Introduction to Databases (Postgres)

Zeyad Ashraf

Agenda

01 SQL

02 Why Postgres

03 History of Postgress

04 Normalization

05 DML

06 DCL

07 Our Demo

Normalization

Database Normalization

Normalization: The process of structuring data to minimize duplication and inconsistencies.

- The process usually involves breaking down a single Table into two or more tables and defining relationships between those tables.

Normalization is usually done in stages, with each stage applying some rules to the types of information which can be stored in a table.

Normalization

- bottom-up Analysis
- used to reduce Null Values
- used to improve performance

Anomalies

- Insertion anomaly
- Deletion anomaly
- Modification anomaly

Example

SID	Sname	Bdate	City	ZipCode	Subject	Grade	Teacher
1	Ahmed	1/1/1980	Cairo	1010	DB	A	Hany
1	Ahmed	1/1/1980	Cairo	1010	Math	B	Eman
1	Ahmed	1/1/1980	Cairo	1010	WinXP	A	khalid
2	Ali	1/1/1983	Alex	1111	DB	B	Hany
2	Ali	1/1/1983	Alex	1111	SWE	B	Heba
3	Mohamed	1/1/1990	Mansoura	1210	NC	C	Mona

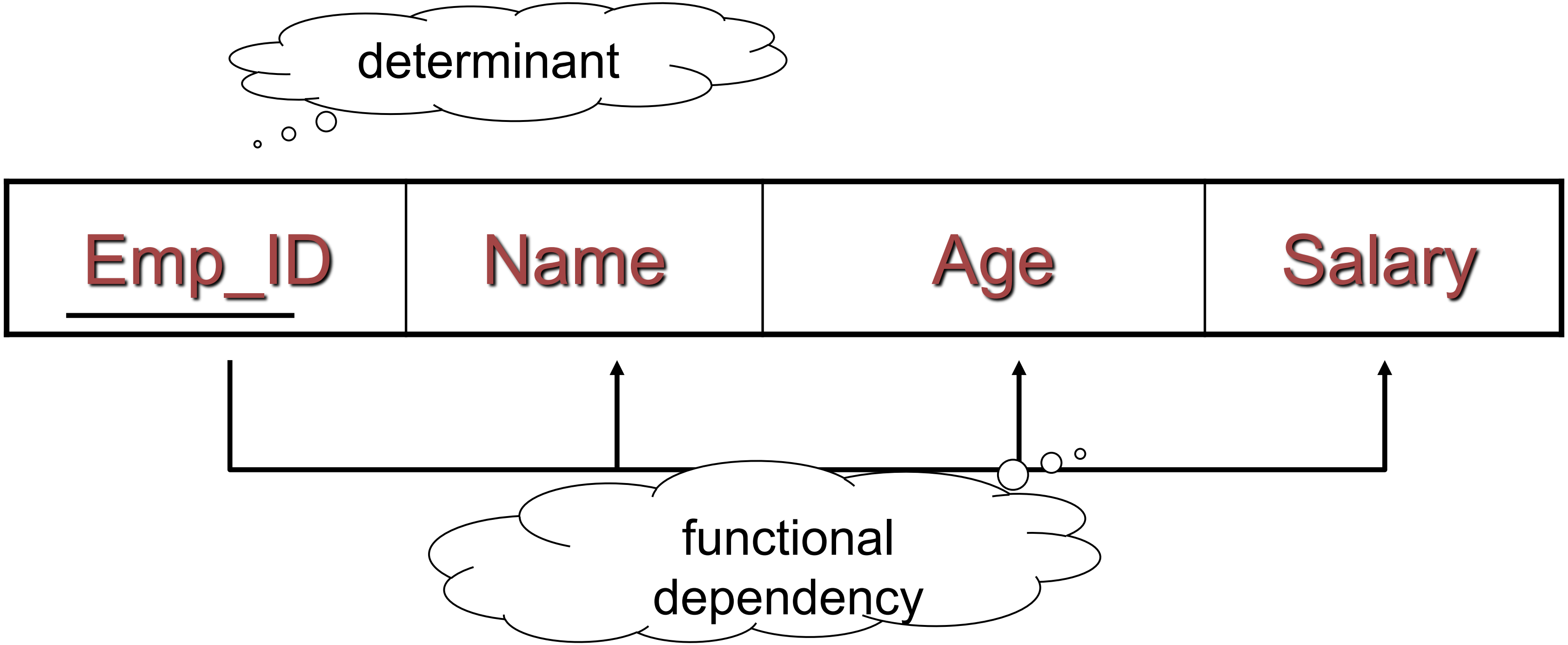
.... functional dependency

some examples

- social security number determines employee name
  SSN -> ENAME
- project number determines project name and location
  PNUMBER -> {PNAME, PLOCATION}
- employee ssn and project number determines the hours per week that the employee works on the project
  {SSN, PNUMBER} -> HOURS

keys and dependencies

EMPLOYEE1 (Emp_ID, Name, Age, Salary)



