① Normalization (Process of normalizing dimension tables is called ...)
② Query Complexity (Snowflake schema is more complex) (Enter)

Use Star Schema → ① In data marts ② (subset of central data warehouse)
Use Snowflake Schema → ① In data warehouses (Saves a lot of space)

SQL

Structured Query language (SQL) → Read, Write, Update → DATA

→ Database management
→ Data manipulation

Used to insert, update, modify & query data.

- Relational DBMS → MySQL, SQLite, Micro SQL Server, IBM DB2 Oracle, PostgreSQL etc.

- Measures → Stored in fact tables. They are basically values.
- Dimensions → what these measures actually mean. Stored in dimension tables.

Think. before you do.
Understand the data.

- Data with meaning → Information
- Meta data is the data that describes data.

Database : A container to store organized data.
Table : Structured list of data.
Column : A single field in table
Row : A record in a table.

- Data Modelling : Organizes and structures info into multiple related tables.
  Represents a business process.
  Should always represent real world.

- NoSQL → A mechanism for storage and retrieval of unstructured data modeled by means other than tabular relations in relational databases.

SQL offers flexibility by supporting distributed databases i.e. databases that can be run on several computer networks at a time.

Relational → Easy querying
Nosql → Easy data manipulation
Transactional → Operational database

(Adjective)

① Entity ② Attribute ③ Relationship → One to one, Many to many, One to many

Entity: Text (Singular Noun)
Attribute: Text
Relationship: Text → Type of relationship

ER diagrams → (Entity Relationship Diagrams)

Cardinality:

Primary key: A column whose values uniquely identify every row in a table.
(eg) Order ID. Employee ID.

Foreign key: One or more columns that can be used together to identify a single row in another table.

- Chen Notation: (1:m) → One to Many; (m:N) → Many to many; (1:1) → One to one
- Crow's foot Notation: (1:m) → 1 … m; (m:N) →; (1:1) →
- UML Class Diagram: (1:m) → 1..1, 1..*; (m:N) → 1..*, 1..*; (1:1) → 1..1, 1..1

- Retrieving data with SELECT statement : Most fundamental

SELECT statement : What you want and where you want to select it from.

Columns

```
SELECT   prod-name
FROM     Products;
         Tables
```

Syntax

If selecting multiple columns use commas (,).
If Requesting all columns use asterik (*).

If we want only a sample of data out of a big data set then use LIMIT.

```
SELECT   columns you wish to see
FROM     specific table
LIMIT    number of records
```

SQLite

```
SELECT ____
FROM ____
LIMIT 5;
```

Oracle

```
"
"
OR
WHERE ROWNUM <=5;
```

DB2

```
"
"
FETCH FIRST 5 ROWS ONLY;
```

- Delete rows from a table →

```
DELETE FROM tablename WHERE column name = value ;
```

UPDATE ____ SET ____ = ____ WHERE ____ = ____;

table name — column name — value

Date / Page No.

- Creating Tables :

CREATE TABLE Name of table (Shoes)

Id	char (10)	PRIMARY KEY,
Brand	char (10)	NOT NULL,
Type	char (250)	NOT NULL,
Color	char (256)	NOT NULL,
Price	decimal (8,2)	NOT NULL,
Desc	Varchar (750)	NULL

);

String length in bytes and not no. of characters.

Primary key must have a value.

8 significant digits are being stored out of which 2 are in decimal places. -999999.99 to 999999.99

- DROP
  - ↳ To drop tables

(DROP TABLE ____ ;)

↓ Table name

Data of Tables is not lost

Adding Data to Table :

- TRUNCATE
  - ↳ Data of tables is lost

(TRUNCATE TABLE ____ ;)

↳ Table name

INSERT INTO Name of table Shoes

VALUES (' ', ' ', ...);

- ALTER
  - ↳ Modification is allowed

(ALTER TABLE ____ RENAME TO ____)

↳ Table name ↳ New table name

- Creating Temporary Tables :

Table name

CREATE TEMPORARY TABLE Sandals AS

(

SELECT *

FROM shoes

WHERE shoe_type = 'sandals'

);

- GRANT ALL PRIVILEGES ON
- REVOKE

Adding Comments :

(i) Single Line

```
SELECT shoe_id
  , brand_id
  , shoe_name
FROM shoes
```

Section

```
SELECT shoe_id
/*, brand_id
, shoe_name
*/
FROM shoes
```

→ Filtering, Sorting and Calculating Data.

