

## MY NOTES FOR AN INTERVIEW IN DATA (SQL USING POSTGRESQL)

### Data Processing Aspects

- Cleaning
- Aggregating
- Joining

Instead of using one machine, clusters of machines used to process the data.

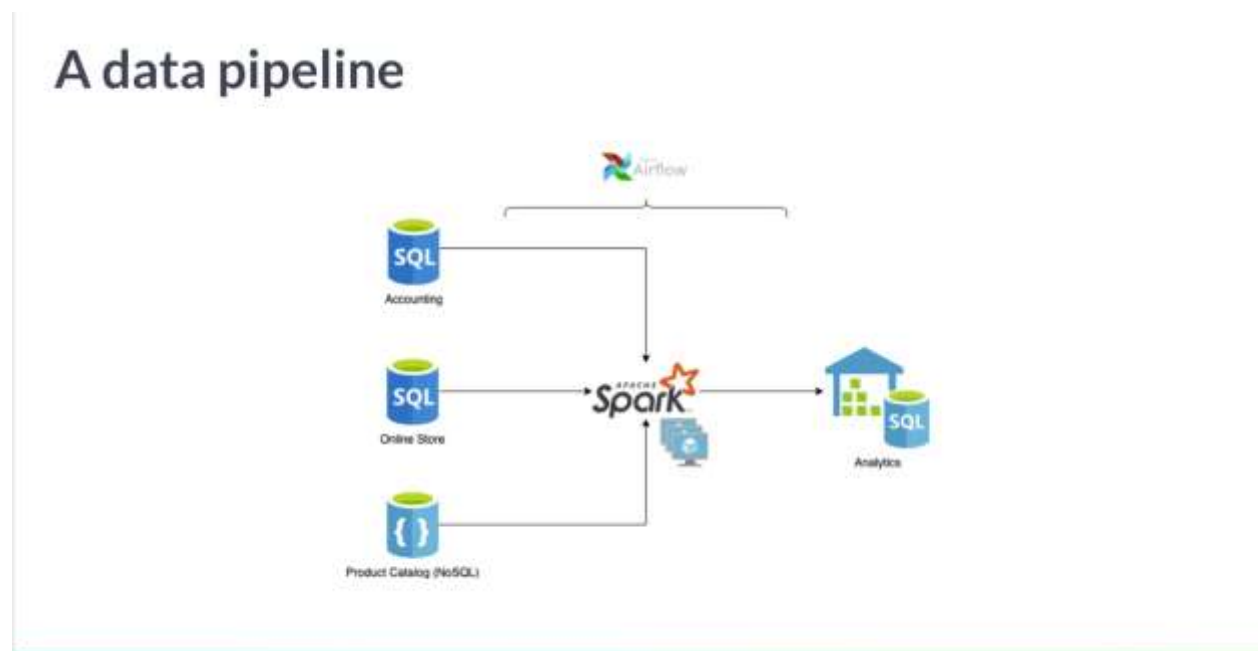
### Scheduling

- Helps to make sure data moves from one place to another at the correct time at a specific interval.
- Plan jobs with specific intervals
- Resolve dependency requirements of jobs

### Tools

- Databases: MySQL, PostgreSQL,
- Processing: Spark, Hive
- Scheduling: Apache Airflow, Oozie

### Data Pipeline



Gist:

Extract all data through connections to several databases → Transform using clustered computing framework like Apache Spark → Load into an analytical database (ETL)

Everything is scheduled to run in a specific order through a scheduling framework like Airflow

Data sources can be external APIs or raw files.

## Cloud Computing

Data Engineers heavily use Cloud.

For data processing in the cloud,

Clusters of machines are required.

**Problem companies face:** Self-Hosting Data Center

**Why** a problem?

- Covering electrical and maintenance cost incurred due to the infrastructure
- It's hard to optimize the so-called 'Peak' and 'Quiet' Moments in the way it is utilized

**Solution:**

**Cloud!**

- Cloud not just conquers cost and space maintenance but also ensures database reliability

## 3 big players in Cloud Provider Market

- AWS
- Microsoft Azure
- Google Cloud

## 3 types of services Cloud Provides:

- Storage
  - Allow upload of files of all types eg. Storing product images
  - Inexpensive
  - AWS uses AWS S3
  - Azure uses Blob Storage
  - Google Cloud Storage
- Computation
  - Perform calculations eg. Hosting a web server
  - Flexible; you can start and stop a virtual machine as per need and time
  - AWS EC2
  - Azure Virtual Machines
  - Google Compute Engine
- Databases
  - Hold structured information
  - AWS has RDS
  - Azure SQL Databases
  - Google Cloud SQL

## From Spreadsheets to Databases

- Databases
  - Data Integrity
  - Can handle massive amounts of data
  - Quickly combine different datasets
  - Automate steps for re-use
  - Can support data for websites and applications

