masai

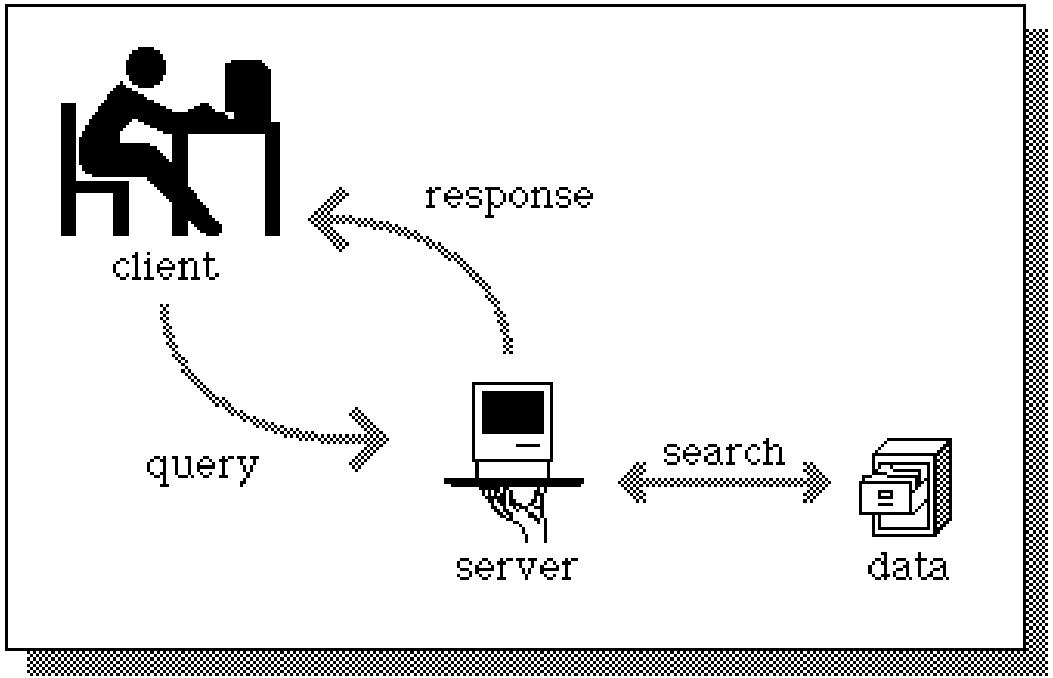
SQL Fundamentals

Part - 1

Topics

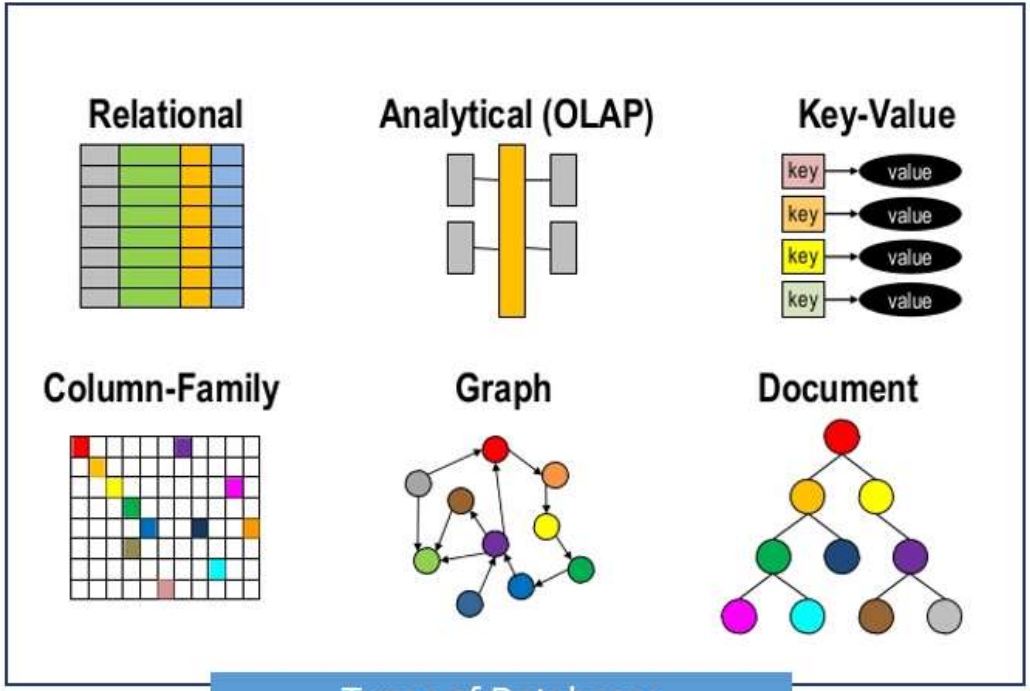
1. Understanding Big Picture
2. Retrieving the data
    a. Select
    b. Distinct
    c. Count
    d. Where
    e. Order By
    f. Limit
    g. Between
    h. In


ZAPPOS.COM
PROCESS.PHP
BACK-END
FRONT-END
HOMEPAGE
CART
REGISTRATION


response
client
query
search
server
data

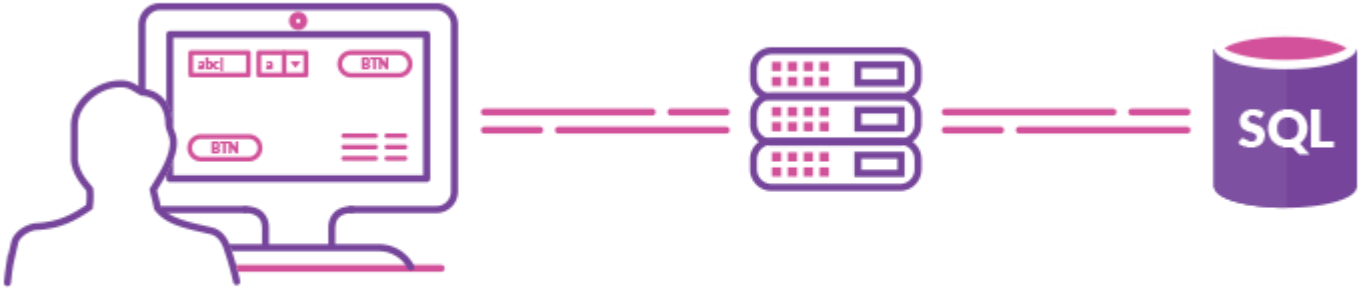
Which data exactly I am talking about ?

Where to store the data ?



Types of Databases

Big Picture



User works with the front-end

The back-end performs business logic and accessses data from the database

Why we need SQL ?

