SQL

Structured Query Language

What does SQL mean ?

SQL is a kind of encoding language in which we query data in a database.

What is a database ?

- A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS) .Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950

Each structure that holds data in tables and rows in lists is actually its own database.

What does the database consist of ?

customer_id	first_name	last_name	phone	email	street	city	state	zip_code
1	Debra	Burks	NULL	debra.burks@yahoo.com	9273 Thorne Ave.	Orchard Park	NY	14127
2	Kasha	Todd	NULL	kasha.todd@yahoo.com	910 Vine Street	Campbell	CA	95008
3	Tameka	Fisher	NULL	tameka.fisher@aol.com	769C Honey Creek St.	Redondo Beach	CA	90278
4	Daryl	Spence	NULL	daryl.spence@aol.com	988 Pearl Lane	Uniondale	NY	11553
5	Charolette	Rice	(916) 381-6003	charolette.rice@msn.com	107 River Dr.	Sacramento	CA	95820
6	Lyndsey	Bean	NULL	lyndsey.bean@hotmail.com	769 West Road	Fairport	NY	14450
7	Latasha	Hays	(716) 986-3359	latasha.hays@hotmail.com	7014 Manor Station Rd.	Buffalo	NY	14215
8	Jacquline	Duncan	NULL	jacquline.duncan@yahoo.com	15 Brown St.	Jackson Heights	NY	11372
9	Genoveva	Baldwin	NULL	genoveva.baldwin@msn.com	8550 Spruce Drive	Port Washington	NY	11050
10	Pamelia	Newman	NULL	pamelia.newman@gmail.com	476 Chestnut Ave.	Monroe	NY	10950

- Tables
- Columns
- Rows
- Index

What is a database server ?

A database server runs a database management system and provides database services to clients. The server manages data access and retrieval and completes clients' requests.

- Database server is software.
- It listens to the system over the network and sends the desired data according to incoming commands.
- Systems such as SQL Server, MySQL, PostgreSQL, Oracle are database servers.
- Access and Excel are not servers.

Types of databases

- Relational databases
- Object-oriented databases
- Distributed databases
- Data warehouses
- NoSQL databases
- Graph databases
- Open source databases
- Cloud databases

What is a Relational Database (RDBMS)?

- A relational database is a type of database that stores and provides access to data points that are related to one another. Relational databases are based on the relational model, an intuitive, straightforward way of representing data in tables. In a relational database, each row in the table is a record with a unique ID called the key. The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.

- Orderfiche - Order information and this table is called the master table.
- Orderline - Detail table