Types of functional dependency

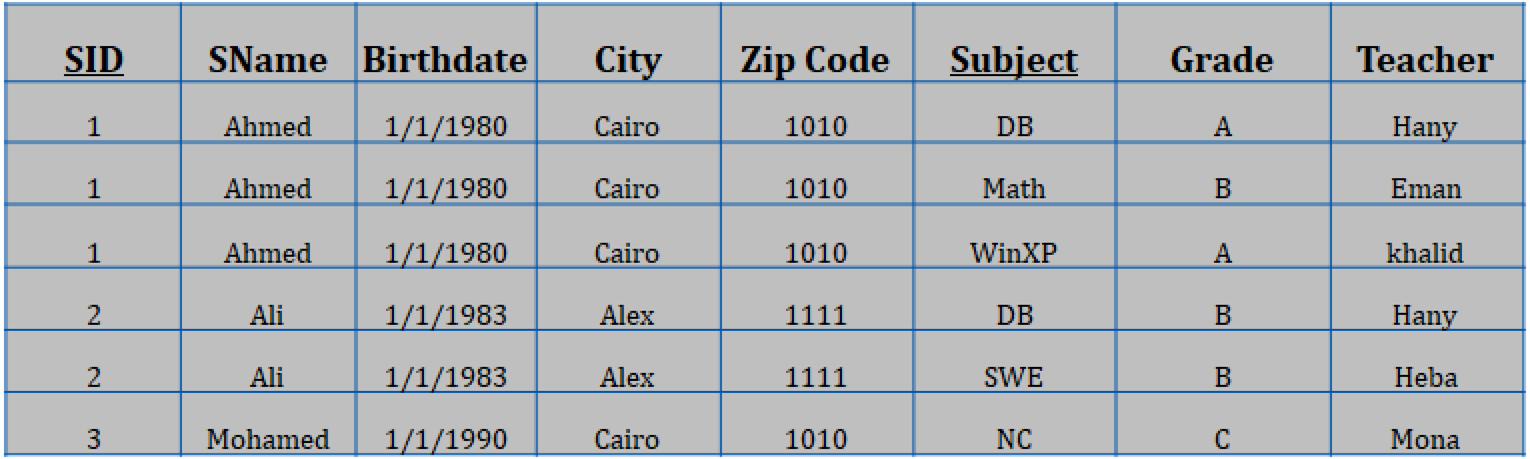
- Full Functional Dependency

  Attribute is fully Functional Dependency on a PK if its value is determined by the whole PK
- Partial Functional Dependency

  Attribute if has a Partially Functional Dependency on a PK if its value is determined by part of the PK(Composite Key)
- Transitive Functional Dependency

  Attribute is Transitively Functional Dependency on a table if its value is determined by anther non-key attribute which itself determined by PK

Example



<u>SID</u>	SName	Birthdate	City	Zip Code	<u>Subject</u>	Grade	Teacher
1	Ahmed	1/1/1980	Cairo	1010	DB	A	Hany
1	Ahmed	1/1/1980	Cairo	1010	Math	B	Eman
1	Ahmed	1/1/1980	Cairo	1010	WinXP	A	khalid
2	Ali	1/1/1983	Alex	1111	DB	B	Hany
2	Ali	1/1/1983	Alex	1111	SWE	B	Heba
3	Mohamed	1/1/1990	Cairo	1010	NC	C	Mona

Full Functional Dependency	Sid,Subject → Grade
Partial Functional Dependency	Sid → SName Subject →Teacher
Transitive Functional Dependency	ZipCode → City