WHERE	NOT
BETWEEN	LIKE
IN	ORDER BY
OR	GROUP BY

Wildcards → Precise search.

→ DELETE
DELETE FROM ___ WHERE ___

→ GRANT
Authorize users to modify
GRANT ALL PRIVILEGES ON ___

→ REVOKE
Just opposite

It's used to define the rows in which the search will be carried

FILTERING

```
SELECT column_name, column_name
FROM table_name
WHERE column_name operator value;
```

→ Use of operators

<> → Not equal to
BETWEEN → Between an inclusive range.

↳ BETWEEN and AND are used together.

** The DBMS will not evaluate the second condition in a WHERE clause if the first condition is met.

NULL → No data in the column.

* IS NULL → To check whether there is no data or not.

{ =, <>, >, <, >=, <=, BETWEEN, IS NULL, AND }

Advanced Filtering

IN operator
- → Specify a range of cond^n's
- → Semicolon terminated.
- → Enclosed in ()

OR operator
- → Semicolon terminated

⇒ IN executes faster than OR.
In IN we can contain another SELECT for sub queries.

OR with **AND**

Use of ().

```
SELECT ____
FROM ____
WHERE (____ OR ____)
AND ____ ;
```

If this is not there then SQL processes OR before AND statement.

- Order of Operations → SQL processes AND before OR.
  - Use ( ).

NOT Operator → To exclude diff options.

```
SELECT *
FROM ____
WHERE NOT ____ AND NOT ____;
```

Using Wildcards

- Special characters used to match parts of a value.
- Uses LIKE as an operator
- Search pattern made from literal text, wildcard char, or a combination.

(LIKE) → Used with STRINGS
CANNOT be used for non-text datatypes

→ %Pizza = Grabs anything ending with pizza
Pizza% = " " after the word pizza

%Pizza% = Grabs anything before and after the word pizza

S%E = Grabs anything that starts with S and ends with E.

t%@gmail.com → Grabs gmail addresses that start with t.

→ % wildcards will not match NULLs

→ Underscore (_)

Matches a single character.

Not supported by DB2.

WHERE size LIKE '_pizza'

Output : spizza
mpizza

→ Bracket ([])

Used to specify a set of chars in a specific location.

Does not work with SQLite.

Downsides

→ Takes longer to run
→ Better to use operator
→ Placement of wildcards is imp.

SORTING

SELECT _____
FROM _____
ORDER BY _____.

To sort the results

1 → allows user to sort data by particular columns.

- → Multiple column names
- → Add comma
- → Can sort a column not retrieved
- → last clause

Sort direction

DESC → descending order
ASC → Ascending order

Specify each column what to do before

ORDER BY 2, 3 → 3rd column.
2nd column

→ +, −, /, *

SELECT
_____
, _____
, _____
, _____ * _____ AS _____
FROM _____

Date
Page No.

Order of Operations :

1) Parentheses
2) Exponents
3) Multiply
4) Div
5) Addn
6) Subn

PEMDAS

Aggregate Functions

1) AVG()
2) COUNT()
3) MIN()
4) MAX()
5) SUM()

AVG()

Rows containing NULL values are ignored.

```
SELECT AVG(____) AS ____
FROM ____
```
(alias)

COUNT(*)

Counts NULL

```
SELECT COUNT(*) AS ____
FROM ____
```

COUNT(column) → Ignores NULL

```
SELECT COUNT(____) AS ____
FROM ____
```

→ NULL is ignored.

MAX MIN

SELECT MAX (____) AS _____
FROM _____

SELECT MAX (____) AS _____
, MIN (____) AS _____
FROM _____

SUM

SELECT SUM (____) AS _____
FROM _____

SELECT SUM (____ * ____) AS ____
FROM _____
WHERE _____ ;

DISTINCT

↳ → If it is not specified, ALL is assumed
→ Cannot use DISTINCT on COUNT(*)
→ No value to use with MIN and MAX function.

