GROUP BY:

- SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

- The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

- The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column1, column2 FROM table_name WHERE [conditions]

GROUP BY column1, column2 ORDER BY column1, column2; **Ex:**

select name, sum(salary) from customers group by name;

Aggregate Functions :-

Aggregate functions are Functions that take a collection (a set (or) multiset) of values as input and Return a single value.

SQL offers five standard built-in-aggregate Functions.

* Average : Avg.
* minimum : min
* maximum : max
* Total : sum
* Count : count.

The Input to sum and avg must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings.

Basic aggregation :-

Consider the query "Find the average salary of Instructors in the computer science department". we write this query as

```
Select avg (salary)
from Instructor
wher dept-name = 'comp.sci";
```

The Result of this query is a Relation with a single attribute containining a single tuple with a numerical value corresponding to the average salary of Instructors in the comp. science department.

ORDER BY:

- SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns.

- Some database sorts query results in ascending order by default.

Syntax:

- The basic syntax of ORDER BY clause is as follows:

SELECT
column-list
FROM
table_name

[WHERE
condition]

[ORDER BY column1, column2, .. columnN] [ASC |
DESC]; **Ex:**

1. select * from customers order by name, salary;
2. select * from customers order by name desc;

Aggregate Functions:

- SQL aggregate functions return a single value, calculated from values in a column.
- Useful aggregate functions:
  - **AVG()** - Returns the average value
  - **COUNT()** - Returns the number of rows
  - **MAX()** - Returns the largest value
  - **MIN()** - Returns the smallest value
  - **SUM()** - Returns the sum

AVG () Function

The AVG () function returns the average value of a numeric column.

AVG () Syntax

SELECT AVG (column_name) FROM table_name; Ex:

SELECT AVG (Price) FROM Products;

COUNT () Function

COUNT aggregate function is used to count the number of rows in a database table.

COUNT () Syntax:

SELECT COUNT (column_name) FROM table_name; Ex:

SELECT COUNT (Price) FROM Products;

MAX () Function

The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.

MAX () Syntax:

SELECT MAX (column_name) FROM table_name; EX:

SELECT MAX (SALARY) FROM EMP;

SQL MIN Function:

SQL MIN function is used to find out the record with minimum value among a record set.

MIN () Syntax:

SELECT MIN (column_name) FROM table_name; EX:

SELECT MIN (SALARY) FROM EMP;

SQL SUM Function SQL:

SUM function is used to find out the sum of a field in various records.

SUM () Syntax:

SELECT COUNT (column_name) FROM table_name; EX:

SELECT COUNT (EID) FROM EMP;

SQL Join Types:

- There are different types of joins available in SQL: They are:
  - INNER JOIN
  - OUTER JOIN
  - SELF JOIN
  - CARTESIAN JOIN

INNER JOIN:

- The most frequently used and important of the joins is the **INNER JOIN**. They are also referred to as an EQUIJOIN.

- The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate.

- The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate.

- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax:

- The basic syntax of INNER JOIN is as follows:

SELECT table1.column1, table2.column2... FROM table1 INNER JOIN table2 ON table1.common_filed = table2.common_field;

Ex: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS INNER JOIN

______ORDERS CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

OUTER JOIN:

- The Outer join can be classified into 3 types. They are:

  Left Outer Join\
  Right Outer Join
  Full Outer Join

Left Outer Join:

- The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table.

- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax:

- The basic syntax of **LEFT JOIN** is as follows:

SELECT table1.column1, table2.column2... FROM table1 LEFT JOIN table2 ON table1.common_filed = table2.common_field;

EX: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS

LEFT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

RIGHT JOIN:

- The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table.

- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax:

- The basic syntax of **RIGHT JOIN** is as follows:

SELECT table1.column1, table2.column2... FROM table1 RIGHT JOIN table2 ON table1.common_filed = table2.common_field;

Ex: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS
RIGHT JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

FULL JOIN:

- The SQL **FULL JOIN** combines the results of both left and right outer joins.