Steps in normalization

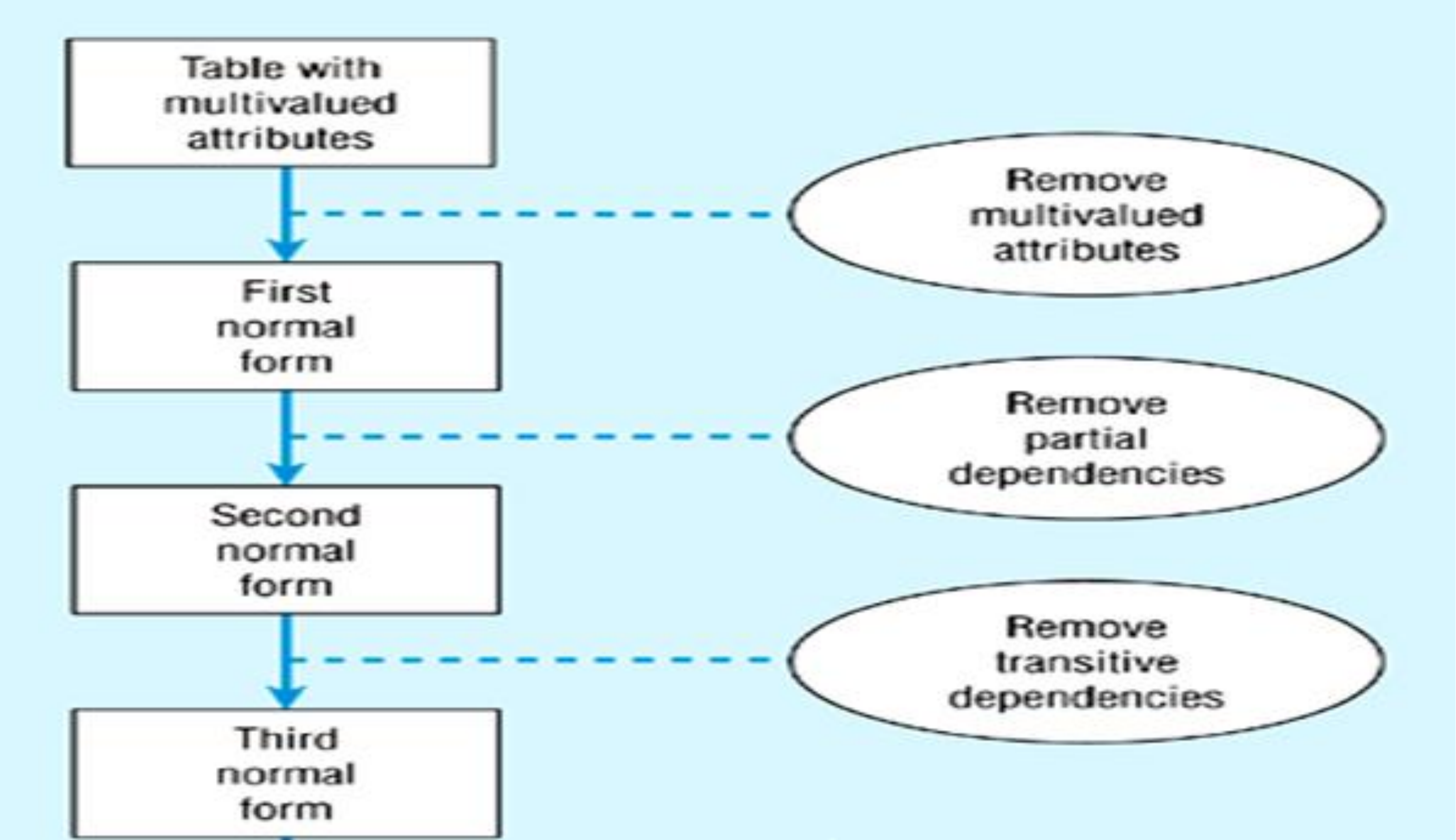


Table with multivalued attributes
Remove multivalued attributes
First normal form
Remove partial dependencies
Second normal form
Remove transitive dependencies
Third normal form

INF

- relation is in first normal form if it contains no multivalued or composite attributes
- remove repeating groups to a new table as already demonstrated, “carrying” the PK as a FK
- All columns (fields) must be atomic

  Means : no repeating items in columns

INF

Student(SID, Sname, Birthdate, City, Zip Code)

