Database Management & Big Query

why prefer database over excel?

1) Excel limited to 1 million rows.
2) Data in excel might not be secured.
3) Don't support Concurrency.
4) Data integrity.
5) Performance.

2) Data Types in Sql

a) Int
b) Char(n) → fixed size string of length n.
c) Varchar(n) → Variable size string of length upto n.
d) float(n,y) → n #s before decimal point & y # after.
e) Date → yyyy-mm-dd
f) Time → hh-mm-ss
g) DateTime → yyyy-mm-dd hh-mm-ss
h) Timestamp → DateTime as an universal time co-ordinate.

Keys

1) Primary key → unique + not null
2) Unique key → Unique + can be null
3) Foreign key → It is a primary key in some other table.
4) Candidate key → It is a combination of columns to make a primary key.

Query Code :-

1) ~~Select~~ :- Select rows of table.

2) where :- Conditional Filtering.

between :- Select * from movies
where
year between x and y;

3) select * from movies
where title like "Toy Story %";

% Special operator.
- Match with zero or more characters.
- Can be used only with like or not like.

- where is always written after the from clause.

String Manipulation functions :-

1) concat(x, y, ... z)

2) lower/upper(x)

3) Substr(column, start index, length) to extract.

% length of substring

Q:- why = for comparision instead of ==?

- In sql there is no assignment operator so = is used instead.

- For other languages who have variable assignment == is used for comparision instead.

% used to skip rows.

% offset is always used after limit.
% limit is compulsory for offset to work.

Between :- Between is used to retain range (both inclusive).

ex:- Find booth assignments for vendor id 7 b/w dates 2019-04-03 & 2019-05-16 :-

```
Select * from vendor_booth_assignments
where vendor_id = 7 and market_date between '2019-04-03'
and '2019-05-16';
```

Syntax of the code & order of execution :-

Code :-

```
Select
from
where
order by
limit
offset
```

Order of Execution :-

```
from
where
select
order by
offset
limit
```

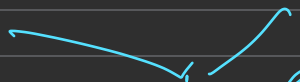
Entity Relationships → Relationship b/w tables in ER diagram.

1) one to one → ||—|| → one customer has one user id.

one user_id is assigned to one customer.

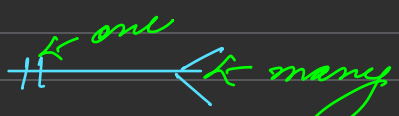
- Very rarely seen in ER diagram.
- Most often, the two tables can be combined into one table.

ex → c_id | ... + v_id | ...



c_id | v_id | ....

2) one to many / many to one →



ex → (1) A brand can make many products.

1 : many.

A product can be made by only one brand.

1 : 1

→ brand ||—< products.

2) A teacher can teach many students.

||—<

A student can be taught by only one teacher in a class.

||—||

3) Child & Parents →

1 child can have 1 parents. ||—||

1 parents can have many children. ||—<

>—|| (many to one)

3) Many to Many →

ex → Book & Author →

1 book can have many authors.
1 author can have many books.

(many to many).

→ Languages involved in sql →

1) DDL → Data definition language.
- used for developmental work on a table.
Drop/ create/ Insert/ Alter table.

2) DML → Data Manipulation Language.
- used for CRUD operations.

3) DCL / TCL → Data & Transaction control language.
↳ Commit/ Rollback
↳ Grant/ Revoke.

4) DQL → Data query language.
→ There are only 10 clauses in DQL.
select, from, where, order by, limit, offset

Filtering & Sub-queries

where → filters things out.

Exact match (=, !=, in, not in)

Partial match → Used to partially match strings.
↳ like
"%" → match any # of characters.
_ → match only one character.

Ex → select *
from customers
where lower(first_name) like "%ana_";

match all the customers who have ana in their penultimate position.

Distinct → Returns unique values in a column.

If 2 columns → Returns all the unique combinations of the columns.

→ Considers Null as an unique value.

ifnull (expression/column, value to replace with) →

if the value is null, replace it with value in 2nd parameter.

→ Subqueries → Executing queries inside queries.

1) Subquery used in where section of outer query.
2) one can be used to include multiple subqueries.
3) subquery should return a list or a single value.
4) Inner subquery gets executed first.

Window functions → (Aggregation functions)

- Always creates a new column.
-

count(colum), sum( ), min/max, avg, lead, lack

Over → Creates a window over the specified conditions.

ex → `select *, max(salary) over() from employees;`

↳ Considers the entire table as one window.

→ **partition by →** Divides the table into multiple windows based on the # of unique elements in the partition by column.

→ **order by →** Sort according to the order by column in the partitioned window.

Groups → ntile → Dividing the dataset into n groups of fixed sizes.

ex → select ntile(4) over(order by salary),
salary
from employees
order by employees;

Approach →

```
select
from ( select ... ntile( ... )      - Table Pre-Processing
         :                          - Executed first.   → query2(inside)
       order by ...
     )
group by ...                         → query1(outside)
order by                             - Executed second.
```

2 → lead & lack →

lead → shift the column up by n steps.

- Initiates bottom n rows to null.

lag → shifts the column down by n steps.

- Initializes the first n rows to null.

ex → month on month analysis →

Compare sales of current month w.r.t sales (next month)

```
select ntile(4) over(order by month) as quarter,
           sum(sales) as sales
       lead(ntile(4) over(order by month), 1) as next_month
from ( select extract(month from market_date) as
         month, sum(quantity) as sales
       from customer_purchases
       group by 1
       order by 1
     )
group by 1
order by 1;
```

→ Cumulative Sum using window fn →

date	sales	1	2
d_1	100	900	100
d_2	200	900	300
d_3	100	900	400
d_4	500	900	900

1) Sum(sales) over( ) → 900 → Returns the total sum for each row.

2) Sum(sales) over(order by date) → Returns the cumulative sum for each row.