- The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Syntax:

- The basic syntax of **FULL JOIN** is as follows:

SELECT table1.column1, table2.column2... FROM table1 FULL JOIN table2 ON table1.common_filed = table2.common_field;

Ex: SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS FULL JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

SELF JOIN:

- The SQL **SELF JOIN** is used to join a table to it as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax:

- The basic syntax of **SELF JOIN** is as follows:

SELECT a.column_name, b.column_name...FROM table1 a, table1 b WHERE a.common_filed = b.common_field;

Ex:

SELECT a.ID, b.NAME, a.SALARY FROM CUSTOMERS a, CUSTOMERS b WHERE a.SALARY < b.SALARY;

CARTESIAN JOIN:

- The CARTESIAN JOIN or CROSS JOIN returns the cartesian product of the sets of records from the two or more joined tables.

- Thus, it equates to an inner join where the join-condition always evaluates to True or where the join-condition is absent from the statement.

Syntax:

- The basic syntax of **CROSS JOIN** is as follows:

SELECT table1.column1, table2.column2... FROM table1, table2 [, table3]; **Ex:** SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS, ORDERS;

Normal Forms based on F.D:-

First normal Form:-

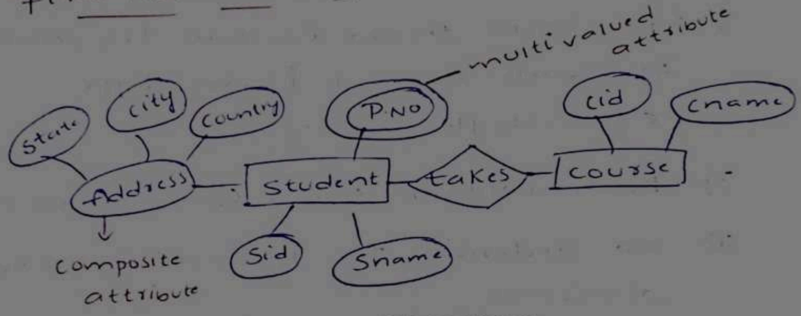


fig:- ER Diagram

(Student)

Sid	Sname	S.address	P.NO.
1	A	Haryana India	P_1, P_2
2	B	Flourida US	P_3
3	C	ABC Canada	P_4, P_5
4	D	SAP	P_6
5	E	Guntur	P_7

S.address → composite attributes; P.NO. → multivalued attribute

→ This is not in 1NF. because It contains multivalued attributes & composite attributes

Tabel / Relation will be in first normal form (1NF):-

1) If all the attributes of the table contain only atomic values / atomic domains

atomic : that cannot be further decomposed into smaller pieces

Converting in to 1NF.

Sid	Sname	State	County	Phone No
1	A	Haryana	India	P_1
1	A	Haryana	India	P_2
2	B	Flourida	US	P_3
3	C	ABC	Canada	P_4
3	C	ABC	Canada	P_5
4	–	:	:	:
5	–			

→ Relation in 1NF.

Here, each attribute having only one value.

2) A column should contain the values of same domain / same type. (eg:- Int, float, char)

3) Each column should have unique name.

4) No ordering to Rows and columns applicable.

5) No duplicate Rows are allowed.

5) No duplicate Rows are allowed.

* when E·R diagram is converted into Relational schema then differently that schema must be in 1NF by default.

another way of Converting in 1NF.

P.K

Sid	Sname	State	country
1	A	HR	IN
2	B	HR	IN US
3	C	FI	US

F.K

Sid	P.No
1	P_1
1	P_2
2	P_3
3	P_4
3	P_5

(Base table)

(1 - many Relationship)

These tables are in 1NF.

Second normal Form (2NF):- A Relation will be second normal form (2NF) if it follows the following condition:

1) The table (or) Relation should be in 1NF

2) No Partial Dependency present in the Relation /table.

P.D ⇒ $\boxed{\text{proper subset of CK} \rightarrow \text{non-prime attribute.}}$

3) All the non-prime attributes should be fully functionally dependent on the Candidate Key.

examples:- R (A, B, C, D, E, F)

$$F.D = \{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow E\}$$

1) Relation is in 1NF.