SID	SName	Birthdate	City	Zip Code
1	Ahmed	1/1/1980	Cairo	1010
2	Ali	1/1/1983	Alex	1111
3	Mohamed	1/1/1990	Cairo	1010

Stud_Subject (SID, Subject, Grade, Teacher)

SID	Subject	Grade	Teacher
1	DB	A	Hany
1	Math	B	Eman
1	WinXP	A	khalid
2	DB	B	Hany
2	SWE	B	Heba
3	NC	C	Mona

2NF

- a relation is in second normal form if it is in first normal form AND every nonkey attribute is fully functionally dependant on the primary key
- i.e. remove partial functional dependencies, so no nonkey attribute depends on just part of the key

2NF

Student(SID, Sname, Birthdate, City, Zip Code)

SID	SName	Birthdate	City	Zip Code
1	Ahmed	1/1/1980	Cairo	1010
2	Ali	1/1/1983	Alex	1111
3	Mohamed	1/1/1990	Cairo	1010

2NF
Because there is no Composite PK

Stud_Subject (SID, Subject, Grade, Teacher)

SID	Subject	Grade	Teacher
1	DB	A	Hany
1	Math	B	Eman
1	WinXP	A	khalid
2	DB	B	Hany
2	SWE	B	Heba
3	NC	C	Mona

SID, Subject → Grade……FFD

Subject → Teacher……PFD

2NF

Student(SID, Sname, Birthdate, City, Zip Code)

SID	SName	Birthdate	City	Zip Code
1	Ahmed	1/1/1980	Cairo	1010
2	Ali	1/1/1983	Alex	1111
3	Mohamed	1/1/1990	Mansoura	1210

Stud_Subject (SID, Subject, Grade)

SID	Subject	Grade
1	DB	A
1	Math	B
1	WinXP	A
2	DB	B
2	SWE	B
3	NC	C

Subject (Subject,Teacher)

Subject	Teacher
DB	Hany
Math	Eman
WinXP	khalid
SWE	Heba
NC	Mona

Third Normal Form

- 2NF PLUS no transitive dependencies (one attribute functionally determines a second, which functionally determines a third)

2NF

Student(SID, Sname, Birthdate, City, Zip Code)

SID	SName	Birthdate	City	Zip Code
1	Ahmed	1/1/1980	Cairo	1010
2	Ali	1/1/1983	Alex	1111
3	Mohamed	1/1/1990	Cairo	1010

Zip Code ->City …….TFD

Stud_Subject (SID, Subject, Grade)

SID	Subject	Grade
1	DB	A
1	Math	B
1	WinXP	A
2	DB	B
2	SWE	B
3	NC	C

Subject (Subject,Teacher)

Subject	Teacher
DB	Hany
Math	Eman
WinXP	khalid
SWE	Heba
NC	Mona

3NF

Because there is no Transtive Functional Dependency

3NF

Student(SID, Sname, Birthdate,)

SID	SName	Birthdate	ZipCode
1	Ahmed	1/1/1980	1010
2	Ali	1/1/1983	1111
3	Mohamed	1/1/1990	1010

Stud_City(City, Zip Code)

City	Zip Code
Cairo	1010
Alex	1111

Stud_Subject (SID, Subject, Grade)

SID	Subject	Grade
1	DB	A
1	Math	B
1	WinXP	A
2	DB	B
2	SWE	B
3	NC	C

Subject (Subject,Teacher)

Subject	Teacher
DB1	Hany
Math	Eman
WinXP	khalid
DB2	Hany
SWE	Heba
NC	Mona

SQL

SQL

ANSI-SQL defined by the American National Standards Institute

Categories

DML – Data Manipulation Language

DCL – Data Control Language

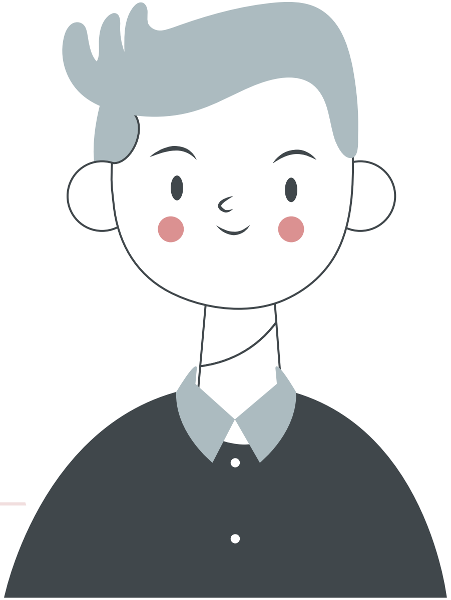
DDL – Data Definition Language

TCL – Transactional Control Language

DQL– Data Query Language (Select)

Why PostgreSQL?