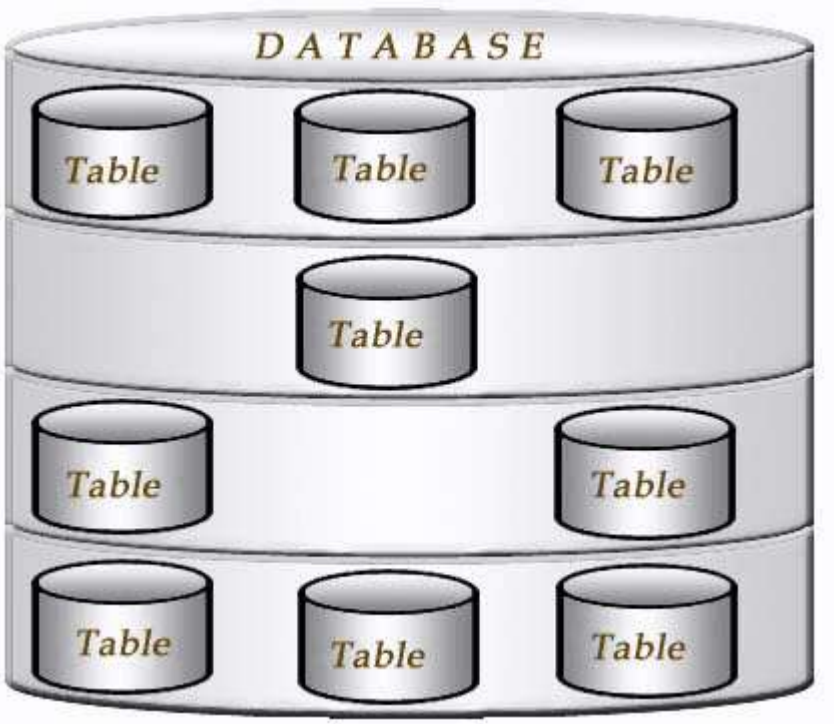
Columns stores a specific data type

Row →
Or record

Emp No	Name	Age	Department	Salary
001	Alex S	26	Store	5000
002	Golith K	32	Marketing	5600
003	Rabin R	31	Marketing	5600
004	Jons	26	Security	5100

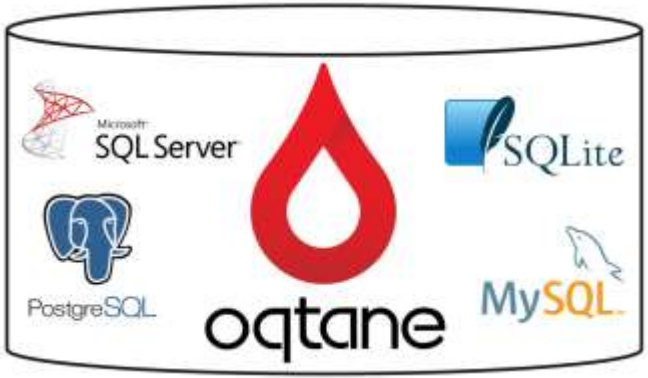
The Numbers in the Brackets give the Maximum Marks in Each Subject.

Student	Subject (Max. Marks)					
	Maths	Chemistry	Physics	Geography	History	Computer Science
	(150)	(130)	(120)	(100)	(60)	(40)
Ayush	90	50	90	60	70	80
Aman	100	80	80	40	80	70
Sajal	90	60	70	70	90	70
Rohit	80	65	80	80	60	60
Muskan	80	65	85	95	50	90
Tanvi	70	75	65	85	40	60
Tarun	65	35	50	77	80	80


DATABASE
Table
Table
Table
Table
Table
Table
Table
Table
Table

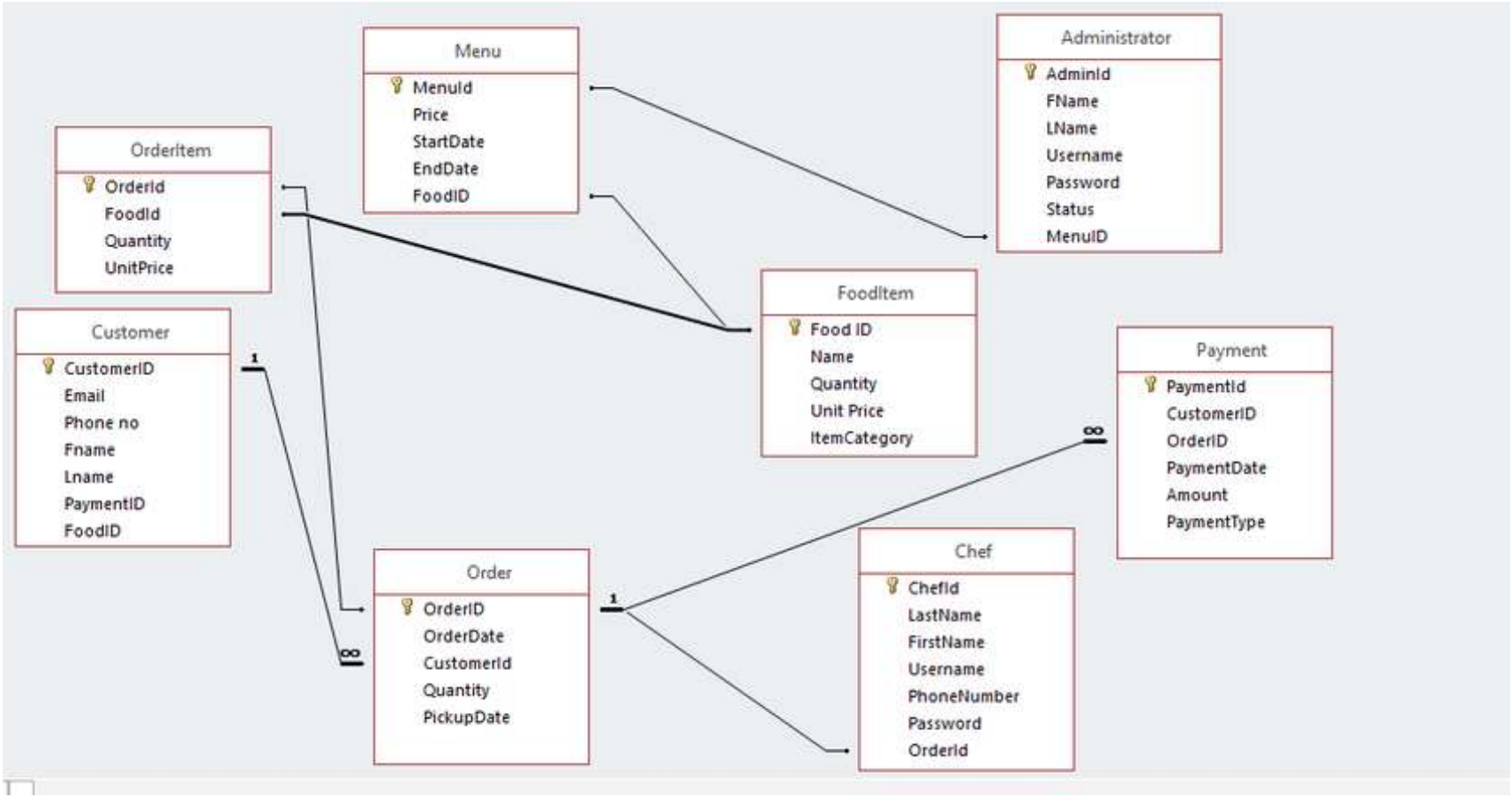
Database Management System

- **Databases** are collections of information — or data — that have been logically modeled.
- The computer programs that interact with databases are called **database management systems**, or **DBMSs**.