## Database Platform Options

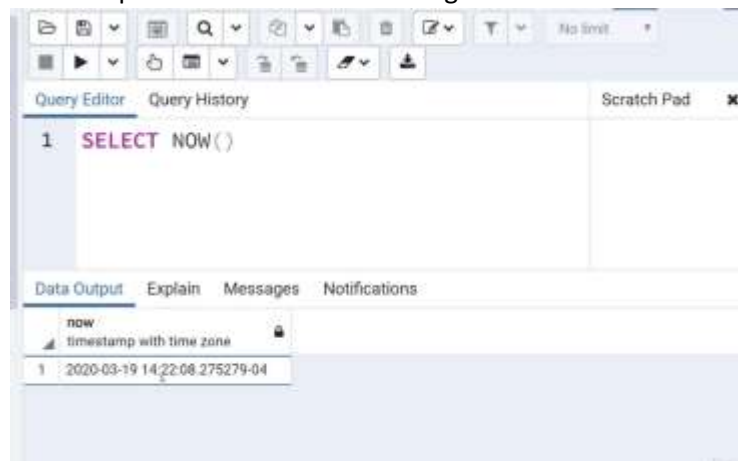
PostgreSQL		Free (Open Source) Widely used on internet Multi platform
MySQL MariaSQL		Free (Open Source) Widely used on internet Multi platform.
MS SQL Server Express		Free, but with some limitations Compatible with SQL Server Windows only
Microsoft Access		Cost (-) Not easy to use just SQL (-)
SQLite		Free (Open Source) Mainly command line (-)

## POINTS TO NOTE:

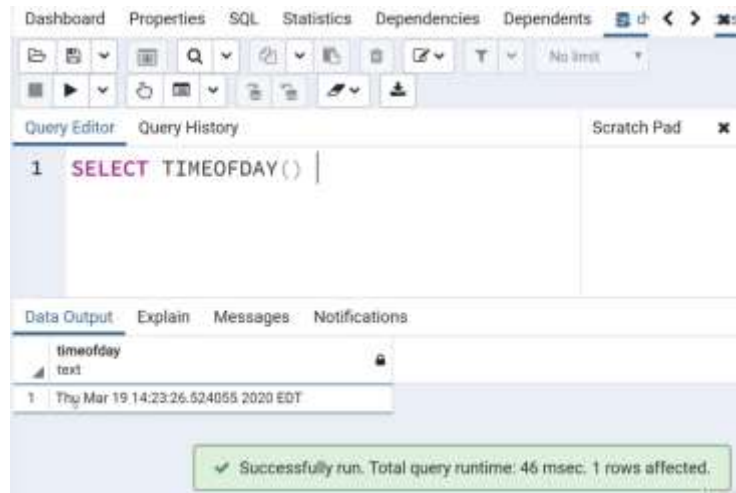
- It's not advisable to use \* in every SELECT statement if not needed; this automatically queries everything which increases the traffic between the database server and the application, which can slow down the retrieval of the results.
- Aggregate function calls happen only in the SELECT clause or the HAVING clause.
- AVG() returns floating point value of many decimal points; we can use ROUND() to decide the number of decimal points we want in our answer.
  - eg. SELECT ROUND(AVG(amount),2) from taxes; // 2 stands for 2 decimal places.
- GROUP BY() allows us to aggregate columns per some category
- We need to choose a categorical column to call GROUP BY on; categorical meaning they are not continuous.
- General use case syntax of GROUP BY:
  - SELECT category\_col, AGG(data\_col) FROM table GROUP BY categorical\_col
  - SELECT category\_col, AGG(data\_col) FROM table WHERE category\_col != 'A' GROUP BY category\_col
- The GROUP BY clause must appear right after a FROM statement or a WHERE statement
- In the SELECT statement, columns must either have an aggregate function or be in the GROUP BY call.
- WHERE statement should not refer to the aggregation results; HAVING can instead be used to filter those results.
- If you want to sort the results based on the aggregate, make sure to reference the entire function.
  - eg. SELECT company, SUM(sales) FROM finance\_table, GROUP BY company ORDER BY SUM(sales) LIMIT 5; // LIMIT function is just extra here for a situation where top 5 sales etc is needed.
- SELECT customer\_id, SUM(amount) FROM payment GROUP BY customer\_id
- HAVING allows us to use the aggregate result as a filter along with GROUP BY.
- The AS operator gets executed at the very end of a query, meaning we cannot use ALIAS inside a WHERE operator.
- INNER JOIN results in a set of records that match in two different tables.
- INNER JOIN is symmetrical.
- OUTER JOINS allow to deal with values only present in one of the tables being joined.
- Types of OUTER JOINS: FULL, LEFT, RIGHT
- General syntax:
  - SELECT \* from TABLE A (OR TABLE B) FULL OUTER JOIN TABLE B( OR TABLE A) ON TABLE A.col\_match = TABLEB.col\_match;
- FULL OUTER JOIN with WHERE is used to get rows unique to both tables being joined that is the values of both tables that don't match. It works completely opposite from INNER JOIN.