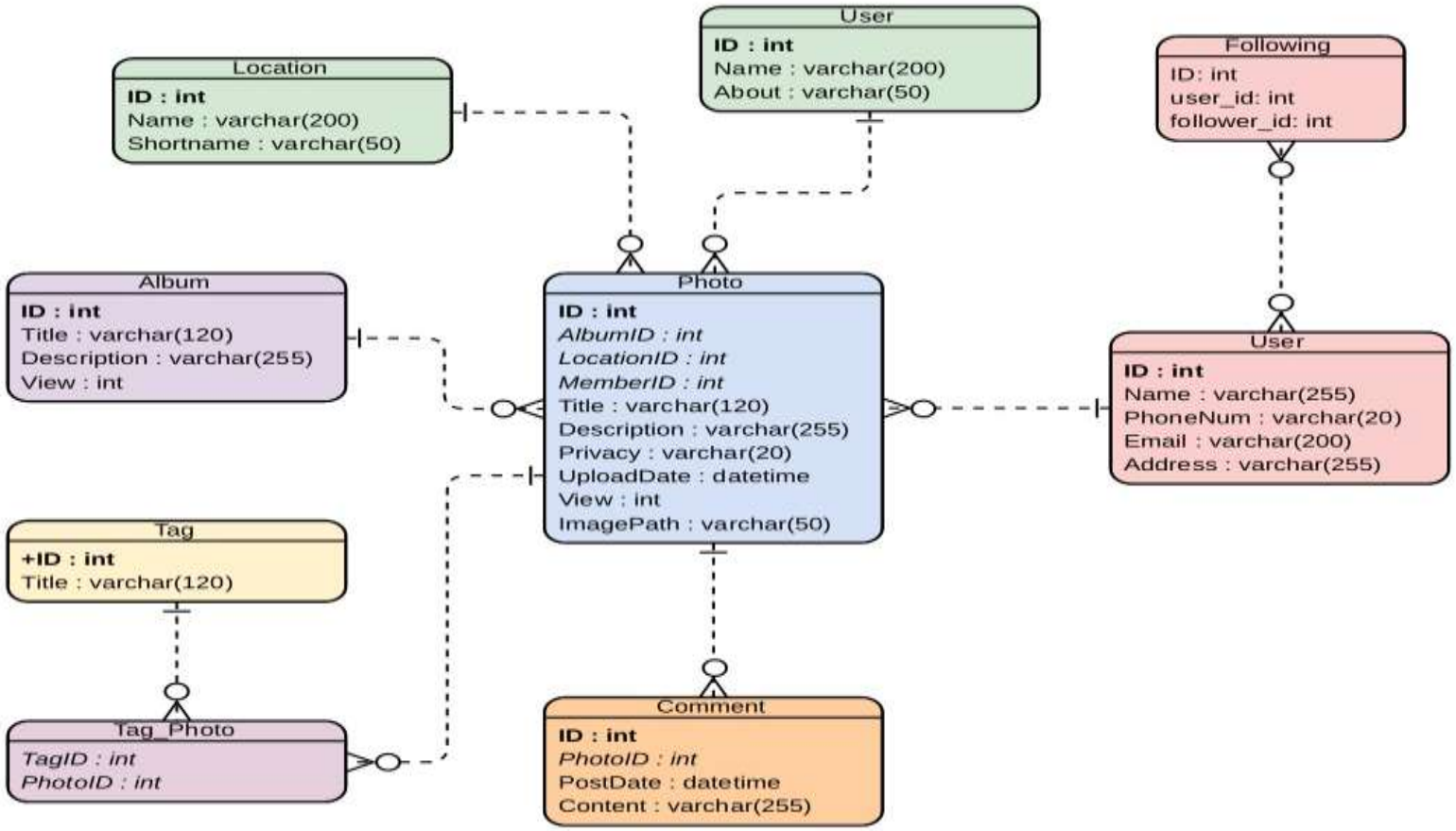
ORDERFICHE

ID	FICHENO	PAYMENTTYPE	CUSTOMERID	DELIVERYTYPE	DATE_	NETPRICE	TOTALVAT	TOTALPRICE
1	FICHE001	1	2602	1	1.05.2020	12.000	2.160	14.160

ORDERLINE

ID	ORDERFICHEID	ITEMID	AMOUNT	UNITPRICE	VAT	VATAMOUNT	NETTOTAL	TOTAL
1	1	2305	1	4.000	18	720	4.000	4.720
2	1	5120	2	8.000	18	1.440	8.000	9.440

RDMS Scheme


User
ID : int
Name : varchar(200)
About : varchar(50)
Location
ID : int
Name : varchar(200)
Shortname : varchar(50)
Following
ID: int
user_id: int
follower_id: int
Album
ID : int
Title : varchar(120)
Description : varchar(255)
View : int
Photo
ID : int
AlbumID : int
LocationID : int
MemberID : int
Title : varchar(120)
Description : varchar(255)
Privacy : varchar(20)
UploadDate : datetime
View : int
ImagePath : varchar(50)
User
ID : int
Name : varchar(255)
PhoneNum : varchar(20)
Email : varchar(200)
Address : varchar(255)
Tag
+ID : int
Title : varchar(120)
Tag_Photo
TagID : int
PhotoID : int
Comment
ID : int
PhotoID : int
PostDate : datetime
Content : varchar(255)

Database Management Systems

- Oracle RDBMS
- IBM DB2
- Altibase
- Microsoft SQL Server
- SAP Sybase ASE
- Teradata
- ABABAS
- MySQL
- SQLite
- FileMaker
- Iformix
- PostgresSQL
- AmazonRDS
- MangoDB
- SQL Developer

Basic SQL Commands

Data Manipulation Commands

SELECT
INSERT
UPDATE
DELETE
TRUNCATE

DataBase Manipulation Commands

CREATE
ALTER
DROP

Data Manipulation Commands

- SELECT : Select statement retrieves the data from database according to the constraints specifies alongside.
- INSERT : Insert statement is used to insert data into database tables.
- UPDATE : The update command updates existing data within a table.
- DELETE : Deletes records from the database table according to the given constraints.
- TRUNCATE : Deletes data from tables

DataBase Manipulation Commands

- CREATE - Creates a database object.
- ALTER - Changes the property of a database object.
- DROP - Deletes a database object.