SELECT COUNT(DISTINCT ____)
FROM _____

- GROUPING DATA

GROUP BY

```
SELECT
  Region
, COUNT(____) AS ____
FROM ____
GROUP BY Region;
```

- → multiple columns
- → Listed columns only works
- → NULLS will be grouped too

WHERE does not work for groups.
filters on rows.

Instead use **HAVING** clause.

```
SELECT
____
, COUNT(____)AS ____
FROM ____
GROUP BY ____
HAVING COUNT(____) ____;
```

Total no. of orders by customer

```
SELECT CustomerId
, COUNT(*) as orders
FROM Invoices
GROUP BY CustomerId
Having COUNT(*) >= 0;
```

WHERE filters before data is group
while Having after.

ORDER BY sorts data.

```
SELECT _____
, COUNT(____) AS _____

FROM _____
WHERE _____
GROUP BY _____
HAVING COUNT(____) _____;
 ORDER BY _____;
```

→ SQLite is a software library that implements a (1) self-contained, (2) serverless, (3) zero-config, (4) transactional SQL database engine.

• SUBQUERIES → Queries embedded into another
RDs store data in multiple tables
They merge data from multiple source
Helps with filtering

```
SELECT
CustomerID
, CompanyName
, Region
FROM Customers
WHERE customerID IN (SELECT
customerID
    FROM Orders
    WHERE Freight > 100);
```

executed first

[www.postgresql.com]

Date
Page No.

A single column at a time.
* Indendation is imp.

Benefits of RDBMS :

Joining Tables :

→ Efficient storage
→ Easier manipulation
→ Greater scalability
→ Logically models a process
→ Tables are related through common values (Keys)

Joins → • Associate correct records from each table on the fly.

• Allows data retrieval from multiple tables in one query

• Joins are not physical - they persist for the duration of the query execution.

* Cartesian (Cross) Joins :

Each row from the first table joins with all the rows of another table.

Computationally taxing

```
SELECT _____
, _____
FROM _____ CROSS JOIN _____ ;
```

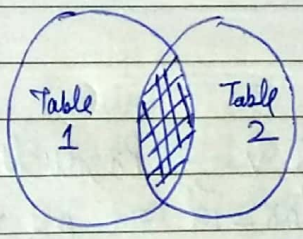
Table 1 — Supplier

Table 2: Products

Output will be the no. of joins in the 1st table multiplied by the no. of rows in the 2nd table.

Drawbacks
- → Not frequently used
- → Computationally Taxing
- → Will return products with the incorrect vendor

* Inner Joins:



INNER JOIN keyword selects records that are matching in both!

SELECT _____

→ _____

FROM Suppliers INNER JOIN Products

* (ON) Suppliers.supplierid = Products.supplierid

Pre-qualifying names

SELECT o.OrderID, c.CompanyName, e.LastName

Orders placed & Customers who placed them → FROM ((Orders o INNER JOIN Customers c ON o.CustomerID = c.CustomerID)

Employees associated with above → INNER JOIN Employees e ON o.EmployeeID = e.EmployeeID);

- **Aliases and Self Joins :**

SQL aliases give a table or a column a temporary name.

```
SELECT column-name
FROM table_name AS alias-name
```

```
SELECT ____
FROM ____
WHERE Vendors.vendor_id = Products.vendor_id;
```

Using Alias

```
SELECT ____
FROM Vendors AS v, Products AS p
WHERE v.vendor_id = p.vendor_id;
```

SELF Joins

Join the original table to itself.

```
SELECT column_name(s)
FROM table1 T1, table2 T2
WHERE condition
```

Customers that are from the same city.

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID = B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

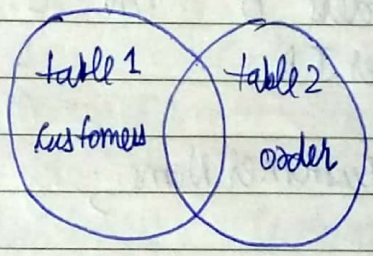
Full

Left, Right and Outer Joins :

- SQLite only does Left Joins.

