SQL EXAM CHEAT SHEET

1. Inner Joins: Using Postgres, produce a report containing Employee_Name and calculated years of service (YOS) as of September 1st, 2023. Limit the report to employees where YOS > 30. Order the output alphabetically by Employee_Name.
    a. **TAGS: (JOIN, DATES, ORDER BY)**

```
select "Employee_Name", floor((TO_DATE('01SEP2023', 'DDMONYYYY') -
TO_DATE("Start_Date", 'DDMONYYYY'))/365.25) as YOS from
"Jupiter".employee_information i, "Jupiter".employee_addresses a where
i."Employee_ID" = a."Employee_ID" and floor((TO_DATE('01SEP2023',
'DDMONYYYY') - TO_DATE("Start_Date", 'DDMONYYYY'))/365.25) > 30 order by 1;
```

2. Creating a Summary Report from Two Tables: The head of the Sales Department wants to know how many of each product was sold since the beginning of 2010 (determined by Order_Date). The report should include the product ID, the product name, and the total sold for that product (determined by the sum of Quantity). Order the report by descending Total Sold, and ascending Product_Name.
    a. **TAGS: (JOIN, DATES, GROUP BY, ORDER BY)**

```
select p."Product_ID", p."Product_Name", sum(o."Quantity") as Total_Sold
from "Jupiter".product_dim p, "Jupiter".order_fact o
where p."Product_ID" = o."Product_ID" and EXTRACT(YEAR FROM
TO_DATE(o."Order_Date", 'DDMONYYYY')) >= 2010
group by 1,2
order by 3 desc, 2;
```

3. Inner Joins - SQLite: Using SQLite and the practice database, create a report that lists the name of the person, and the name of the movie they own. Order the report by ascending Name.
    a. **TAGS: (JOIN, ORDER BY)**

```
select p.name, m.movie_name
from people p, people_movies pm, movies m
where p.id=pm.person_id and pm.movie_id=m.id
order by 1;
```

4. Creating a Summary Report from Two Tables – SQLite: Using SQLite and the practice database, provide a report that lists the person name and the count of movies owned by person. Provide the count of movies in descending order.
    a. **TAGS: (JOIN, GROUP BY, ORDER BY)**

```
select p.name, count(*) as count_movies
from people p, people_movies pm
where p.id=pm.person_id
group by 1
order by 2 desc;
```

5. Using a Noncorrelated Subquery (Postgres): The jupiter.order_fact table contains information about orders that were placed by customers. Create a report that lists the retail customers whose average retail price exceeds the company average retail sales.
    a. **TAGS: (SUBQUERY, HAVING, GROUP BY, ORDER BY)**

```
select AVG(CAST(REPLACE(REPLACE("Total_Retail_Price",'$',''),',','') AS
DOUBLE PRECISION)) as AVG
from "Jupiter".order_fact where "Order_Type"=1;
select "Customer_ID",
AVG(CAST(REPLACE(REPLACE("Total_Retail_Price",'$',''),',','') AS DOUBLE
PRECISION)) as MeanPrice
        from "Jupiter".order_fact
        where "Order_Type"=1
        group by 1
        having AVG(CAST(REPLACE(REPLACE("Total_Retail_Price",'$',''),',','')
AS DOUBLE PRECISION)) > (select
AVG(CAST(REPLACE(REPLACE("Total_Retail_Price",'$',''),',','') AS DOUBLE
PRECISION)) as AVG
        from "Jupiter".order_fact
        where "Order_Type"=1)
        order by 2 desc;
```

6. Using a Noncorrelated Subquery (Postgres): Each month, a memo that lists the employees who have birthdates for that month is posted. Create a report for the month of September and list Employee_ID and the first and last names for all employees who have birthdates during the month of September.
    a. **TAGS: (DATES, SUBQUERY, ORDER BY)**

```
select "Employee_ID"
from "Jupiter".employees
where EXTRACT(MONTH FROM TO_DATE("Birth_Date", 'MM/DD/YYYY'))=9
order by "Employee_ID";

select "Employee_ID","Employee_Name"
from "Jupiter".employee_addresses
where "Employee_ID" in
                (select "Employee_ID"
from "Jupiter".employees
where EXTRACT(MONTH FROM TO_DATE("Birth_Date", 'MM/DD/YYYY'))=9
order by "Employee_ID")
order by 2;
```

7. Queries (SQLite): Find all movies that are NOT one of the following genre categories: 'Comedy','Comedy/Drama','Exercise','Fantasy','Foreign','Animation',' Horror','TV Classics','VAR','War' • Display only the movie name • Order the report by descending movie name
    a. **TAGS: (SUBQUERY)**

```
select movie_name
```

```
from movies
where genre_id NOT IN (select id from genres where genre in
('Comedy','Comedy/Drama','Exercise','Fantasy','Foreign','Animation','Horror',
'TV Classics','VAR','War'))
order by movie_name desc;
```

8. Queries (SQLite): Find the names of the people who own the following movies: • Movie_ID = '20372','8727','31670' • Note that in the table people_movies, the column ID refers to the ID of the table, and person_id refers to the ID of the person. Refer back to the last slide of Class #3 for a diagram of the tables. • Order the report by ascending person name
    a. **TAGS: (JOIN, ORDER BY)**

```
select p.name
from people p, people_movies pm
where p.id=pm.person_id and movie_id IN ('20372','8727','31670')
order by p.name;
```

9. In Postgres, create a report that displays the total salary for female and male sales representatives and the total number of female and male sales representatives. The jupiter.salesstaff table contains information about all the Jupiter Star sales representatives, including Salary and Gender.
    a. **TAGS: (LIKE, UNION)**

```
select 'Total Paid to All Female Sales Representatives',
sum(CAST(REPLACE(REPLACE("Salary",'$',''),',','') AS INTEGER)),
count(*) as total
from "Jupiter".salesstaff
where "Gender"='F' and "Job_Title" like '%Rep%'
union
select 'Total Paid to All Male Sales Representatives',
sum(CAST(REPLACE(REPLACE("Salary",'$',''),',','') AS INTEGER)),
count(*) as total
from "Jupiter".salesstaff
where "Gender"='M' and "Job_Title" like '%Rep%';
```

10. In Postgres, create a report that displays the Employee_ID of employees who have phone numbers, but do not appear to have address information. The jupiter.employee_phones table contains Employee_ID and Phone_Number. If an employee's address is on file, the jupiter.employee_addresses table contains the Employee_ID value and address information.
    a. **TAGS: (EXCEPT, ORDER BY)**

```
select "Employee_ID"
from "Jupiter".employee_phones
except
select "Employee_ID"
```

```
from "Jupiter".employee_addresses
order by "Employee_ID";
```

11. In SQLite, list the movies that are not owned by anyone. Order the report by ascending movie_id
    a. **TAGS: (EXCEPT, ORDER BY)**

```
select id
from movies
except
select movie_id
from people_movies
order by 1;
```

12. In SQLite, how many movies in the list are not owned by anyone?
    a. **TAGS: (EXCEPT)**

```
select count(*)
from (select "id"
from movies
except
select "movie_id"
from people_movies)
```