- **CREATE DATABASE** : Creates a new database.
- **ALTER DATABASE** : Changes the property of a database.
- **CREATE TABLE** : Creates a new table.
- **ALTER TABLE** : Changes the property of a table.
- **DROP TABLE** : Completely deletes a table.
- **CREATE INDEX** : Creates an index.
- **DROP INDEX** : Deletes the index

SELECT Statement

Syntax

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

- `SELECT * FROM table_name;`

The SQL INSERT INTO Statement

- INSERT INTO Syntax,

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

- ```
  INSERT INTO table_name
  VALUES (value1, value2, value3, ...);
  ```

The SQL UPDATE Statement

- UPDATE Syntax

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


  ```

Example,

```
UPDATE Customers  -- (Table Name)
SET ContactName='Juan'
WHERE Country='Mexico';
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Alfred Schmidt	Obere Str. 57	Frankfurt	12209	Germany
2	Ana Trujillo Emparedados y helados	Juan	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Juan	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

The SQL WHERE Clause

- WHERE Syntax

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


  ```

The SQL ORDER BY Keyword

- ORDER BY Syntax,
-
```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

- ORDER BY Example,
-
```
SELECT * FROM Customers
ORDER BY CustomerName DESC;
```

SQL DELETE Statement

- DELETE Syntax,

- `DELETE FROM table_name WHERE condition;`
- `DELETE FROM table_name;`
- `DELETE FROM Customers;`

AGGREGATE FUNCTION

- SUM - Function returns the total sum of a numeric column.
- MIN - Function returns the smallest value of the selected column.
- MAX - Function returns the largest value of the selected column.
- AVG - Function returns the average value of a numeric column.
- COUNT-Function returns the number of rows that matches a specified criterion.

MIN() Syntax & MAX() Syntax

- ```
  SELECT MIN(column_name)
  FROM table_name
  WHERE condition;
  ```

- ```
  SELECT MAX(column_name)
  FROM table_name
  WHERE condition;
  ```