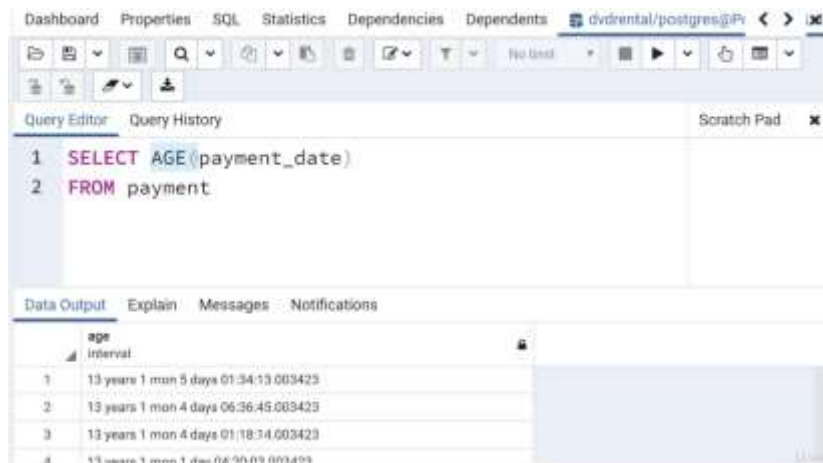
- A LEFT OUTER JOIN (or just LEFT JOIN) results in the set of records that are in the left table. If there's no match with the right table, the results are null.
- General Syntax:
  - SELECT \* FROM Table A (left table) LEFT OUTER JOIN Table B (right table) ON Table A.col\_match = Table B.col\_match;
- LEFT OUTER JOIN results in matched/unmatched; basically, all values from Table A and no values from Table B.
- RIGHT JOIN is the same as LEFT JOIN; only tables are switched.
- UNION operator is used to combine the result-set of two or more SELECT statements
- Directly concatenates two results together, essentially pasting them together
- General syntax:
  - SELECT col\_name(s) from Table1 UNION SELECT col\_name(s) from Table2;
- TIME – contains only time
- DATE – only date
- TIMESTAMP – date and time
- TIMESTAMPTZ – date, time, timezone
- SHOW – shows the value of a run-time parameter
  - name of parameter setting it's currently on, description text of the parameter and its setting
- SHOW TIMEZONE (returned US/Eastern)
- SELECT NOW() – timestamp information of the time right now.



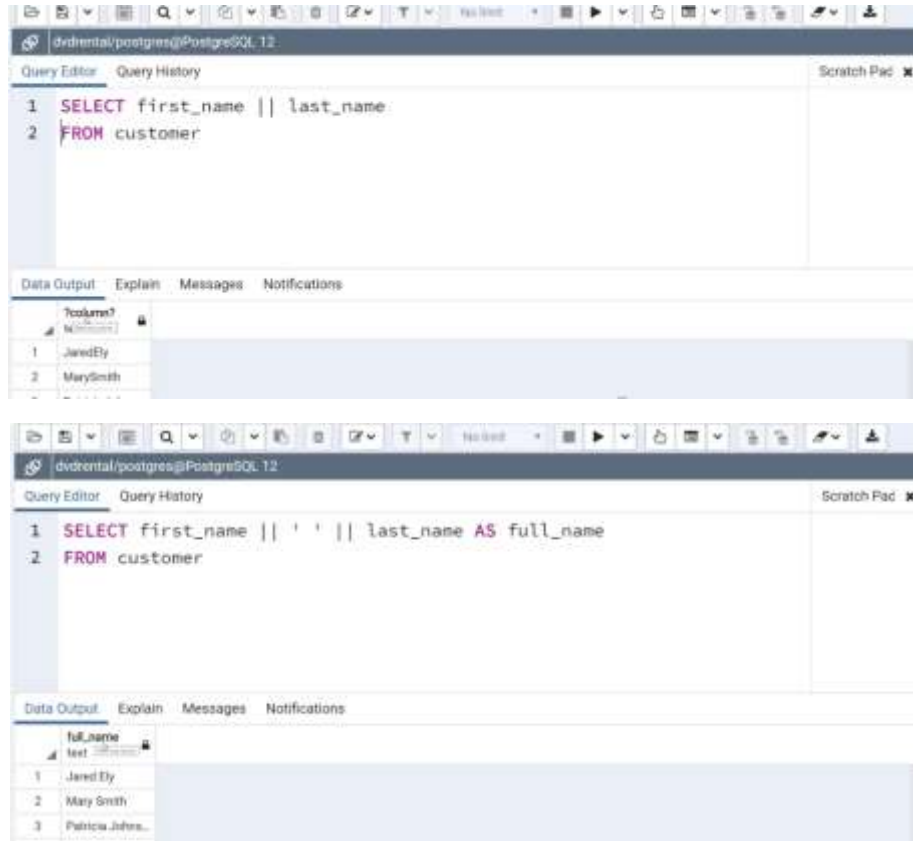
- SELECT TIMEOFDAY()