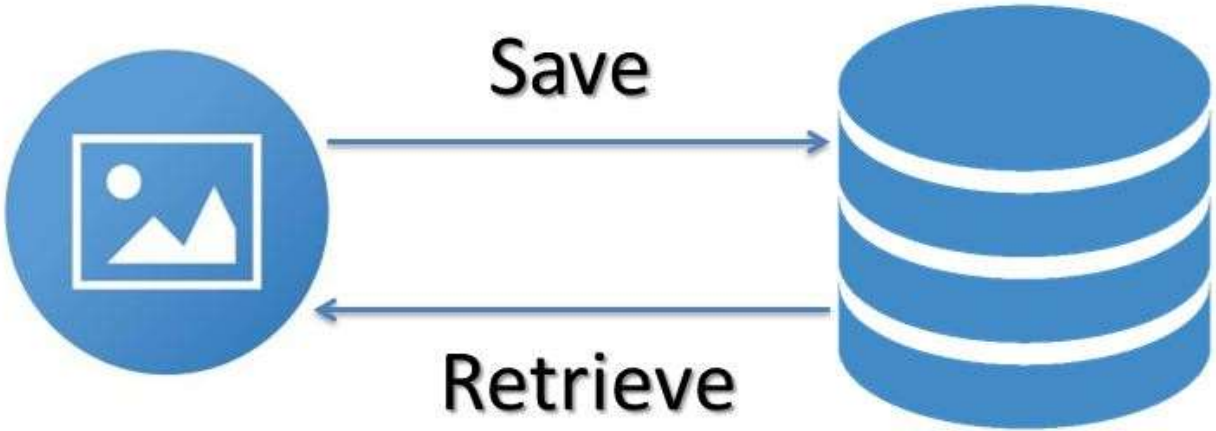


SQL Standards

- IEC — The International Electrotechnical Commission;
- ISO — The International Organization for Standardization;
- ANSI —The American National Standards Institute.


Menu
MenuId
Price
StartDate
EndDate
FoodID
Administrator
AdminId
FName
LName
Username
Password
Status
MenuID
OrderItem
OrderId
FoodId
Quantity
UnitPrice
FoodItem
Food ID
Name
Quantity
Unit Price
ItemCategory
Customer
CustomerID
Email
Phone no
Fname
Lname
PaymentID
FoodID
Payment
PaymentId
CustomerID
OrderID
PaymentDate
Amount
PaymentType
Order
OrderID
OrderDate
CustomerId
Quantity
PickupDate
Chef
ChefId
LastName
FirstName
Username
PhoneNumber
Password
OrderId
1
∞
1
∞

Database Operations



Select Statement

- Used to retrieve Information

Syntax :

- SELECT **COLUMN_NAME** FROM **TABLE_NAME**

Query

- Run in a Backward Manner

```
SELECT column1 FROM table
```

Select Statement

1. SINGLE COLUMN
2. MULTIPLE COLUMN

1. SINGLE COLUMN

- **SELECT** column1 **FROM** table_1

2. MULTIPLE COLUMN

- **SELECT** column1, column2 **FROM** table_1

ALL COLUMNS

- Get All Columns.

```
SELECT * FROM table_1
```

DISTINCT

- To list the distinct values.

```
SELECT DISTINCT column FROM table.
```

DISTINCT

- To list the distinct values.

```
SELECT DISTINCT(column) FROM table.
```

How many different continents are their ?

COUNT

- Returns number of input rows that matches a specific condition of a query.

```
SELECT COUNT(column_name) FROM table_name.
```

Note : It won't work without parenthesis.

COUNT

- Count -> return number of rows
- Count is useful with other commands like distinct.

HOW MANY UNIQUE NAMES ARE THERE IN THE TABLE ?

```
SELECT COUNT(DISTINCT column_name) FROM table_name;
```

WHERE

- Where statement allows us to specify condition on column for the rows.

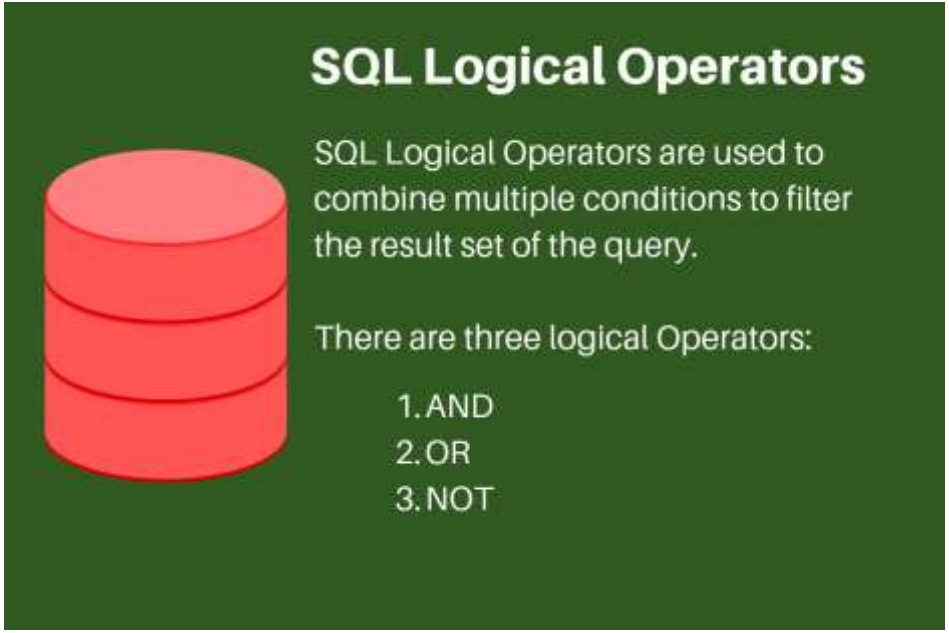
BASIC SYNTAX :

```
SELECT column1, column2
                    FROM table_name
                    WHERE condition ;
```

Comparison Operator

S. No	Operator	Description	Sample Query
1.	<	Less Than	Select * from student where marks < 10;
2.	>	Greater Than	Select * from student where marks > 10;
3.	=	Equal To	Select * from student where marks = 10;
4.	<=	Less Than Or Equal To	Select * from student where marks <= 10;
5.	>=	Greater Than Or Equal To	Select * from student where marks >= 10;
6.	<>	Not Equal To	Select * from student where marks < > 10;

Logical Operator



Logical Operators:

Logical operators in SQL will return either true or false value.