- Designed for high volume environments.
- Cross platform
- Low / No Cost.
- Stability
- Open Source



History of PostgreSQL

- PostgreSQL is derived from the POSTGRES package written at the University of California by a computer science professor named Michael Stonebraker
- POSTGRES used PostQUEL as query language

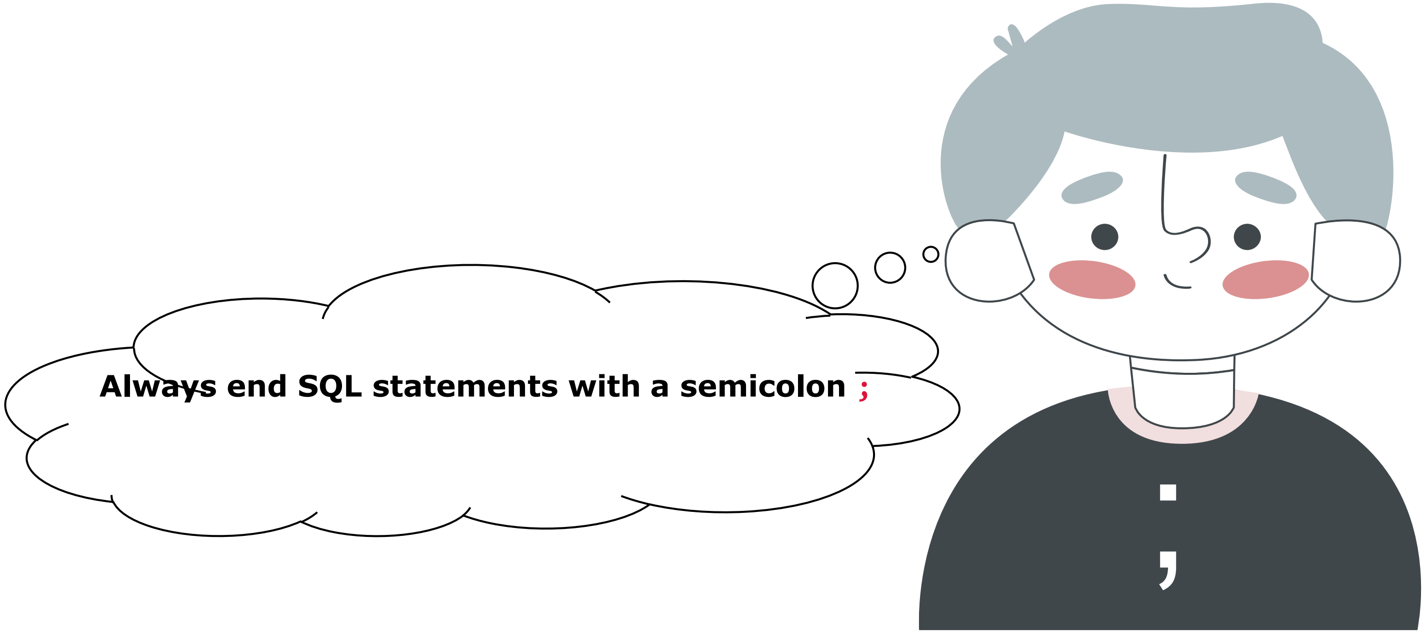
Installation

- https://sbp.enterprisedb.com/getfile.jsp?fileid=1259601

Resources :

- https://www.tutorialspoint.com/postgresql/index.htm
- https://www.pgtutorial.com/

Very important !



Always end SQL statements with a semicolon ;

Version

- Syntax :

```
SELECT version();
```

Rename Database

- Syntax :

```
ALTER DATABASE old_database_name RENAME TO
new_database_name;
```

create table

- Syntax :

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

- Example:

```
CREATE TABLE cars (
  brand VARCHAR(255),
  model VARCHAR(255),
  year INT
);
```

Constraints

NOT NULL Constraint – Ensures that a column cannot have NULL value.

UNIQUE Constraint – Ensures that all values in a column are different.