- SELECT CURRENT\_TIME()
- SELECT\_CURRENT\_DATE() (YYYY-MM-DD)
- EXTRACT() -allows to extract a subcomponent of a date value
  - YEAR
  - MONTH
  - DAY
  - WEEK
  - QUARTER
- eg. EXTRACT(YEAR FROM date\_col);
- TO\_CHAR() – General function to convert data types to text
  - Useful for timestamp formatting
  - Usage: TO\_CHAR(date\_col, 'mm-dd-yyyy')
- SELECT AGE(payment\_date) – tells us how old the timestamp is



- TO\_CHAR() Usage: SELECT TO\_CHAR(payment\_date, 'MONTH-YYYY') FROM payment;
- LENGTH(col\_name) to find length of string text present in the column.
- String concatenation:



- A subquery allows you to construct complex queries, essentially performing a query on the result of another query.
- EXISTS operator checks the existence of rows in a subquery; it's passed to check if any rows are returned with the subquery.



### • Typical Syntax

```
SELECT column_name
FROM table_name
WHERE EXISTS
(SELECT column_name FROM
table_name WHERE condition);
```

- 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.
- It can be seen as a join of two copies of the same table; the table is not actually copied but SQL performs the command as though it were.
- Data Types:



- Boolean
  - True or False
- Character
  - char, varchar, and text
- Numeric
  - integer and floating-point number
- Temporal
  - date, time, timestamp, and interval



- UUID
  - Universally Unique Identifiers
- Array
  - Stores an array of strings, numbers, etc.
- JSON
- Hstore key-value pair
- Special types such as network address and geometric data.

- A primary key is a column or a group of columns used to uniquely identify a row in a table.
- They also play a role in determining which columns should be used to join tables together.
- SERIAL data type which automatically creates unique integers as we add data to the table.
- A foreign key is a field or a group of fields in a table that uniquely identifies a row in another table.
- A foreign key is defined in a table that references to the primary key in another table.
- The table containing the foreign key is called referencing table or child table.
- The table which the foreign key references is called the referenced table or parent table.
- A table can have multiple foreign keys depending on the type of relationships it has with other tables.
- Constraints: rules enforced on data columns in a table.
- Used to prevent invalid data from being entered into the database.
- Also ensures accuracy and reliability of data in the database





SQL

- Constraints can be divided into two main categories:
  - Column Constraints
    - Constrains the data in a column to adhere to certain conditions.
  - Table Constraints
    - applied to the entire table rather than to an individual column.

- Most common column constraints:
  - NOT NULL: ensures no null value is entered
  - UNIQUE: to maintain distinct values
  - PRIMARY KEY
  - FOREIGN KEY
  - CHECK: ensures all values in a column satisfy certain conditions
  - EXCLUSION: ensures that if two rows are compared on the specified column or expression using the specified operator, not all of these comparisons will return TRUE
- Most common Table constraints:
  - CHECK: to check condition when inserting or updating data
  - REFERENCES: to constrain the value stored in the column that must exist in a column in another table.
  - UNIQUE: forces values stored in the column list it accepts as argument to be unique.
  - PRIMARY KEY: allows you to define a primary key that consists of multiple columns (also accepts a column list as argument)

- CREATE TABLE general syntax:



SQL

- Full General Syntax
  - `CREATE TABLE table_name (  
    column_name TYPE column_constraint,  
    column_name TYPE column_constraint,  
    table_constraint table_constraint  
);` INHERITS existing\_table\_name;

- SERIAL: In PostgreSQL, a sequence is a special kind of database object that generates a sequence of integers.

- A sequence is often used as the primary key column of a table.



- **SERIAL**

- It will create a sequence object and set the next value generated by the sequence as the default value for the column.
- This is perfect for a primary key, because it logs unique integer entries for you automatically upon insertion.

PIERIAN DATA



- **SERIAL**

- If a row is later removed, the column with the SERIAL data type will not adjust, marking the fact that a row was removed from the sequence, for example
  - 1,2,3,5,6,7
    - You know row 4 was removed at some point

PIERIAN DATA

- INSERT command on a table
  - General syntax:



- INSERT allows you to add in rows to a table.
- General Syntax
  - INSERT INTO table (column1, column2, ...) VALUES (value1, value2, ...), (value1, value2, ...) ,...;

- INSERT from another table syntax:



- INSERT allows you to add in rows to a table.
- Syntax for Inserting Values from another table:
  - `INSERT INTO table(column1,column2,...)  
SELECT column1,column2,...  
FROM another_table  
WHERE condition;`