2) $A\cancel{BCDE}F^{+} = \{A, B, C, D, E, F\}$

$AF^{+} = \{A, B, C, D, E, F\}$

(not S.K) $A^{+} = A, B, C, D, E$

〃 $F^{+} = F$ (trivial F.D)

P.A = A, F

Non prime Attributes = B, C, D, E

* → If prime attributes are present on the Right Hand Side of F.D then there are more C.K.

∵ A → B (non P.A) . (Partial dependency)

(2) R (A, B, C, D)

F.D = {AB → CD, C → A, D → B}.

$(AB\cancel{CD})^+ = \{ABCD\}$

S.K → AB^+ = A, B, C, D

$A^+ = A$

$B^+ = B$

Candidate Key = AB

P.A = A, B.

Here, more C.K are present.

CK = AB → CB, AD

$B^+ = B$, $C^+ = CA$

$A^+ = A$, $D^+ = D, B$

Candidate Keys = AB, CB, DA.

P.A = A, B, C, D.

Non P.A = 0

∵ All attributes of Relation are P.A definetly that Relation is in 2NF.

(3) R = {A, B, C, D}

F.D = {A → B, B → C, C → D}

$A\cancel{BCD}^+ = \{ABCD\}$

SK → $A^+ = \{A, B, C, D\}$

C.K.

candidate Key = A

P.A = A

N.P.A = B, C, D.

∵ No P.D, this Relation is in 2NF.

If C.K Having only single attribute then that Relation would be in 2NF

Third Normal Form:- (3NF)

The Relation will be in 3NF if It follows the following conditions.

1) It is in 2NF

2) No "transitive dependency." for non prime attributes

transitive A → B and B → C

transitive dependency:- $A \to B$ and $B \to C$

$$\Downarrow$$

$$A \to C$$

(N.P.A) (N.P.A)

$$\boxed{NPA \to NPA.}$$

(or)

A table is in 3NF, if and only if for each of its non trivial functional dependency atleast one of the following Condition Holds:-

① L.H.S is SK.

② RHS is prime attributes.

Examples: ① R (A, B, C, D)

F.D = (A→B, B→C, C→D)

SK → ~~ABCD~~⁺ = {ABCD}

CK ← A^+ = {ABCD}

Candidate Key = A

Prime attributes = A

Non prime attributes = B, C, D.

∵ A → B (P.A → NPA) ✓

B → C [N.P.A → N.P.A] x

C → D [N.P.A → N.P.A] x

∵ Hence, the above Relation is not in 3NF.

② R (A, B, C, D, E, F)

F.D = {AB → CDEF, BD → F}

AB ~~CDEF~~⁺ = {ABCDEF}

SK → AB^+ = {ABCDEF}

A^+ = {A} x

B^+ = {B} x

Candidate Key = AB.

P.A = A, B

N.P.A = C, D, E, F.

∵ AB ⇒ CDEF (P.A → NPA) ✓

BD → F (N.P.A → N.P.A) (x)

(P.A)(N.P.A)

∨

(N.P.A)

∵ Hence, the above Relation is not in 3NF.

S.No.	BCNF	4NF
1	A relation in BCNF must also be in 3NF.	A relation in 4NF must also be in Boyce Codd Normal Form (BCNF).
2	A relation in BCNF may have multi-valued dependency.	<A relation in 4NF must not have any multi-valued dependency.
3	A relation in BCNF may or may not be in 4NF.	A relation in 4NF is always in BCNF.
4	BCNF is less stronger in comparison to 4NF.	4NF is more stronger in comparison to BCNF.
5	If a relation is in BCNF then it will have more redundancy as compared to 4NF.	If a relation is in 4NF then it will have less redundancy as compared to BCNF .

6	If a relation is in BCNF then all redundancy based on functional dependency has been removed.	If a relation is in 4NF then all redundancy based on functional dependency as well as multi-valued dependency has been removed.
7	For a relation, number of tables in BCNF is less than or equal to number of tables in 4NF.	For a relation, number of tables in 4NF is greater than or equal to number of tables in BCNF.
8	Dependency preserving is hard to achieve in BCNF.	Dependency preserving is more hard to achieve in 4NF as compared to BCNF.
9	In real world database designing, generally 3NF or BCNF is preferred.	In real world database designing, generally 4NF is not preferred by database designer.