PRIMARY Key – Uniquely identifies each row/record in a database table.

FOREIGN Key – Constrains data based on columns in other tables.

CHECK Constraint – The CHECK constraint ensures that all values in a column satisfy certain conditions.

EXCLUSION Constraint – The EXCLUDE constraint ensures that if any two rows are compared on the specified column(s) or expression(s) using the specified operator(s), not all of these comparisons will return TRUE.

DML

Selecting All Data from a Table

- Syntax :

```
SELECT * FROM table_name;
```

- Example:

```
SELECT * FROM cars;
```

Inserting Data into a Table

- Syntax :

```
INSERT INTO table_name (column1, column2, column3)
VALUES (value1, value2, value3);
```

- Example:

```
INSERT INTO cars (brand, model, year)
VALUES ('Ford', 'Mustang', 1964);
```

Insert Multiple Rows

- Syntax :

```
INSERT INTO table_name (column1, column2, column3)
VALUES
  (value1, value2, value3),
  (value4, value5, value6),
  ...;
```

- Example:

```
INSERT INTO cars (brand, model, year)
VALUES
  ('Volvo', 'p1800', 1968),
  ('BMW', 'M1', 1978),
  ('Toyota', 'Celica', 1975);
```

Selecting Specific Columns

- Syntax :

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

- Example:

```
SELECT brand, year FROM cars;
```

Altering a Table – Add Column

- Syntax :

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

- Example:

```
ALTER TABLE cars
ADD color VARCHAR(255);
```

UPDATE Statement

- Syntax :

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

- Example :

```
UPDATE cars
SET color = 'red'
WHERE brand = 'Volvo’;
```

Without the WHERE clause, ALL records will be updated

Updating Multiple Columns

- Syntax :

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

- Example:

```
UPDATE cars
SET color = 'white', year = 1970
WHERE brand = 'Toyota';
```

Alter Column Type

- Syntax :

```
ALTER TABLE table_name
ALTER COLUMN column_name TYPE new_data_type;
```

- Example:

```
ALTER TABLE cars
ALTER COLUMN year TYPE VARCHAR(4);
```

Note: Some data types cannot be converted if the column has value.
E.g. numbers can always be converted to text, but text cannot always be converted to numbers.

Change Maximum Allowed Characters

- Syntax :

```
ALTER TABLE table_name
ALTER COLUMN column_name TYPE new_data_type;
```

- Example:

```
ALTER TABLE cars
ALTER COLUMN color TYPE VARCHAR(30);
```

Drop a Column from a Table

- Syntax :

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

- Example:

```
ALTER TABLE cars
DROP COLUMN color;
```

Operators in the WHERE clause

Degree	Define
=	Equal to
<	Less than
>	Greater than
<=	Greater than or equal to
>=	Greater than or equal to
<> !=	Not equal to
LIKE	Check if a value matches a pattern (case sensitive)

Degree	Define
ILIKE	Check if a value matches a pattern (case insensitive)
AND	Logical AND
OR	Logical OR
IN	Check if a value is between a range of values
BETWEEN	Check if a value is between a range of values
IS NULL	Check if a value is NULL
NOT	Makes a negative result e.g. NOT LIKE, NOT IN, NOT BETWEEN

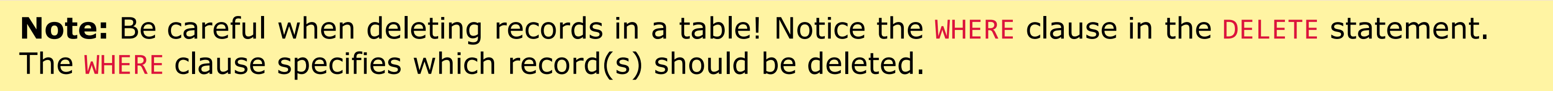
Delete Column

- Syntax :

```
DELETE FROM table_name
WHERE condition;
```

- Example:

```
DELETE FROM cars
WHERE brand = 'Volvo';
```



Note: Be careful when deleting records in a table! Notice the `WHERE` clause in the `DELETE` statement. The `WHERE` clause specifies which record(s) should be deleted.