- Inserted row values must match up for the table, including constraints.
- SERIAL columns do not need to be provided a value.
- UPDATE keyword allows changing of values of the columns in a table.
- General syntax:



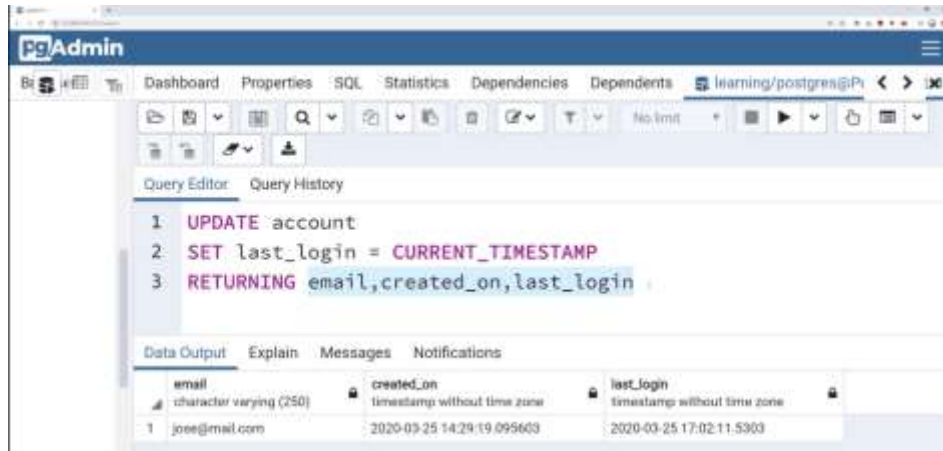
- General Syntax
  - `UPDATE table  
SET column1 = value1,  
column2 = value2 ,...  
WHERE  
condition;`

- Updating from one table, values in another table



- Using another table's values (UPDATE join)
  - `UPDATE TableA  
SET original_col = TableB.new_col  
FROM tableB  
WHERE tableA.id = TableB.id`

- Using SET will set the value to whatever changes you make but just update you that the change has been made successfully. To see the result right after the query without having to run select \* from table on which you made changes, you can use the RETURN keyword:



- DELETE

- General syntax:

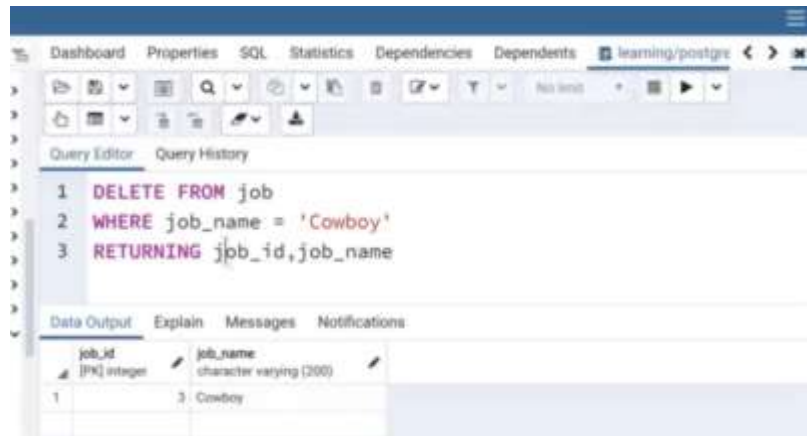


- We can use the DELETE clause to remove rows from a table.
- For example:
  - DELETE FROM table  
WHERE row\_id = 1



- We can delete rows based on their presence in other tables
- For example:
  - DELETE FROM tableA  
USING tableB  
WHERE tableA.id=TableB.id

- To delete all rows from a table:
  - DELETE FROM table;
- Using RETURN keyword to check reflected changes:



- ALTER table



- The ALTER clause allows for changes to an existing table structure, such as:
  - Adding, dropping, or renaming columns
  - Changing a column's data type
  - Set DEFAULT values for a column
  - Add CHECK constraints
  - Rename table
- General Syntax:
  - ALTER table table\_name action;
  - ALTER table table\_name ADD COLUMN new\_col TYPE
  - ALTER table table\_name DROP COLUMN col\_name
  - ALTER table table\_name ALTER COLUMN col\_name SET DEFAULT value //to alter the constraints of an existing column
  - ALTER table table\_name ALTER COLUMN col\_name DROP DEFAULT value
  - ALTER table table\_name ALTER COLUMN col\_name SET NOT NULL
  - ALTER table table\_name ALTER COLUMN col\_name DROP NOT NULL
  - ALTER table table\_name ALTER COLUMN col\_name ADD CONSTRAINT constraint\_name
  - SELECT table table\_name RENAME TO new\_table\_name
  - SELECT table table\_name RENAME COLUMN TO new\_col\_name

- DROP



- DROP allows for the complete removal of a column in a table.
- In PostgreSQL this will also automatically remove all of its indexes and constraints involving the column.
- However, it will not remove columns used in views, triggers, or stored procedures without the additional CASCADE clause.