Operators	Function	Example
AND	Check two conditions are true	Select * from emp where basic >= 10000 AND basic <= 20000;
OR	Check any of the two conditions are true	Select * from emp where basic >= 10000 OR dept = 'Sales';
NOT	Reversed the result of logical expression	Select * from emp where NOT(basic >= 10000 OR dept = 'Sales');

Order By

- Compiler returning the output in different order each time.
- Order By is used to sort the output in ascending order or descending order.

```
SELECT column_1, column_2
        FROM table
        ORDER BY column_1 ASC/DESC
```

LIMIT

- Limit command limit the number of rows returned for a query.
- Limit is the last command to be executed’

```
Select * from table_name LIMIT 2
```

Between

- Used to Match the values against range of values.
  - Values Between Low and High
- Between is same as
  - Value >= low AND value <= high

```
SELECT * FROM table_name

WHERE column_1 BETWEEN low AND high
```

IN

- Checks if a value is included in a list of multiple options.

```
SELECT color FROM table_name WHERE color IN ('red', 'blue')
```

Thank You

masai

SQL Fundamentals

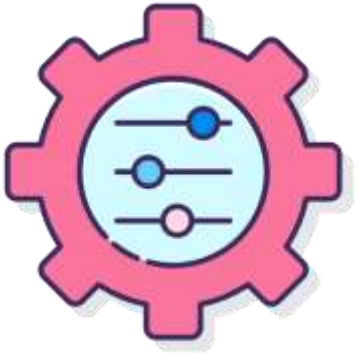
Part - 2

Topics

1. Aggregate Functions
2. Group By
3. Having

Aggregate Functions

- Sql provides variety of aggregate functions.
- It returns single output.



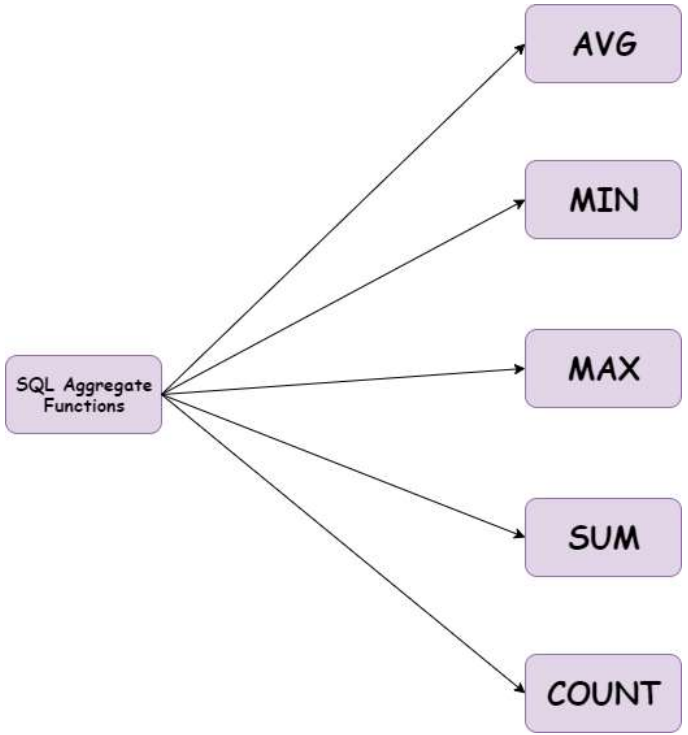
Most Common Aggregate Functions

1. AVG() Return avg value
2. COUNT() Return number of values
3. MAX() Return maximum value
4. MIN() Return minimum value
5. SUM() Return sum of all values

Aggregate Function

- Aggregate function only used with

  Select Clause or having clause.



Group By

- Group By allows us to aggregate columns per some category.

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Group By

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id;
```

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
...	...

DEPT_ID	MAX(SALARY)
-	7000
90	24000
20	13000
...	...

Group By

```
Select category_column, AVG(data_col)
FROM table
GROUP BY category_column
```

Group By

```
Select category_column, AVG(data_col)
FROM table
WHERE category_col != 'A'
GROUP BY category_column
```

Group By

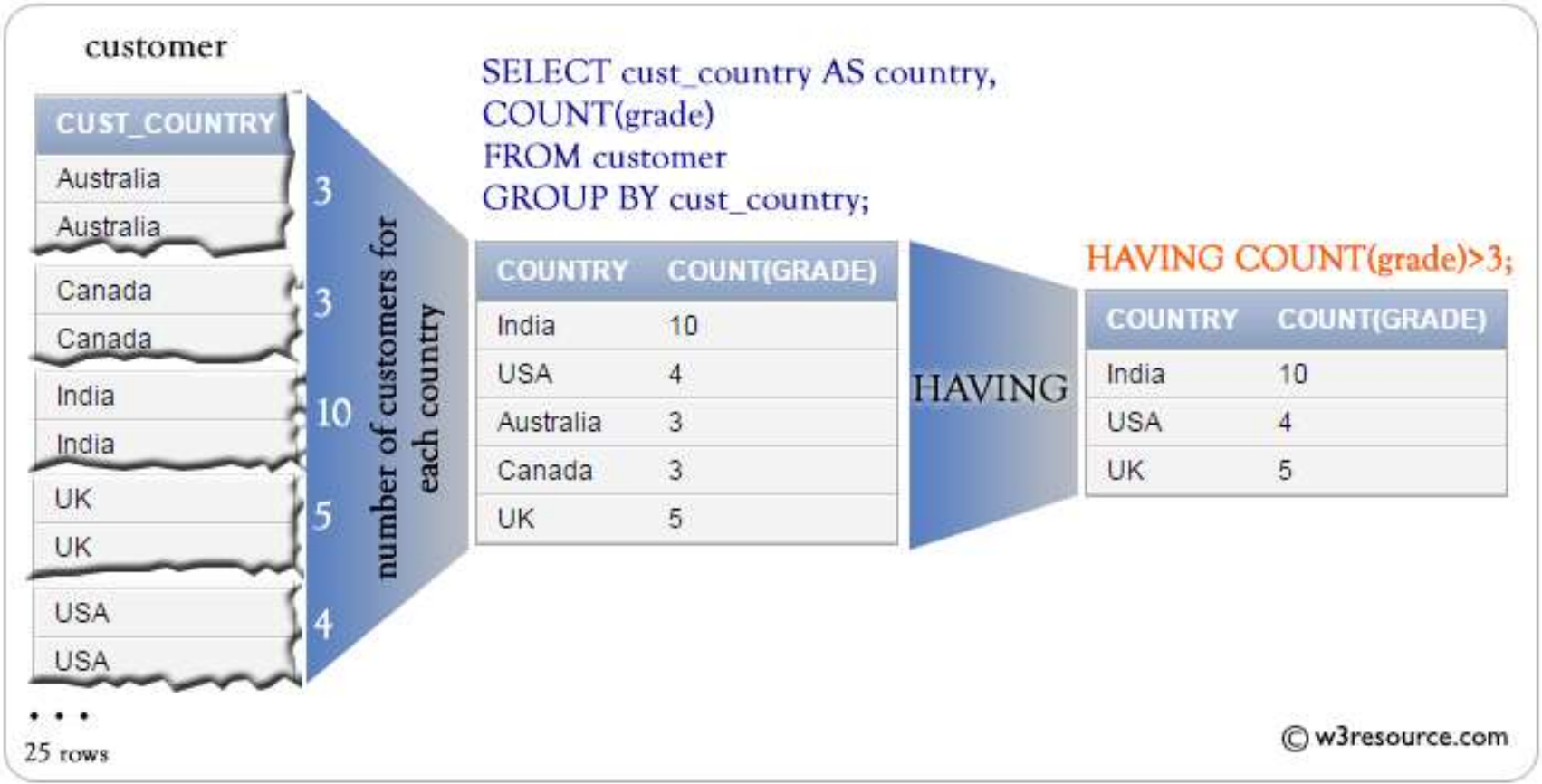
```
Select category_column, AVG(data_col)
FROM table
WHERE category_col != 'A'
GROUP BY category_column
ORDER BY AVG(data_col)
```

Group By