Delete All Rows from a Table

- Syntax :

```
DELETE FROM table_name;
```

- Example:

```
DELETE FROM cars;
```

Truncate a Table

- Syntax :

```
TRUNCATE TABLE table_name;
```

- Example:

```
TRUNCATE TABLE cars;
```

DROP TABLE

- Syntax :

```
DROP TABLE table_name;
```

- Example:

```
DROP TABLE cars;
```

Note: Be careful before dropping a table. Deleting a table will result in loss of all information stored in the table!

DELETE, TRUNCATE, and DROP

	DELETE	TRUNCATE	DROP
Purpose	Removes specific row	Removes all rows	Removes the entire table
WHERE clause	Yes	No	No
Rollback supported	Yes (Transactional)	Yes (in most DBMSs)	No (irreversible)
structure	Keeps table & schema	Keeps table & schema	Deletes structure & data
Resets auto-increment	No	Yes	Yes
Speed	Slower (row-by-row)	Faster (bulk delete)	Fastest (structure-level)
Triggers	Yes	No	No

Demo Database

DISTINCT

- Syntax :

```
SELECT DISTINCT column_name
FROM table_name;
```

- Example:

```
SELECT DISTINCT country
FROM customers;
```

The SELECT DISTINCT statement is used to return only distinct (different) values.
It removes duplicate entries from the result set.

Filter Records WHERE

- Syntax :

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

- Example:

```
SELECT * FROM customers
WHERE city = 'London';
```

The WHERE clause is used to filter records.

It is used to extract only those records that fulfill a specified condition.

Sort Data

- Syntax :

```
SELECT * FROM table_name
ORDER BY column_name
[ASC|DESC];
```

- Example:

```
SELECT * FROM products
ORDER BY price;
```

```
SELECT * FROM products
ORDER BY price DESC;
```

The ORDER BY keyword is used to sort the result in ascending or descending order.

LIMIT & OFFSET

- Syntax :

```
SELECT * FROM table_name
LIMIT number;
```

```
SELECT * FROM table_name
LIMIT number
OFFSET number;
```

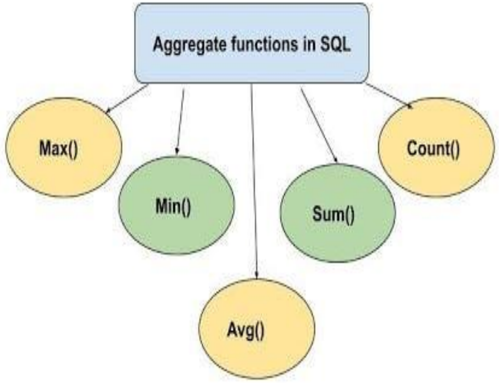
- Example:

```
SELECT * FROM customers
LIMIT 20;
```

```
SELECT * FROM customers
LIMIT 20 OFFSET 40;
```

The LIMIT clause is used to limit the maximum number of records to return.

Aggregation Functions



Aggregation Functions

- What Are Aggregation Functions?

Aggregation functions are built-in SQL functions used to perform calculations on multiple rows and return a single summarized value.

- Why Were They Introduced?

  To analyze data efficiently by summarizing large datasets.
  To gain insights without manually iterating over rows.
  To group and compute meaningful statistics like totals, averages, counts, etc.

MIN()

- Syntax :

```
SELECT MIN(column_name)
FROM table_name;
```

- Example:

```
SELECT MIN(price)
FROM products;
```

The MIN() function returns the smallest value in the selected column.

MAX()

- Syntax :

```
SELECT MAX(column_name)
FROM table_name;
```

- Example:

```
SELECT MAX(price)
FROM products;
```

The MAX() function returns the highest value in the selected column.

COUNT()

- Syntax :

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

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

- Example:

```
SELECT
COUNT(customer_id)
FROM customers;
```

```
SELECT COUNT(customer_id)
FROM customers
WHERE city = 'London';
```

The COUNT() function returns the number of non-NULL values in the specified column.

SUM()

- Syntax :

```
SELECT SUM(column_name)
FROM table_name;
```

- Example:

```
SELECT SUM(quantity)
FROM order_details;
```

The SUM() function returns the total sum of values in a numeric column – in this case, the total quantity of all orders.

AVG()

- Syntax :

```
SELECT AVG(column_name)
FROM table_name;
```

- Example:

```
SELECT AVG(price)
FROM products;
```

```
SELECT AVG(price)::NUMERIC(10,2)
FROM products;
```

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

Thanks

zeyadashraf015@gmail.com

01097143595

Zeyad Elmalky