PIERIAN DATA

- General syntax:
  - ALTER TABLE table\_name DROP COLUMN col\_name
  - To avoid error, check for existence of column you're trying to drop:
    - ALTER TABLE table\_name DROP COLUMN IF EXISTS col\_name
  - Drop multiple columns:
    - ALTER TABLE table\_name DROP COLUMN col\_one, DROP COLUMN col\_two, DROP COLUMN col\_3
- CHECK:



- General Syntax
  - CREATE TABLE example(
    - ex\_id SERIAL PRIMARY KEY,
    - age SMALLINT CHECK (age > 21),
    - parent\_age SMALLINT CHECK (
      - parent\_age > age)

ASSESSMENT:

## ***Assessment Test***

Welcome to your final assessment test! This will test your knowledge of the previous section, focused on creating databases and table operations. This test will actually consist of a more open-ended assignment below:

### **Complete the following task:**

Create a new database called "School" this database should have two tables: **teachers** and **students**.

The **students** table should have columns for student\_id, first\_name, last\_name, homeroom\_number, phone, email, and graduation year.

The **teachers** table should have columns for teacher\_id, first\_name, last\_name, homeroom\_number, department, email, and phone.

The constraints are mostly up to you, but your table constraints do have to consider the following:

1. We must have a phone number to contact students in case of an emergency.
2. We must have ids as the primary key of the tables
3. Phone numbers and emails must be unique to the individual.

Once you've made the tables, insert a student named Mark Watney (student\_id=1) who has a phone number of 777-555-1234 and doesn't have an email. He graduates in 2035 and has 5 as a homeroom number.

Then insert a teacher names Jonas Salk (teacher\_id = 1) who as a homeroom number of 5 and is from the Biology department. His contact info is: jsalk@school.org and a phone number of 777-555-4321.

## *Solutions to Assessment Test 3*

To create the database simply right-click on the databases drop down menu and select "New Database".

Then you can use the following SQL scripts to execute the tasks:

To create the students table:

```
1. CREATE TABLE students(  
2. student_id serial PRIMARY KEY,  
3. first_name VARCHAR(45) NOT NULL,  
4. last_name VARCHAR(45) NOT NULL,  
5. homeroom_number integer,  
6. phone VARCHAR(20) UNIQUE NOT NULL,  
7. email VARCHAR(115) UNIQUE,  
8. grad_year integer);
```

To create the teachers table:

```
1. CREATE TABLE teachers(  
2. teacher_id serial PRIMARY KEY,  
3. first_name VARCHAR(45) NOT NULL,  
4. last_name VARCHAR(45) NOT NULL,  
5. homeroom_number integer,  
6. department VARCHAR(45),  
7. email VARCHAR(20) UNIQUE,  
8. phone VARCHAR(20) UNIQUE);
```

Then for inserting the student information:

```
1. INSERT INTO students(first_name,last_name, homeroom_number,phone,grad_year)VALUES ('Mark','Watney',5,'7755551234',2035);
```

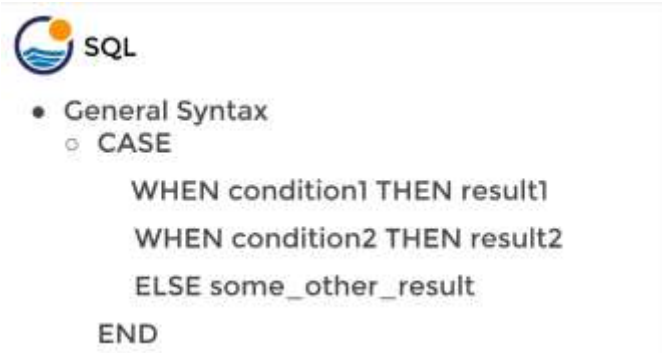
Then for inserting the teacher information:

```
1. INSERT INTO teachers(first_name,last_name, homeroom_number,department,email,phone)VALUES ('Jonas','Salk',5,'Biology','jsalk@school.org','7755554321');
```

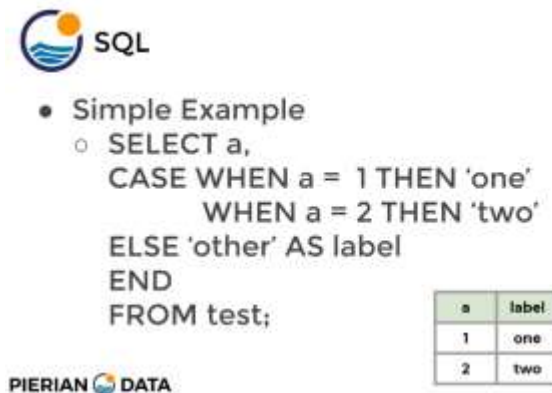


## Conditional Expressions and Procedures:

- CASE statement: used to execute SQL statement only when certain conditions are met.
- This works like IF/ELSE in Java/C++
- Two ways to use CASE:
  - Way 1: General CASE or CASE Expression:



- Example:



- CASE Expression syntax:



- *CASE Expression Syntax:*

- CASE expression

WHEN value1 THEN result1

WHEN value2 THEN result2

ELSE some\_other\_result

END

- Example – CASE statement

```
1 SELECT customer_id,  
2 CASE  
3     WHEN (customer_id <= 100) THEN 'Premium'  
4     WHEN (customer_id BETWEEN 100 and 200) THEN 'Plus'  
5     ELSE 'Normal'  
6 END AS customer_class  
7 FROM customer
```

- Example – CASE expression

```
1 SELECT customer_id,  
2 CASE customer_id  
3     WHEN 2 THEN 'Winner'  
4     WHEN 5 THEN 'Second Place'  
5     ELSE 'Normal'  
6 END AS raffle_results  
7 FROM customer
```

- Implementing COUNT in using SUM and CASE

```
divdental/postgres@postgresSQL 12
Query Editor  Query History
1 SELECT
2 SUM(CASE rental_rate
3     WHEN 0.99 THEN 1
4     ELSE 0
5 END) AS number_of_bargains
6 FROM film
```

- COALESCE function



- The COALESCE function accepts an unlimited number of arguments. It returns the first argument that is not null. If all arguments are null, the COALESCE function will return null.
  - COALESCE (arg\_1,arg\_2,....,arg\_n)

- CAST operator and function



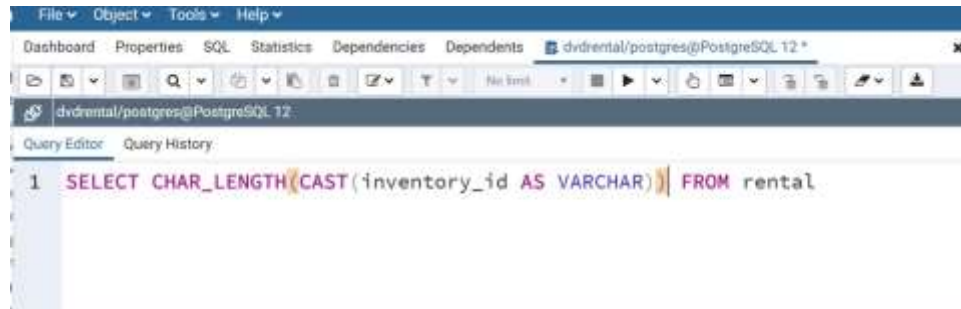
- The CAST operator let's you convert from one data type into another.
- Keep in mind not every instance of a data type can be CAST to another data type, it must be reasonable to convert the data, for example '5' to an integer will work, 'five' to an integer will not.

- CAST implementation – 2 ways:



- Syntax for CAST function
  - SELECT CAST('5' AS INTEGER)
- PostgreSQL CAST operator
  - SELECT '5'::INTEGER

- Example: (Converted integer value to varchar and simultaneously checked the length of each value as if it were a character string and the query returned lengths for all values).

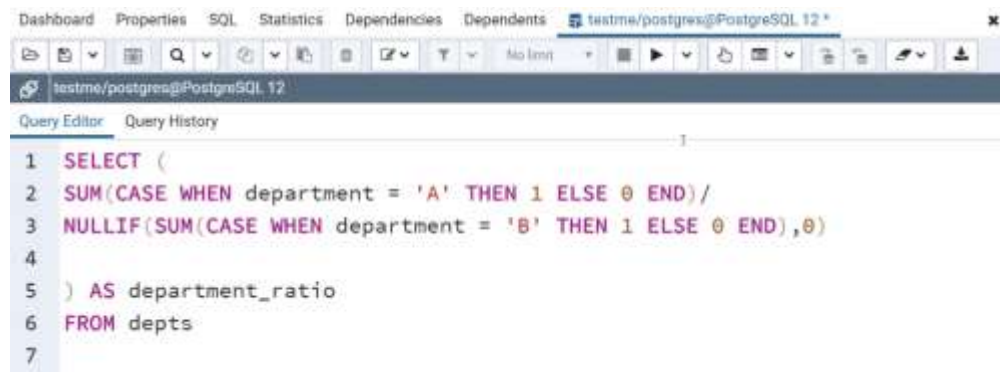


```

1 SELECT CHAR_LENGTH(CAST(inventory_id AS VARCHAR)) FROM rental

```

- NULLIF: used to avoid error in case we end up performing some operation involving a NULL value.
- NULLIF returns NULL if both arguments passed to it match. If they don't match, it simply returns the first argument.



```

1 SELECT (
2 SUM(CASE WHEN department = 'A' THEN 1 ELSE 0 END) /
3 NULLIF(SUM(CASE WHEN department = 'B' THEN 1 ELSE 0 END), 0)
4 ) AS department_ratio
5 FROM depts
6
7