```
Select category_column, AVG(data_col)
FROM table
WHERE category_col != 'A'
GROUP BY category_column
ORDER BY AVG(data_col)
LIMIT 5
```

Having

- Having Clause allows us to filter after an aggregation has already taken place.
- To perform filtering on Group By.


customer
CUST_COUNTRY
Australia
Australia
Canada
Canada
India
India
UK
UK
USA
USA
• • •
25 rows
3
3
10
5
4
number of customers for each country
SELECT cust_country AS country,
COUNT(grade)
FROM customer
GROUP BY cust_country;
COUNTRY COUNT(GRADE)
India 10
USA 4
Australia 3
Canada 3
UK 5
HAVING
HAVING COUNT(grade)>3;
COUNTRY COUNT(GRADE)
India 10
USA 4
UK 5
© w3resource.com

Having

```
Select company, SUM(sales)

FROM finance_table

GROUP BY company

Having sum(sales) > 1000
```

Having

```
Select company, SUM(sales)

FROM finance_table

WHERE company != 'Google'

GROUP BY company

Having sum(sales) > 1000
```

Thank You

masai

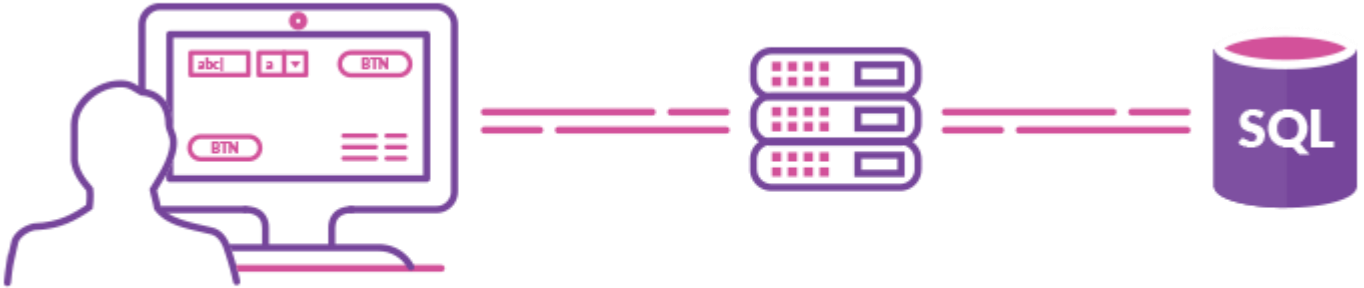
SQL Fundamentals

Part - 3

Topics

1. Data Types
2. Primary Keys & Foreign Keys
3. Constraints
4. Insert, Update, Delete
5. Alter Table
6. Drop Table
7. Check Constraint

Big Picture

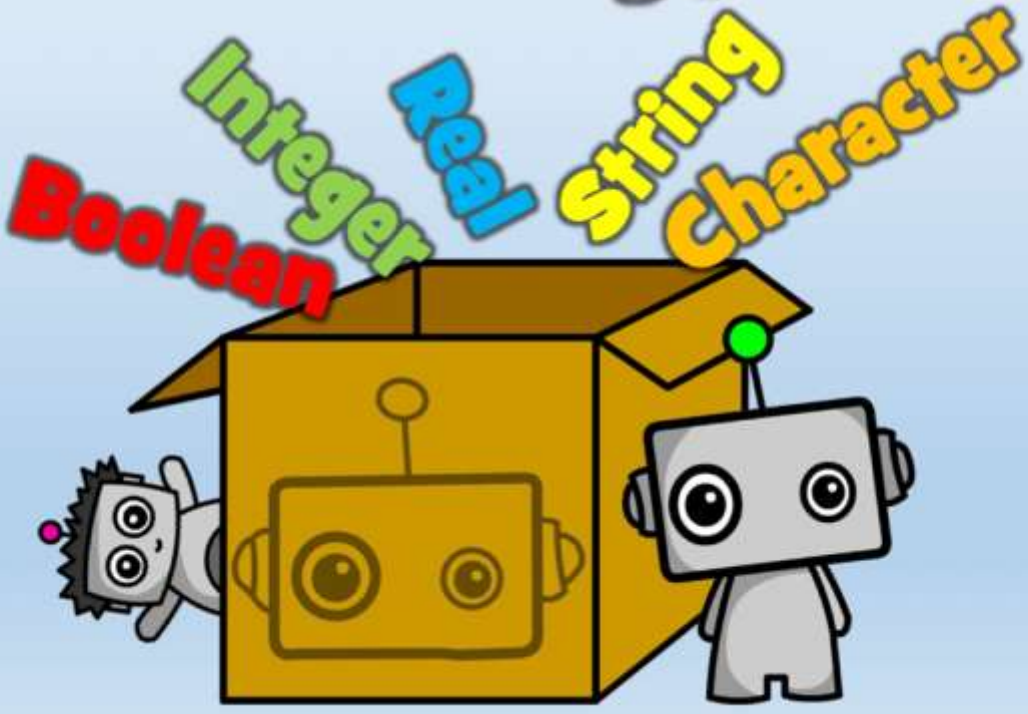


User works with the front-end

The back-end performs business logic and accessses data from the database

GCSE 9-1 Computer Science

Data Types



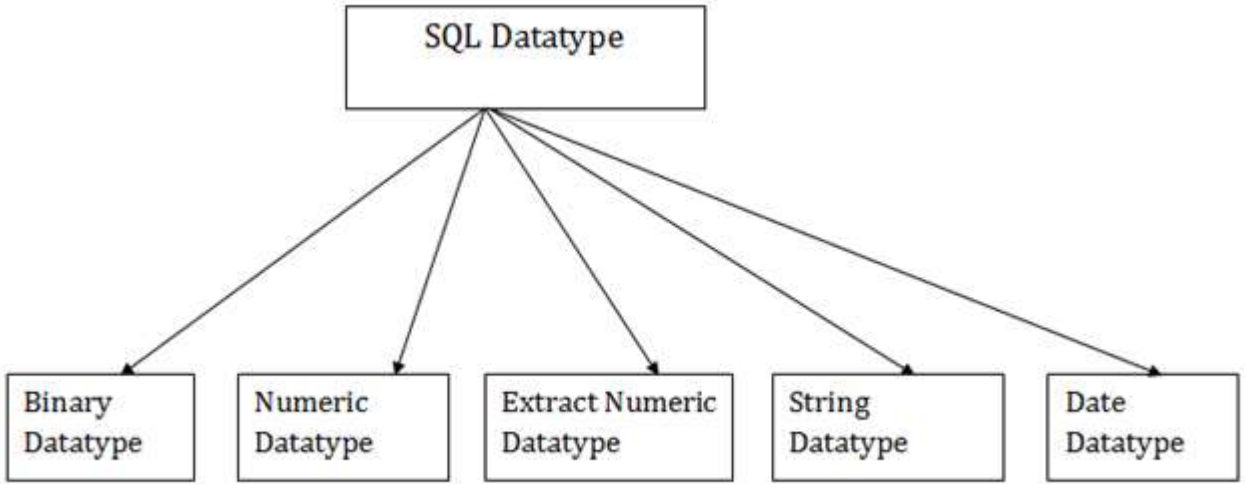
Columns stores a specific data type

Row →
Or record

Emp No	Name	Age	Department	Salary
001	Alex S	26	Store	5000
002	Golith K	32	Marketing	5600
003	Rabin R	31	Marketing	5600
004	Jons	26	Security	5100

Why we need Data Types ?

Data Type


SQL Datatype
Binary Datatype
Numeric Datatype
Extract Numeric Datatype
String Datatype
Date Datatype

Data Types

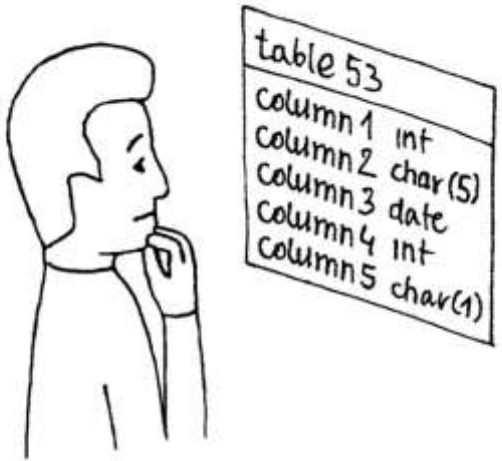


table 53
column 1 int
column 2 char(5)
column 3 date
column 4 int
column 5 char(1)

SQL Data Type Quick Reference

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

Data Types

- On Creating the Database & Tables, Carefully consider which data type should be used for the data to be stored.
- For Eg :
  - I want to store a phone number
  - Which Data type , it should be ?

Which data type should we use ?

	A	B
1	**Customer Name**	**Phone Number**
2	John Smith	5186259017
3	David Jones	8602930292
4	Michael Johnson	7194217400
5	Chris Lee	8572733103
6	Mike Brown	9197678175
7	Mark Williams	3105499319
8	Paul Rodriguez	7703598588
9	Eric Ferguson	6266729887
10		

SQL Data Type Quick Reference

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

Why Bother Numerics at all ?

Data Types

- No Arithmetic Operation

SQL Data Type Quick Reference

