# DBMS-1 Day 1 Summary

- DBMS => Store, retrive, manipulate data
- RDBMS => DBMS + Relational b/w tables
  - includes apart from tables
    - Schema => Define table & R/w b/w them.
    - Instance => no. of data / data @ a given data stored.
    - key => uniquely get the row.
    - Domain => Values permited in attribute/column.
- Database Objects = DB + tables
- key
  - Candidate key => unique identify row; many
  - primary key => one of candidate key
  - alternate key => {candidate key} - {primary key}
  - foreign key => linked to primary key of other table
  - unique key => no. duplicate + only one null permitted.
- Commands in broad categories
  - DDL => structure of DB & table
  - DML => content / row / record of table.
  - DQL => select
  - DCL
  - TCL
- DDL
  - Create (DB, table), Alter (table), Drop (DB, table), Truncate (the table {is most used}), rename (the table {is most used}) -> to
  - Apply on Database objects {DB, tables}
- DML => [Mostly] on records of table
  - Insert (values), update (set), delete (from)
- Missing value => use is null, is not null
- Alter => Change DB structure {column}
  - Change => column name, data type, constraint
  - Modify => datatype, constraint, {no name change}
  - add => column {use [After 'col_name'] for specific}
  - drop => column

↳ Data types (mainly)

- Numeric
  - Integer (exact value) => TinyInt, SmallInt, MediumInt, Int, BigInt
  - fixed (exact value) => Decimal (length, after decimal length)
    ↳ use for financial
  - Floating (approx.) => float, double
    ×  2×
    ↳ not use for financial
  - Bit (0/1)
- String
  - varchar, char
  - enum => predetermined values only allow to input
  - set
  - Blob, Text
- Date Time
  - Date — YYYY-MM-DD
  - Time — HH:mm:SS
  - DateTime
  - TimeStamp => But max is in 2030s.
  - Year — YYYY

- CharLength() vs Length()
  ↳ length/no. of character were written in english.
  ↳ Storage bytes
- != or <>
- curdate()
- now()
- (curdate()) + interval 1 day/month/year
- Show databasename;
- Show tablename;
- describe tablename;

# DBMS 1 day2 Summary

## Constraints

- rules for data
- use/done by create or alter
- alter + add => add new constraint to existing column
- alter + drop => remove existing constraint from existing column
- e.g. NOT NULL, UNIQUE, PRIMARY KEY, DEFAULT, CHECK (condi.), FOREIGN KEY
- two level (mostly) → column level
  - table level (e.g. foreign key)

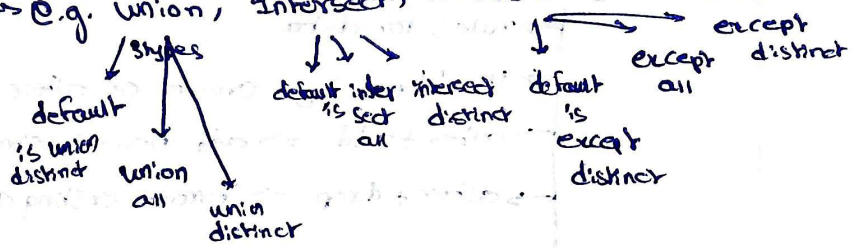
## Where Caluse

- to filter / filter out the records based on conditions.
- use/done with predicates
  - expression that return boolean value.
- predicates
  - in
  - between val1 and val2
  - is NULL, is NOT NULL
  - pattern by => like (with) => wild filtering
  - comparison op. (e.g >, >=, <, <=, etc)

## Wild filtering

- is the method to match the column and to get that row when that column is matched with a pattern.
- wild card => is a character used for pattern.
  - e.g. %, _, -, [ ], [^ ], #
- wild card used with like, not like, where caluse [in most of the times]
- % → only wild card as of now lessons for zero or any no. of chars.
- _ => can be used for n times
- mix of wildcard is also allowed.
- escape operator used for wild card / pattern to be used as data

Sets
- to join two or more Select Stmt.
- e.g. union, intersect, minus [keyword except]




duplicate records
- avoid it by use of : Select Stmt with distinct keyword.
- arraise e.g. when different sources data mingled.