9	In real world database designing, generally 3NF or BCNF is preferred.	In real world database designing, generally 4NF is not preferred by database designer.
10	A relation in BCNF may contain multi-valued as well as join dependency.	A relation in 4NF may only contain join dependency.

Purpose of Normalization:-(or) Schema Refinement:-

→ Normalization Helps to Reduce Redundancy and complexity by examining new data types used in the table.

→ It is Helpful to divide the large database table into smaller tables and link them using Relationship.

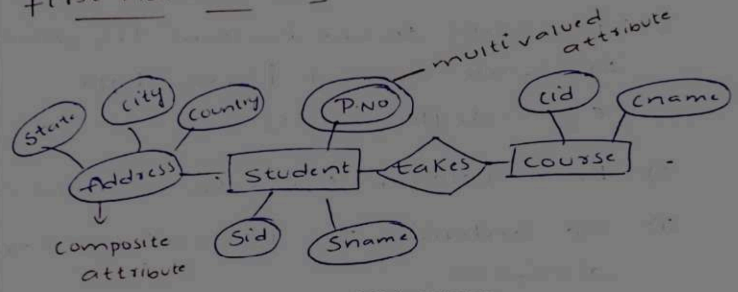
→ It avoids duplicate data (or) no Repeating groups into a table.

→ It Reduces the chances for anomalies to occur in a database.

Anomalies:- Anomalies make the table

Normal Forms based on F.D:-

First normal Form:-



(Student) fig:- ER Diagram

Sid	Sname	S.address	P.NO.
1	A	Haryana India	P_1, P_2
2	B	Flourida US	P_3
3	C	ABC Canada	P_4, P_5
4	D	S.AP	P_6
5	E	guntur	P_7

→ This is not in 1NF because It contains multivalue attributes & composite attributes

composite attributes — multivalued attribute

Scanned with CamScanner

Tabel / Relation will be in first normal form (1NF) :-

1) If all the attributes of the table contain only atomic values / atomic domains

atomic : that cannot be further decomposed into smaller pieces.

2) converting in to 1NF.

Sid	Sname	State	county	Phone No
1	A	Haryana	India	P_1
1	A	Haryana	India	P_2
2	B	Flourida	US	P_3
3	C	ABC	canada	P_4
3	C	ABC	Canada	P_5
4	–	=	=	=
5	–			

→ Relation in 1NF.

Here, each attribute Having only one value.

2) A column should contain the values of same domain / same type.
(eg:- Int, float, char)

3) Each column should have unique name.

4) No ordering to Rows and columns applicable.

5) No duplicate Rows are allowed.

* when E.R diagram is converted into Relational schema then diffinetly that schema must be in 1NF by default

another way of converting in 1NF.

P.K   F.K

Sid	Sname	State	country
1	A	HR	IN
2	B	HR	IN US
3	C	FI	US

Sid	D.No
1	P_1
1	P_2
2	P_3
3	P_4
3	P_5

(Base table)

(1 - many Relationship)

These tables are in 1NF.

Second normal Form (2NF):- A Relation will be second normal form (2NF) if it follows the following condition:

1) The table (or) Relation should be in 1NF

2) No Partial Dependency present in the Relation Itable.

P.D $\Rightarrow$ $\boxed{\text{proper subset of CK} \rightarrow \text{non-prime attribute.}}$

3) All the non-prime attributes should be fully functionally dependent on the Candidate Key.

examples:- R (A, B, C, D, E, F)

$$F.D = \{A \rightarrow B,\ B \rightarrow C,\ C \rightarrow D,\ D \rightarrow E\}$$

1) Relation is in 1NF.

2) $A\cancel{BCDE}F^+ = \{A, B, C, D, E, F\}$

$AF^+ = \{A, B, C, D, E, F\}$

(not S.K) $A^+ = A, B, C, D, E$

" $F^+ = F$ (trivial F.D)

Scanned with CamScanner

C.K = AF

P.A = A, F

Non prime Attributes = B, C, D, E

* → If prime attributes are present on the Right Hand Side of F.D then there are more C.K.