Data type	Access	SQLServer	Oracle	MySQL	PostgreSQL
boolean	Yes/No	Bit	Byte	N/A	Boolean
integer	Number (integer)	Int	Number	Int Integer	Int Integer
float	Number (single)	Float Real	Number	Float	Numeric
currency	Currency	Money	N/A	N/A	Money
string (fixed)	N/A	Char	Char	Char	Char
string (variable)	Text (<256) Memo (65k+)	Varchar	Varchar Varchar2	Varchar	Varchar
binary object	OLE Object Memo	Binary (fixed up to 8K) Varbinary (<8K) Image (<2GB)	Long Raw	Blob Text	Binary Varbinary

Data Types

- Take time to Plan
- Modifying Table Later will be painful

Primary Keys

- A primary key is a column or a group of columns used to identify the row uniquely in a table.
- Primary Keys helps in Joining the table.

Primary Key

	employee_id	course_id	taken_date
	100	3	1987-06-17
	101	3	1989-09-21
	102	3	1993-01-13
	103	3	1990-01-03
	104	3	1991-05-21
	105	3	1997-06-25
	106	3	1998-02-05
	107	3	1999-02-07

Why we need Primary Key ?

Primary Key

CustomerID	CustomerName	PhoneNumber
1	Rohit	9876543210
2	Sonal	9765434567
3	Ajay	9765234562
4	Aishwarya	9876567899
5	Akash	9876541236

Foreign Key

- A foreign key is a field in a table that uniquely identifies a row in another table.
- A foreign key is defined in a table that references to the primary key of the other table.

Why we need Foreign Key ?

Create Table

- To create Table.

```
CREATE TABLE table_name(
    column1 datatype,
    column2 datatype,
    ...
    columnN datatype
 );
```

```
CREATE TABLE students(
    id   INT,
    name CHAR(50),
    age  INT
)
```

SQL Constraints

- NOT NULL Constraint
- CHECK Constraint
- UNIQUE Constraint
- PRIMARY KEY Constraint
- FOREIGN KEY Constraint
- DEFAULT Constraint

Constraints

1. Rules enforced on data columns on table..
2. Used to prevent invalid data from being entered in the database.
3. Ensures accuracy and reliability of data.

Constraints

- **PRIMARY KEY** – Ensures that a column has a unique value for each record.
- **NOT NULL** – Enforces that every record has a value for the column
- **DEFAULT** – Specifies a value for the column when a value is not included
- **UNIQUE** – Enforces that each column uses distinct and unique values
- **CHECK** – Enforces specific rules that a column must follow
- **FOREIGN KEY** – Enforces the child relationship with a parent table

```
CREATE TABLE HR.Employees
(EmpID int IDENTITY PRIMARY KEY,
 FirstName varchar(15) NOT NULL,
 LastName varchar(20) NOT NULL,
 JobTitle varchar(20) NOT NULL,
 HireDate date DEFAULT GETDATE(),
 BirthDate date NULL,
 PhoneNumber varchar(15) UNIQUE,
 DeptCode tinyint)
```

Create Table

- To create Table.