Operator precedence :
- order in which operator execute in expression.
- Interval ; ! ; -(unary) ; */%. ; -+ ; [comparison operator (e.g >, >=) ; is ; in ; like] ; [Between ; case operator] ; NOT ; and ; or ; assignment op.

Built-in function
- categorized based on datatype they operate.
- Mostly for String, Numeric, Date
  - e.g. LOWER()
  - e.g. ABS()
  - eg date_add()

Other functions (to mention explicitly)
- Coalesce => return 1st non-null value in the record. (record by record if do operation for a table, Select Stmt, etc.)
- Cast (value as datatype)

Conditional flow Stmt. (both are used in Select Stmt for ex.)
- if ( condition, true_value, false_value)
- case when cond1 then res1
  when cond2 then res2
  :
  when condn then resn
  else res
  end column name.

Sorting
- using order by clause
  - default ascending order.
  - keywords: asc, desc

* Order by
  - Column name 1 asc/desc, Column name 2 asc/desc, - - -
  - Column number 1 asc/desc, Column number 2 asc/desc, - -

* ifnull (x, x)
  - x: column name to check
  - x: show value if the condition is true.

* Order of table creation depends on the dependency (Parent / 1st created) or dependency (Child / 2nd created)
  - Parent has the primary key
  - Child has the foreign key

* limit → shows that many records in top of result-set
  offset → skip that many records in top of result-set.

  Order of execution is offset (1st) then limit (2nd)
  Order of syntax is limit (1st) then offset (2nd)
  - except in case of syntax: limit <offset value>, <limit value>

# DBMS1 - Day3 Summary

↳ Aggregation function ⇒ for given column it do calculation considering rows values in that column and return a single value.

- → e.g. count(), sum(), Avg(), min(), max()
- → also not consider null values in calculation.
- → if no values due to filtering then return 0.

↳ Group by clause

- → for a specified column, it groups rows that are with same value into group.
- → all other column (i.e not mentioned in Group by clause / non-grouping column) are allowed only with agg. func. in select/ having clauses. (not as direct)
- → also groups null values into one group.

| Grouping column | non grouping column |
|---|---|
| direct use in select / having | use only with agg. func. |

↳ Grouping / Group by clause conditions

- all Grouping columns in select stmt. must be in Group by clause.
- not allowed use of agg. func. in Group by clause
- not allowed use of comparision operators (<, <=, >, >=, etc) in Group by clause.

- Grouping with agg. func. in select and/or having clause.
- Multiple columns in Group by clause.
- Group by clause column should be less unique and 2 primary business entities / fact, not text data.

↳ Having clause with aggregation func.

- Having clause is used to filter the groups based on aggregated data (e.g. count, sum, etc)
- Conditions
  - non grouping columns (or also say as columns not in group by clause) should be/must be used with agg. func. in both having and select stmt.
  - else can directly use in both having (grouping columns) & select stmt.

| select | Group by | having |
|---|---|---|

- with null condition
  - ~~having agg. fun (grouping column)~~
  - having agg. fun (non-grouping column_name) is not null;
- for quick summary → direct use of having clause without Group by clause

e.g.

```
select sum(salary) from employee
having sum(salary) > 100000;
```

Order of execution

from (source)
↓
where (row-level filter)
↓
Group by (groups based on specified column row values into groups)
↓
having (filter groups based on agg. data)
↓
select (show needed data)
↓
order by
↓
offset
↓
limit

Some of concepts

↳ functions

(numeric) → round( ), random( ), random(seed), truncate( )

(string) → replace( ), Soundex( )
↳ case sensitive

(date) → date_add( ), date_format( , ) (date to string) VS Str_to_date( , ) (string to date)

→ date( ), day( ), year( ), minute( ), second( )

→ Implicit type conversion is sql. (i.e. loosly typed lang.)

→ order of execution ≠ order of syntax

→ Explain Analyze Query

→ with clause | common table expression

```
with <CTE name> as
  Query
  Select ... ;
```

↳ CTE name store temporarily that query and also to use as many times within ';' delimiter.