- **Left Join** Returns all records from the left table and the matched records from the right table.

The result is NULL from the right side, if there is no match.



- **Right Join** Returns all records from the right table and the matched records from the left table.

The result is NULL from the left side, when there is no match.

⇒ The table you list first is acted upon by the type of join you use.

- **Full Outer Join** All records when there is a match in either left or right table records.

- **Left Join**

Will select all customers, and any orders they might have.

```
SELECT C.CustomerName, O.OrderID

FROM Customers C

LEFT JOIN Orders O ON C.CustomerID
= O.CustomerID

ORDER BY C.CustomerName;
```

- **Right Join**

It will return all employees, and any orders they might have placed.

```
SELECT Orders.OrderID,
Employees.LastName,
Employees.FirstName

FROM Orders

RIGHT JOIN Employees ON
Orders.EmployeeID = Employees.EmployeeID

ORDER BY Orders.OrderID;
```

- **Full Outer Join:**

It will return all customers, and all orders.

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON
Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Secret Weapon*

Unions :

UNION operator is used to combine the result set of 2 or more SELECT statements.

Each SELECT statement within UNION must have the same no. of columns.

columns must have similar datatypes

Columns in same order.

→ SELECT column-name(s) FROM table 1

UNION

SELECT column-name(s) FROM table 2;

Which German Cities have suppliers

→ SELECT city, Country FROM Customers
WHERE Country = 'Germany'

UNION

SELECT City, Country FROM Suppliers
WHERE Country = 'Germany'
ORDER BY City;

TEXT STRINGS

1) Retrieve the data in the format you want
2) Support Joins
3) String functions → Concatenate
   → Substring
   → Trim
   → Lower
   → Upper

→ Concatenations (SQL never supports '+' instead of '||')

```
Select Company Name,
Contact Name,
Company Name || '(' || Contact Name || ')'
From Customers
```

→ Trimming → Trims the leading or trailing space from a string.

TRIM LTRIM RTRIM

```
Select TRIM ("_You the best._") As Trimmed String.
```

→ Substring → Returns the specified no. of characters from a particular position of a given string.

SUBSTR (string name, string position, no. of characters to be returned);

→ Upper and Lower

→ Select UPPER (column name) FROM table name ;

Select LOWER (") FROM " ;

→ Select UCASE (") FROM " ;

Same shit diff name!!

• DATE AND TIME

Dates are stored as datatypes.

Date → YYYY - MM - DD

Date Time → YYYY - MM - DD HH : MI : SS

Timer Stamp → YYYY - MM - DD HH : MI : SS

SQLite → DATE (time string, modifier, modifier, ...)
TIME (")
DATETIME (")
JULIANDAY (")
STRFTIME (format, timestring, modifier, modifier, ...)

→ Modifiers

NNN days
" hours
" minutes
NNN.NNNN seconds
NNN months
NNN years
start of month
start of year
start of day
weekday N
unixepoch
localtime
utc

```
Select column
    STRFTIME ('%Y', column) as
  , "    ('%m', column) as
  , "    ('%d', column) as ....
From table
```

- Compute current Date →

```
Select DATE ('now')
```

- Compute Y, M, D for Current Date →

```
Select STRFTIME ('%Y %m %d', 'now');
```

- ```
  Select STRFTIME ('%H %M %S %s', 'now');
  ```

