SQL Queries

**BETWEEN CLAUSE**

**select** \***from** customer **where** customer\_id **between** 1 **AND** 10 -- returns customer between 1 and 10 (both inclusive)

**select** \***from** customer **where** customer\_id **not** **between** 1 **AND** 10 -- returns customer not between 1 and 10 (both inclusive)

**Sub Query**

--Want to find films whose rental rate is higher than average rental rate

**select** film\_id,title,rental\_rate **from** film **where** rental\_rate>(**select** AVG(rental\_rate) **from** film)

**COUNT**

**select** count(**distinct** amount) **from** payment;

--Calculating number of dates made between two dates

**select** count(\*) **from** payment

**where** payment\_date **between**

'2007-02-07' **and** '2007-02-15';

**select** count(amount) **from** payment **where** amount>5.00;

**select** count(first\_name) **from** actor **where** first\_name **like** 'P%';

**SELECT** COUNT(**DISTINCT** district) **FROM** address;

**SELECT** **DISTINCT** district **FROM** address;

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

**LIMIT**

**select** \* **from** payment

**where** amount=4.99 limit 10;--returns top 10 rows

**ORDER BY CLAUSE**

**select** first\_name,last\_name

**from** customer

**order** **by** first\_name

**DESC**;

**select** first\_name,last\_name

**from** customer

**order** **by** first\_name

**ASC**,last\_name **desc**;

--It will select first\_name and last\_name from customer table and result will be order by first\_name ascending.

--if there is more than one first\_name there last\_name name will be ordered by descending.

**select** first\_name

**from** customer

**order** **by** last\_name;

--we are selecting first\_name and ordering by

--last name. Here order by column is not

--displyaed. it is unique in Postgres

--which is not allowed in other sql database

--default last\_name order by Ascending.

--in above query first\_name whoose last\_name starts with a will be selected first.

**LAST 10 ROWS**

--payment\_id is primary key and sequencely increasing

--here it is very easy to find last 10 rows

**select** \***from** payment **order** **by** payment\_id **desc** limit 10

IN

--here in statement equivalent to customer\_id=7 or customer\_id=13 or customer\_id=10;

**select** customer\_id,rental\_id,return\_date

**from** rental

**where** customer\_id **not** **in**(7,13,10)

**order** **by** return\_date **desc**;

**LIKE CLAUSE**

**SELECT** first\_name,last\_name **FROM** customer **where** first\_name **like** '%er%';

**SELECT** first\_name,last\_name **FROM** customer **where** first\_name **like** '\_her%';--in result \_ will be replaced by any character

**SELECT** first\_name,last\_name **FROM** customer **where** first\_name **not** **like** 'Jen%'

**GROUP BY CLAUSE**

The GROUP by clause divides the rows returned from the select statement into groups.

for each group we can apply aggregate functions.

--please note in other sql engines we have to include group by column in select

**SELECT** customer\_id **FROM** payment **GROUP** **BY** customer\_id;--gives unique id's

**SELECT** MAX(amount),customer\_id **FROM** payment **GROUP** **BY** customer\_id **order** **by** customer\_id **desc**;

**SELECT** SUM(amount) **FROM** payment **GROUP** **BY** customer\_id;

**SELECT** customer\_id,SUM(amount) **FROM** payment **GROUP** **BY** customer\_id;

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

**SELECT** staff\_id,COUNT(\*) **FROM** payment **GROUP** **BY** staff\_id;--number of transactions made by staffs

**SELECT** rating,COUNT(\*) **FROM** film **GROUP** **BY** rating;

**HAVING CLAUSE**

We often use the HAVING clause in conjunction with the group by clause to filter group rows that do not satisfy a specified condition.

**SELECT** customer\_id, SUM(amount)

**FROM** payment

**GROUP** **BY** customer\_id

**HAVING** SUM(amount)>200;

**SELECT** customer\_id, SUM(amount)

**FROM** payment

**GROUP** **BY** customer\_id

**HAVING** customer\_id<100;

**DIFFERENCE BETWEEN HAVING AND WHERE**

The HAVING clause sets the condition for group rows created by the GROUP BY clause after the GROUP BY clause applies While the WHERE clause sets the condition for the individual rows before GROUP BY clause applies

Just applying WHERE

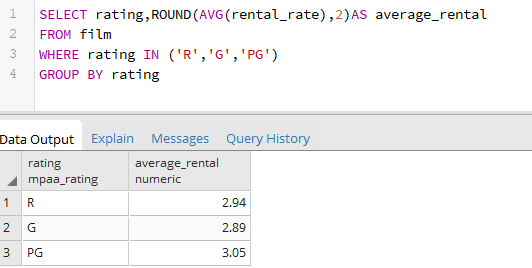
**SELECT** rating,ROUND(AVG(rental\_rate),2)**AS** average\_rental

**FROM** film

**WHERE** rating **IN** ('R','G','PG')

**GROUP** **BY** rating

Output-



Now apply HAVING

**SELECT** rating,ROUND(AVG(rental\_rate),2)**AS** average\_rental

**FROM** film

**WHERE** rating **IN** ('R','G','PG')

**GROUP** **BY** rating

**HAVING** AVG(rental\_rate)>2.90