```
CREATE TABLE table_name (
column1 datatype(length) column_contraint,
column2 datatype(length) column_contraint,
table_constraints
);
```

Create Table

```
CREATE TABLE players (
player_id SERIAL PRIMARY KEY,
age int NOT NULL
);
```

Serial

- It will create a sequence of the number in the increasing order.
- Used as a primary key.

Create Table

```
CREATE TABLE players (
player_id SERIAL PRIMARY KEY,
age int NOT NULL
);
```

Restaurant Database

1. Customer Table
2. Food Table
3. Order Table

Customer Table

```
CREATE TABLE customer (
customer_id SERIAL PRIMARY KEY,
name VARCHAR(50) NOT NULL,
age INT NOT NULL,
email VARCHAR(50) UNIQUE NOT NULL,
created_on TIMESTAMP NOT NULL
);
```

Food Table

```
CREATE TABLE food (
food_id
name
type
created_on
);
```

Food Table

```
CREATE TABLE food (
food_id SERIAL PRIMARY KEY,
name VARCHAR(50) NOT NULL,
type VARCHAR NOT NULL,
created_on TIMESTAMP NOT NULL
);
```

Order Table

```
CREATE TABLE orders (
customer_id ,
food_id ,
Order_time
);
```

Order Table

```
CREATE TABLE orders (
customer_id INTEGER REFERENCES customer(customer_id),
food_id INTEGER REFERENCES food(food_id),
order_time TIMESTAMP
);
```

INSERT

- Insert allows to add in rows to a table.
- No need to provide values for serial column.

```
INSERT INTO table_name(
    column1, column2,.., columnN)
    VALUES (value1, value2,.., valueN);
```

Customer Table

```
INSERT INTO customer(name, age, email, created_on)
VALUES('Shravan',24,'shravan@gmail.com',CURRENT_TIMESTAMP);
```

Food Table

```
INSERT INTO food(name, type, created_on)
VALUES('Dosa','veg',CURRENT_TIMESTAMP);
```

Order Table

```
INSERT INTO orders(customer_id, food_id, order_time)
VALUES(1,1,CURRENT_TIMESTAMP);
```

Update

- It allows for the changing of value of the columns in a table.

```
UPDATE table_name
    SET column1 = value1, column2 = value2, columnN = valueN
    WHERE condition;
```

Update

```
UPDATE CUSTOMER
SET age = 50 ;
```

- Update ALL Rows.

Update

```
UPDATE CUSTOMER
SET age = 50
Where customer_id = 1 ;
```

- Update specific row.

Update

```
UPDATE CUSTOMER
SET created_on = CURRENT_TIMESTAMP
Where customer_id = 1 ;
```

- Update specific row.

Returning

```
UPDATE CUSTOMER
SET created_on = CURRENT_TIMESTAMP
Where customer_id = 1 ;
RETURNING customer_id,age,email;
```

Delete

- It is used to remove all rows from a table.

```
DELETE FROM customer ;
```

Delete

- It is used to remove rows from a table.

```
DELETE FROM customer
Where customer_id = 1 ;
```

Delete + Returning

- It is used to remove rows from a table.

```
DELETE FROM customer
Where customer_id = 1
RETURNING customer_id, age, email;
```

Alter Table

1. Adding Dropping and Renaming Columns.
2. Changing a Column Data Type.
3. Set Default Values for a column.
4. Add Check Constraints.
5. Rename Table.

```
ALTER TABLE table_name action;
```

General Syntax

1. Adding Column

```
ALTER TABLE table_name
ADD COLUMN column_name datatype column_constraint;
```

General Syntax

2. Removing Column

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

General Syntax

3. Alter Constraints

```
ALTER TABLE table_name
ALTER COLUMN column_name
[SET DEFAULT value | DROP DEFAULT];
```

General Syntax

3. Alter Constraints

```
ALTER TABLE table_name
ALTER COLUMN column_name
[SET NOT NULL| DROP NOT NULL];
```

Drop Table

- The CASCADE option allows you to remove the table and its dependent objects.
- The RESTRICT option rejects the removal if there is any object depends on the table. The RESTRICT option is the default.

```
DROP TABLE [IF EXISTS] table_name
[CASCADE | RESTRICT];
```

Drop Multiple Table

```
DROP TABLE [IF EXISTS]
   table_name_1,
   table_name_2,
   ...
[CASCADE | RESTRICT];
```

Check Constraint

```
DROP TABLE IF EXISTS employees;
CREATE TABLE employees (
        id SERIAL PRIMARY KEY,
        first_name VARCHAR (50),
        last_name VARCHAR (50),
        birth_date DATE CHECK (birth_date > '1900-01-01'),
        joined_date DATE CHECK (joined_date > birth_date),
        salary numeric CHECK(salary > 0)
);
```

```
INSERT INTO employees (first_name, last_name, birth_date, joined_date, salary)
VALUES ('John', 'Doe', '1972-01-01', '2015-07-01', - 100000);
```

Thank You

masai

SQL Fundamentals

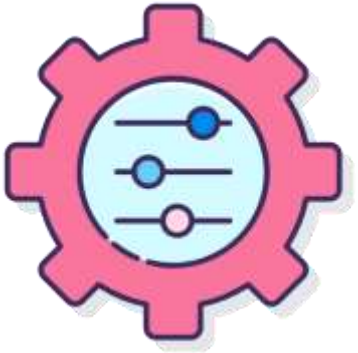
Part - 4

Topics

1. Aggregate Functions
2. Group By
3. Having

Aggregate Functions

- Sql provides variety of aggregate functions.
- It returns single output.



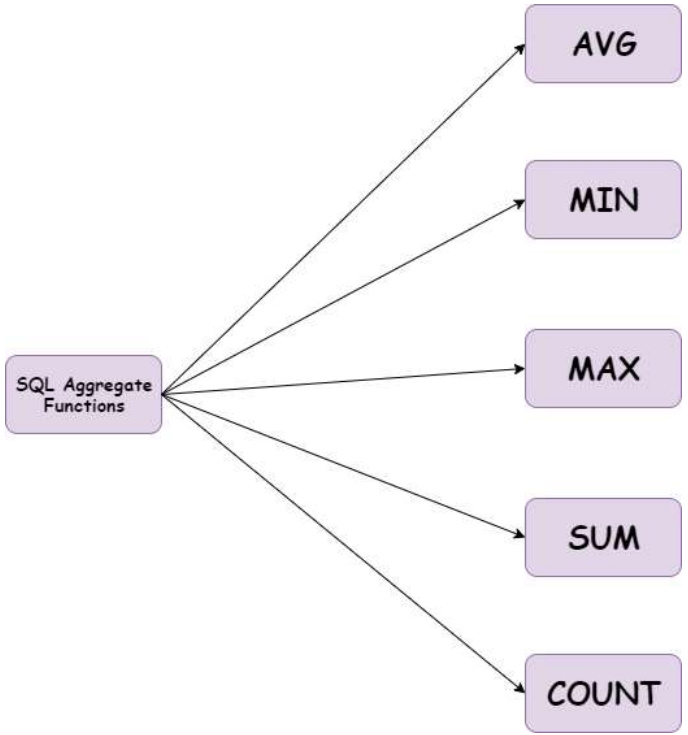
Most Common Aggregate Functions

1. AVG() Return avg value
2. COUNT() Return number of values
3. MAX() Return maximum value
4. MIN() Return minimum value
5. SUM() Return sum of all values

Aggregate Function

- Aggregate function only used with

  Select Clause or having clause.



Group By

- Group By allows us to aggregate columns per some category.

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Group By

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id;
```

DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
...	...

DEPT_ID	MAX(SALARY)
-	7000
90	24000
20	13000
...	...

Group By

```
Select category_column, AVG(data_col)
FROM table
GROUP BY category_column
```

Group By

```
Select category_column, AVG(data_col)
FROM table
WHERE category_col != 'A'
GROUP BY category_column
```

Group By

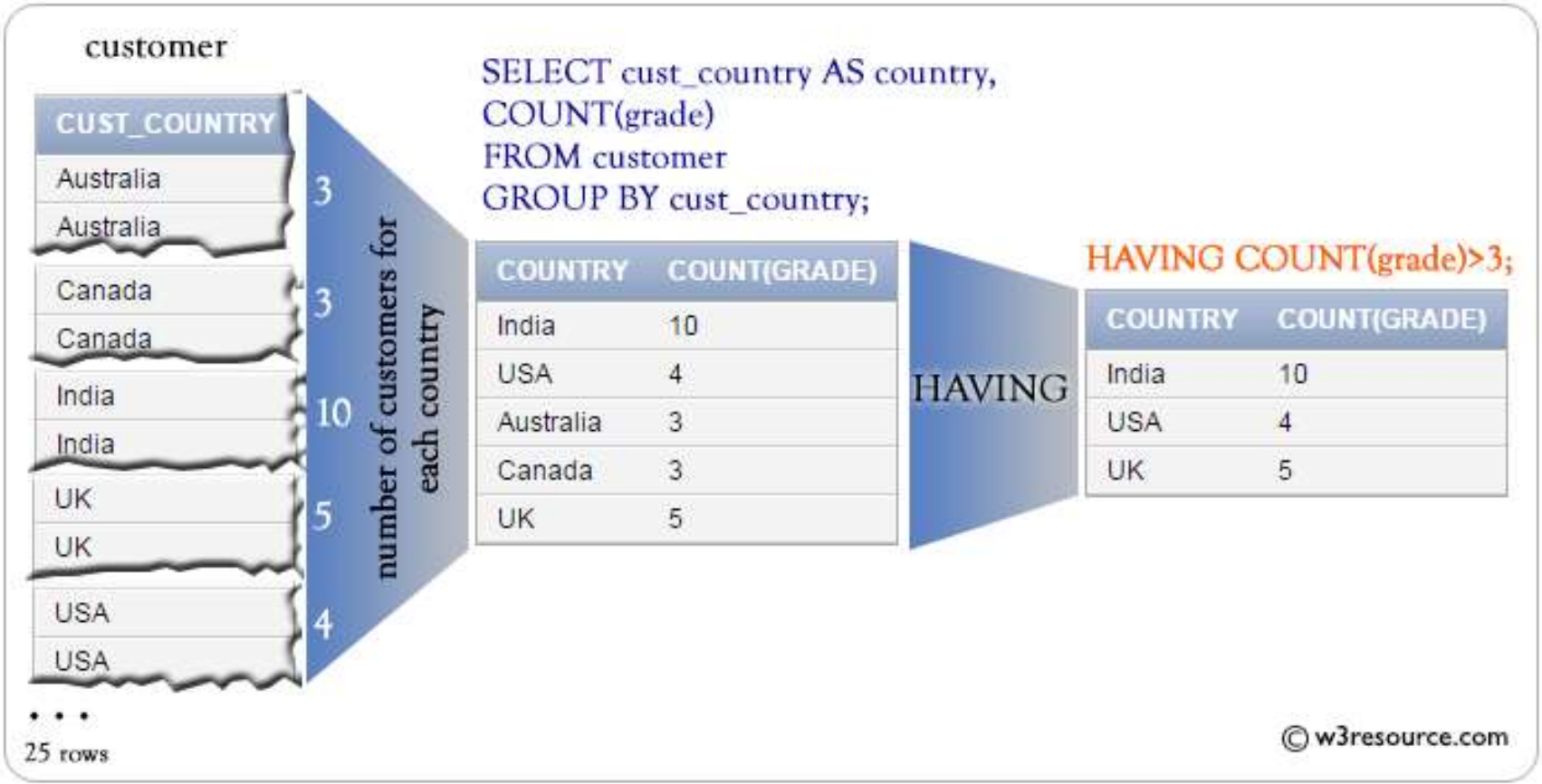
```
Select category_column, AVG(data_col)
FROM table
WHERE category_col != 'A'
GROUP BY category_column
ORDER BY AVG(data_col)
```

Group By

```
Select category_column, AVG(data_col)
FROM table
WHERE category_col != 'A'
GROUP BY category_column
ORDER BY AVG(data_col)
LIMIT 5
```

Having

- Having Clause allows us to filter after an aggregation has already taken place.
- To perform filtering on Group By.


customer
CUST_COUNTRY
Australia
Australia
Canada
Canada
India
India
UK
UK
USA
USA
• • •
25 rows
3
3
10
5
4
number of customers for each country
SELECT cust_country AS country,
COUNT(grade)
FROM customer
GROUP BY cust_country;
COUNTRY COUNT(GRADE)
India 10
USA 4
Australia 3
Canada 3
UK 5
HAVING
HAVING COUNT(grade)>3;
COUNTRY COUNT(GRADE)
India 10
USA 4
UK 5
© w3resource.com

Having

```
Select company, SUM(sales)

FROM finance_table

GROUP BY company

Having sum(sales) > 1000
```

Having

```
Select company, SUM(sales)

FROM finance_table

WHERE company != 'Google'

GROUP BY company

Having sum(sales) > 1000
```

Thank You