→ Age

```
, DATE (('now') - column) as ___
From table
```

- CASE STATEMENTS

Mimics if - else - then statement.
Can be used in SELECT, INSERT, UPDATE and DELETE statements.
```

CASE
WHEN C1 THEN E1
WHEN C2 THEN E2
....
ELSE [result else]
END new table name

VIEWS

- A stored query
- Can add or remove columns without changing schema
- Use it to encapsulate queries
- The view will be removed after database connection has ended.

Are Temporary

CREATE [TEMP] VIEW [IF NOT EXISTS]
view name (column name list)

AS

select - statement ;

Select * From my-view

DROP VIEW my-view;

→ Get a count

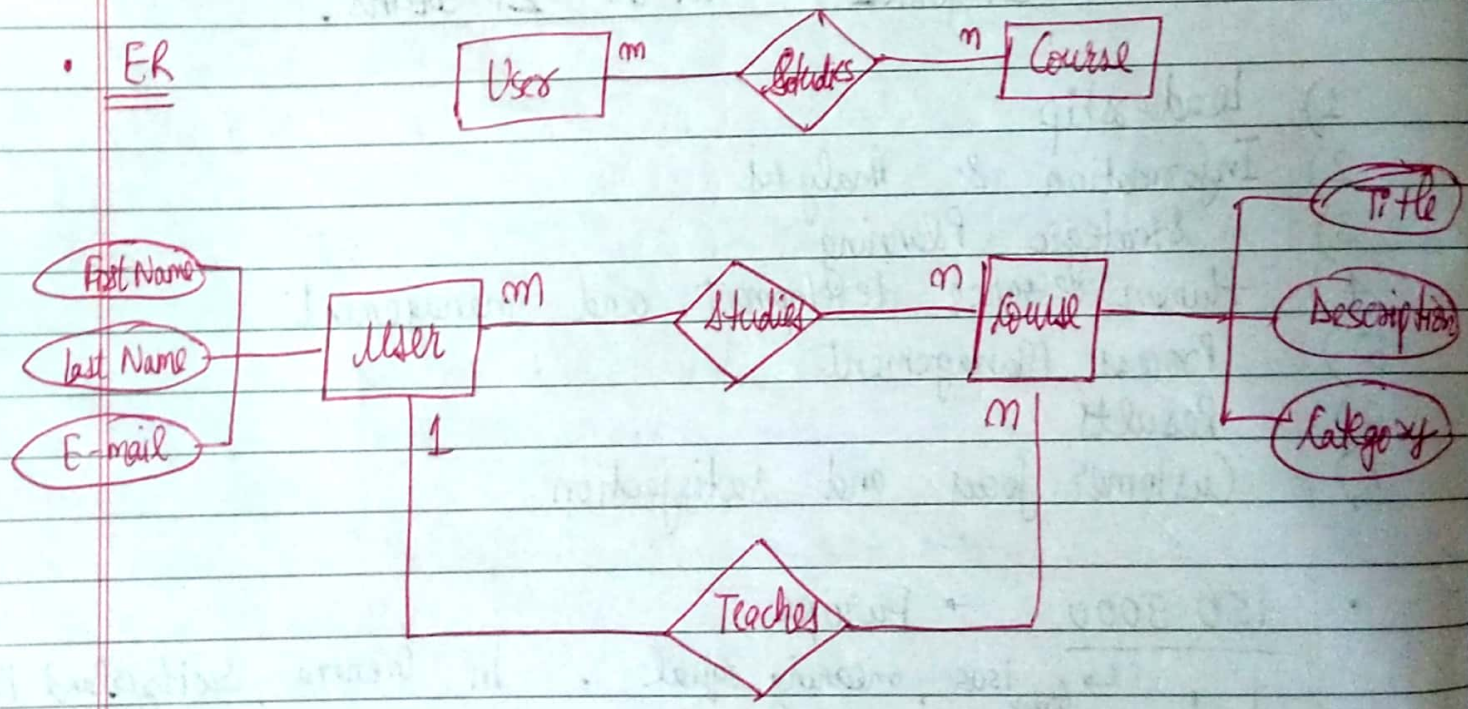
- DATA GOVERNANCE & PROFILING →

Profiling → Examining data for completeness and accuracy.

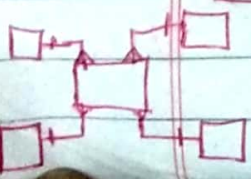
Governance → • Understand your read and write capabilities in different environments.
• Clean up your environments
• Understand your promotion process.

Data Understanding and Business Understanding are Inter-related.

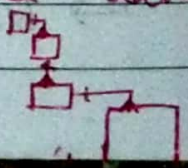
- ER



- Star Schema



Snowflake Schema



Both of them use dimension tables to describe data aggregated in fact tables.