```

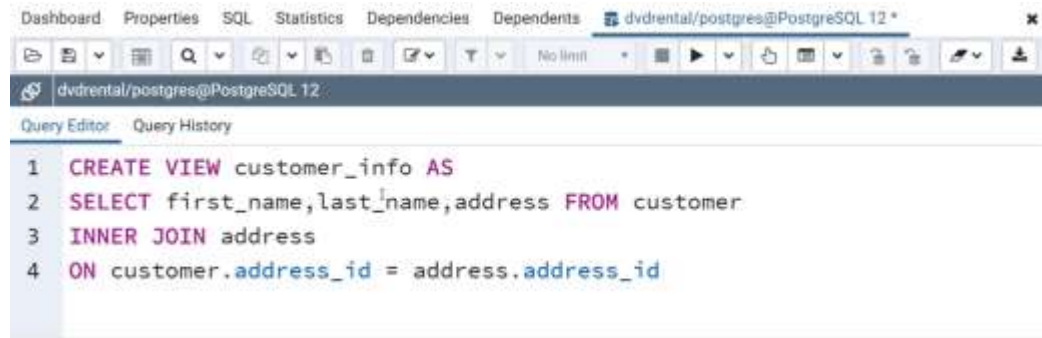
This example returned NULL since the first operation was with a null value and thus came out to be null and the second argument was 0. They didn't match so it returned the first argument i.e. NULL.

- VIEWS



- A view is a database object that is of a stored query.
- A view can be accessed as a virtual table in PostgreSQL.
- Notice that a view does not store data physically, it simply stores the query.

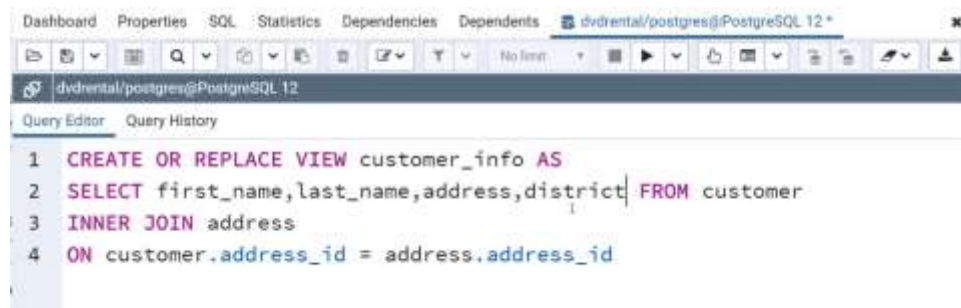
- Example:



The screenshot shows the pgAdmin interface with the 'Query Editor' tab active. The SQL query being entered is:

```
1 CREATE VIEW customer_info AS
2 SELECT first_name,last_name,address FROM customer
3 INNER JOIN address
4 ON customer.address_id = address.address_id
```

Editing that already existing view:



The screenshot shows the pgAdmin interface with the 'Query Editor' tab active. The SQL query being entered is:

```
1 CREATE OR REPLACE VIEW customer_info AS
2 SELECT first_name,last_name,address,district FROM customer
3 INNER JOIN address
4 ON customer.address_id = address.address_id
```

- Deleting a VIEW:
  - DROP VIEW IF EXISTS view\_name
- Renaming a VIEW:
  - ALTER VIEW view\_name RENAME to new\_view\_name
- Important Notes on Importing/Exporting files:
  - **Important Note!**
    - Not every outside data file will work, variations in formatting, macros, data types, etc. may prevent the Import command from reading the file, at which point, you must edit your file to be compatible with SQL.
- Import command does not automatically create a table out of the external csv file. It assumes that the table is already created.
- What you do:
  - Create a table identical to the csv file's records in PostgreSQL; the values in the file will then simply be populated into the created table

- Right click the table; Select Import/Export; pass the location of the filename on your system corresponding to the 'Filename' field.
- Toggle to the 'Options' tab and switch to 'Yes' for 'Header' field; Header means the first row that contains the column names; this is done to ignore that first row from being copied since we already named our columns while creating the table skeleton.
- Look out for other values that may be present in the data being imported including delimiter, quotes etc and make apt settings accordingly.
- Done!
- Now, exporting:
  - Right click on table; select import/export; select the columns you want to be exported
  - In the 'Filename' field, give the location of the file (along with the name you want for the file)
  - Click ok. Done!
- Resources to check out methods of importing/exporting:
  - <https://stackoverflow.com/questions/2987433/how-to-import-csv-file-data-into-a-postgresql-table>
  - <https://www.enterprisedb.com/postgres-tutorials/how-import-and-export-data-using-csv-files-postgresql>
  - <https://stackoverflow.com/questions/21018256/can-i-automatically-create-a-table-in-postgresql-from-a-csv-file-with-headers>