$\because$ $A \rightarrow B$ (non P.A) (Partial dependency)

Third Normal Form:- (3NF)

The Relation will be in 3NF if It follows the following conditions.

1) It is in 2NF
2) No "transitive dependency." for non prime attributes

transitive dependency:- A→B and B→c

⇓

A→c

(N.P.A) (N.P.A)

$$\boxed{NPA \rightarrow NPA.}$$

(or)

A table is in 3NF, if and only if for each of its non trivial functional dependency atleast one of the following Condition Holds:-

1. L.H.S is SK.
2. RHS is prime attributes.

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- **Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.
- **Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.
- **Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.
- **Isolation** − In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

CONCURRENCY CONTROL WITH LOCKING METHODS

A **lock** guarantees exclusive use of a data item to a current transaction. In other words, transaction T2 does not have access to a data item that is currently being used by transaction T1. A transaction acquires a lock prior to data access; the lock is released (unlocked) when the transaction is completed so that another transaction can lock the data item for its exclusive use.

Most multiuser DBMSs automatically initiate and enforce locking procedures. All lock information is managed by a **lock manager**.

Lock Granularity

Indicates the level of lock use. Locking can take place at the following levels: database, table, page, row or even field.

LOCK TYPES

Regardless of the level of locking, the DBMS may use different lock types:

1. Binary Locks

Have only two states: locked (1) or unlocked (0).

2. Shared/Exclusive Locks

An **exclusive lock** exists when access is reserved specifically for the transaction that locked the object .The exclusive lock must be used when the potential for conflict exists. A **shared lock** exists when concurrent transactions are granted read access on the basis of common lock. A shared lock produces no conflict as long as all the concurrent transactions are read only.

DEADLOCKS

A deadlock occurs when two transactions wait indefinitely for each other to unlock data.

The three basic techniques to control deadlocks are:

- Deadlock preventation . A transaction requesting a new lock is aborted when there is the possibility that a deadlock can occur. if the transaction is aborted , all changes made by this transaction are rolled back and all locks obtained by the transaction are released .The transaction is then rescheduled for execution.
- Deadlock detection. The DBMS periodically tests the database for deadlocks. if a deadlock is found one of the transactions is aborted (rolled back and restarted) and the other transaction are continues.
- Deadlock avoidance. The transaction must obtain all of the locks it needs before it can be executed .This technique avoids the rollback of conflicting transactions by requiring that locks be obtained in succession

What is Two-Phase Locking (2PL)?

- Two-Phase Locking (2PL) is a concurrency control method which divides the execution phase of a transaction into three parts.
- It ensures conflict serializable schedules.
- If read and write operations introduce the first unlock operation in the transaction, then it is said to be Two-Phase Locking Protocol.

This protocol can be divided into two phases,
1. In Growing Phase, a transaction obtains locks, but may not release any lock.
2. In Shrinking Phase, a transaction may release locks, but may not obtain any lock.

- Two-Phase Locking does not ensure freedom from deadlocks.

Types of Two – Phase Locking Protocol

Following are the types of two – phase locking protocol:

1. Strict Two – Phase Locking Protocol
2. Rigorous Two – Phase Locking Protocol
3. Conservative Two – Phase Locking Protocol

1. Strict Two-Phase Locking Protocol

- Strict Two-Phase Locking Protocol avoids cascaded rollbacks.
- This protocol not only requires two-phase locking but also all exclusive-locks should be held until the transaction commits or aborts.
- It is not deadlock free.
- It ensures that if data is being modified by one transaction, then other transaction cannot read it until first transaction commits.
- Most of the database systems implement rigorous two – phase locking protocol.

2. Rigorous Two-Phase Locking

- Rigorous Two – Phase Locking Protocol avoids cascading rollbacks.
- This protocol requires that all the share and exclusive locks to be held until the transaction commits.

3. Conservative Two-Phase Locking Protocol

- Conservative Two – Phase Locking Protocol is also called as Static Two – Phase Locking Protocol.
- This protocol is almost free from deadlocks as all required items are listed in advanced.
- It requires locking of all data items to access before the transaction starts.