* job\_name VARCHAR(200) UNIQUE NOT NULL
* )

INSERT:

* INSERT INTO table(col1,col2…)
* VALUES
* (value1,vlaue2,…)
* (value1,value2,..)
* INSERT INTO account\_job(user\_id,job\_id,hire\_date)
* VALUES
* (1,1,Current\_timestamp)

UPDATE:

UPDATE table SET col1=value1, col2=value2,…

WHERE condition;

UPDATE account

SET last\_login = CURRENT\_TIMESTAMP

WHERE last\_login IS NULL

* UPDATE TableA
* SET original\_col = TableB.new\_col
* FROM tableB
* WHERE table.id = TableB.id

Return affected rows

* UPDATE account
* SET last\_login = created\_on
* RETURNING accout\_id,last\_login

--SELECT \* FROM account

UPDATE account

SET last\_login = CURRENT\_TIMESTAMP

* UPDATE account\_job
* SET hire\_date = account.created\_on
* FROM account
* WHERE account\_job.user\_id = account.user\_id
* UPDATE account
* SET last\_login = current\_timestamp
* returning email, created\_on, last\_login

DELETE:

DELETE FROM job

WHERE job\_name = 'Cowboy'

RETURNING job\_id, job\_name

ALTER : Allows for changes to an existing table structure. Such as:

* adding, droping, renaming columns
* changing a column’s data type
* set default values for a column
* Add CHECK constraints
* Rename table

ALTER TABLE table\_name

ADD column col\_name

Set default value

* CREATE TABLE information(
* info\_id SERIAL PRIMARY KEY,
* title VARCHAR(500) NOT NULL,
* person VARCHAR(50) NOT NULL UNIQUE
* )
* SELECT \* FROM information
* ALTER TABLE information
* RENAME TO new\_info
* SELECT \* FROM new\_info
* ALTER TABLE new\_info
* RENAME COLUMN person TO people
* SELECT \* FROM new\_info
* ALTER TABLE new\_info
* ALTER COLUMN people DROP NOT NULL

DROP : allow for the complete removal of a column in a table

* ALTER TABLE table\_name
* DROP COLUMN col\_name CASCADE
* ALTER TABLE table\_name
* DROP COLUMN IF EXIST col\_name
* ALTER TABLE new\_info
* DROP COLUMN people

: CHECK:

CREATE TABLE Example(

ex\_id SERIAL PRIMARY KEY,

age SMALLINT CHECK (age>21),

parent\_age SMALLINT CHECK (parent\_age > age) );

CONDITIONAL CONDITION AND EXPRESSION:

CASE: To main ways to use a case statement, either a general CASE or a CASE expression

GENERAL CASE

* CASE
  + WHEN condition1 THEN result1
  + WHEN condition2 THEN result2
  + ELSE some\_other\_result
* END
* select customer\_id,
* CASE
* WHEN (customer\_id <= 100) THEN 'Premium'
* WHEN (customer\_id) BETWEEN 100 AND 200 THEN 'Plus'
* ELSE 'Normal'
* END AS Customer\_class
* from customer

CASE Expression

* CASE expression
  + WHEN value1 THEN result1
  + WHEN value2 THEN result2
  + ELSE some\_other\_result
* END
* select customer\_id,
* CASE customer\_id
* WHEN 2 THEN 'Winner'
* WHEN 5 THEN 'Second Place'
* ELSE 'Normal'
* END AS raffle\_reults
* from customer

Find the count of retnal\_rate = 0.99 and 2.99

* select
* SUM(CASE rental\_rate
* WHEN 0.99 THEN 1
* ELSE 0
* END) AS bargains,
* SUM(CASE rental\_rate
* WHEN 2.99 THEN 1
* ELSE 0
* END) AS regular
* FROM film

COALESCE: function accepts an unlimited number of arguments. It returns the first argument that is not null. If all argument that is not null. If all arguments are null, the COALESCE function will return null.

COALESCE(arr\_1,arg\_2,arg\_3,…..n)

CAST: The cast operator lets you convert from one data type into another. Keep in mind not every instance of a data type can be CAST to another data type, it must be reasonable to convert the data, for example ‘5’ to an integer will work, ‘five’ to an integer will not.