↳ Lpad (org. String, length in final, append value)

↳

| min<br>max | least<br>Greatest |
|---|---|
| Both agg. func.<br><br>works Column | Both nor agg. func.<br><br>works row wise |
| eg. gives max of Colum | e.g. gives Greatest of row |

↳ truncate

truncate → as → DDL Command truncate

truncate → function truncate( )

# DBMS-1 Day 4 Sum up

## Joins

→ is pooling data from various tables into single result-set (so, it is data source for the query)
{based on Key columns} ↓ gives
it is part of from clause

→ $T_1$ join $T_2$
$T_1$ → left table, $T_2$ → right table

$T_1$ join $T_2$ join $T_3$
$T_1$ join $T_2$ → left table, $T_3$ → right table

'join' checks every record of left table with every record of right table based on key columns.

→ Classification based on use of 'on clause' & 'using clause'

- if used any one → joins
  - inner
    - equi → use of [on & =] or [using]
    - non equi → used [on & not = op] or [not using]
  - left
    - equi
    - non equi → ['lly]
  - right
    - equi
    - non equi → ['lly]
  - full outer
    - equi
    - non equi → ['lly]
- not used both → joins
  - Natural (always equi) → Natural join
    - equi inner
    - equi left
    - equi right
    - equi full outer
  - Cross join → gives Cartesian product {m×n rows}

→ using ⇒ always for common column names in left & right table. & use [= op implicitly]
→ it gives single column for each common column in the result-set of join.
e.g. $T_1$ join $T_2$ using c1, c2, ...
(mention in this using clause)

→ Natural join → use ['implicitly = op] for all common column names in left & right table.
→ mostly not used in real world.
→ gives single column for common column name in result

→ joins works even if no. of rows & no. columns are different in left & right tables.

→ keywords:
1 inner
2 left
3 right
4 Natural
5 Cross

→ Note: Full outer join is union of left & right join.

→ Note: Intersect is not always 'inner join'
- it gives common records from both tables.
- It is 'inner join' where on clause with all columns from left & right table. with = op.
  - values based on condition in on clause or 2 using clause

- Inner join: only rows that match the 1 condition for 2 key columns.
- Left join: inner join + non matching rows (based on 1 key column & given condition) from left table with null values in right table.
- Right join: inner join + non matching rows (based on 1 given key columns & 2 condition) from right table with null values in left table.
- Full outer join:
  - left join
  - union
  - right join
- Self join: left & right table are same
  - inner
  - left
  - right
  - full outer
  - cross

# Joins DBMS1 Day4 Sum up

- Joins
  - what? => to merge data from various tables
    - ⇓
    - to form single data source [from clause]
  - why? => Complex insights
  - Classifications => Inner, left, right, full outer join, etc
- Aliases for
  - table
  - column
  - select query
  - } to simplify the representation in syntax, etc.
- non standard join => Select * from T1, T2;
- parent-child relationship.
  - when same table used for multiple times.
  - 1st act as parent then 2nd child & soon...
  - for : a). Hierarchical data
    - b) R/w two records of same table (e.g. secondary & 1° A/c)
- Multiple columns use in join
  - ↳ those key columns ⇒ called as composite join key
- multiple tables also joined
  - no limit on no. of tables can join.
  - no. of conditions ≥ no. of tables - 1
  - not preferred duplicate column values for key joining columns. else => cross distribution rows.
- Natural join
  - Implicit join on common columns.
  - is a equi-join only.
  - restrictions
    - not preferred duplicate column values for common columns ⇒ else => cross distribution rows
    - not handle null values in common columns.
  - developer no need to know all common columns.
  - use for
    - ↳ when need all common column join.

- Other equi join
  - also called "inner join"
  - Explicit mention of key columns, unlike natural join.
  - handle 1 null values & 2 comparison (as explicit)
  - = operator & explicit mention in on clause
- non-equi join
  - non-equi op. like >, >=, <, etc used.
  - to create different dimensional report.
- self-join
  - join with same table.
  - any no. of times can be joined.
  - to get meaningful data. (eg. hierarchical)
- joins
  - inner join
    - default join
    - rows/records from matching 1 given condition in 2 given key columns.
    - also use the using clause, if column names same in both tables (instead of on clause)
  - left join / left outer join = inner join (i.e. matching) + valid record in left table with null values in right table columns
  - right join / right outer join = inner join (i.e. matching) + valid record in right table with null values in left table columns
  - full join / full outer join = left join union right join.
  - Cross join = cartesian product of two tables.