The SQL COUNT(), AVG() and SUM() Functions

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

- ```
  SELECT AVG(column_name)
  FROM table_name
  WHERE condition;
  ```

- ```
  SELECT SUM(column_name)
  FROM table_name
  WHERE condition;
  ```


  ```

SQL Numeric Data Type

Datatype	From	To
bit	0	1
tinyint	0	255
smallint	-32,768	32,767
int	-2,147,483,648	2,147,483,647
bigint	-9,223,372,036, 854,775,808	9,223,372,036, 854,775,807
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

SQL Character and String Data Types

Datatype	Description
CHAR	Fixed length with a maximum length of 8,000 characters
VARCHAR	Variable-length storage with a maximum length of 8,000 characters
VARCHAR(max)	Variable-length storage with provided max characters, not supported in MySQL
TEXT	Variable-length storage with maximum size of 2GB data

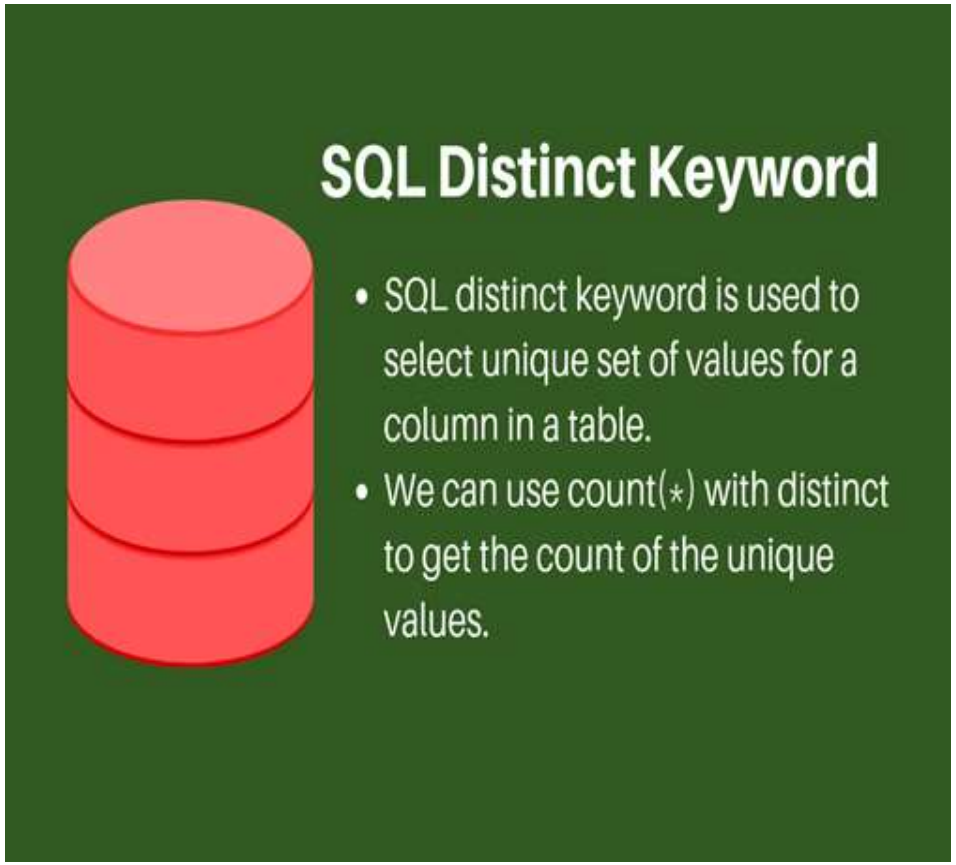
SQL Date and Time Data Types

Datatype	Description
DATE	Stores date in the format YYYY-MM-DD
TIME	Stores time in the format HH:MI:SS
DATETIME	Stores date and time information in the format YYYY-MM-DD HH:MI:SS
TIMESTAMP	Stores number of seconds passed since the Unix epoch ('1970-01-01 00:00:00' UTC)
YEAR	Stores year in 2 digits or 4 digit format. Range 1901 to 2155 in 4-digit format. Range 70 to 69, representing 1970 to 2069.

SQL Binary Data Types

Datatype	Description
BINARY	Fixed length with a maximum length of 8,000 bytes
VARBINARY	Variable-length storage with a maximum length of 8,000 bytes
VARBINARY(max)	Variable-length storage with provided max bytes
IMAGE	Variable-length storage with maximum size of 2GB binary data

SQL Distinct



- It is very often that we have duplicate data available as part of the data storage. For example, as part of an address table, the state column can have the same value multiple times.
- In such cases when we need to identify the unique set of values we use SQL DISTINCT keyword along with SELECT to identify the unique values.
- SQL DISTINCT looks through the list of values and identifies the unique values from the list.

SQL Select Distinct Syntax

- SELECT DISTINCT column FROM table_name;

Let us consider the following Customer Table to understand SQL DISTINCT query.

CustomerId	CustomerName	CustomerAge	CustomerGender
1	John	31	M
2	Amit	25	M
3	Annie	35	F

Query:

SELECT DISTINCT CutomerGender FROM Customer;

CustomerGender
M
F

SQL Comparison Operators

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL Logical Operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

SQL LIKE

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

SQL LIKE Examples

- The following SQL statement selects all customers with a CustomerName starting with "a":
- ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE 'a%’;
  ```
- The following SQL statement selects all customers with a CustomerName ending with "a":
- ```
  SELECT * FROM Customers
  WHERE CustomerName LIKE '%a';
  ```

SQL AND, OR and NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition:
- The AND operator displays a record if all the conditions separated by AND are TRUE.
- The OR operator displays a record if any of the conditions separated by OR is TRUE.
- The NOT operator displays a record if the condition(s) is NOT TRUE

AND & OR & NOT Syntax

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition1 AND condition2 AND condition3 ...;
  ```

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE condition1 OR condition2 OR condition3 ...;
  ```

- ```
  SELECT column1, column2, ...
  FROM table_name
  WHERE NOT condition;
  ```


  ```

SQL Aliases

- SQL aliases are used to give a table, or a column in a table, a temporary name.
- There are cases when the column name or the table name that is existing in the database is not so human readable. We can use SQL ALIAS feature to assign a new name to the table or columns in our query.
- SQL ALIAS is used for temporary naming of table or column of a table for making it more readable.

Alias Column & Table Syntax

Alias Column Syntax

- `SELECT column_name AS alias_name`
  `FROM table_name;`
- `SELECT name_of_table_or_column AS name_of_alias`

Alias Table Syntax

- `SELECT column_name(s)`
  `FROM table_name AS alias_name;`

SQL CREATE TABLE Statement

- The CREATE TABLE statement is used to create a new table in a database.

Syntax,

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

Example,

```
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

PersonID	LastName	FirstName	Address	City

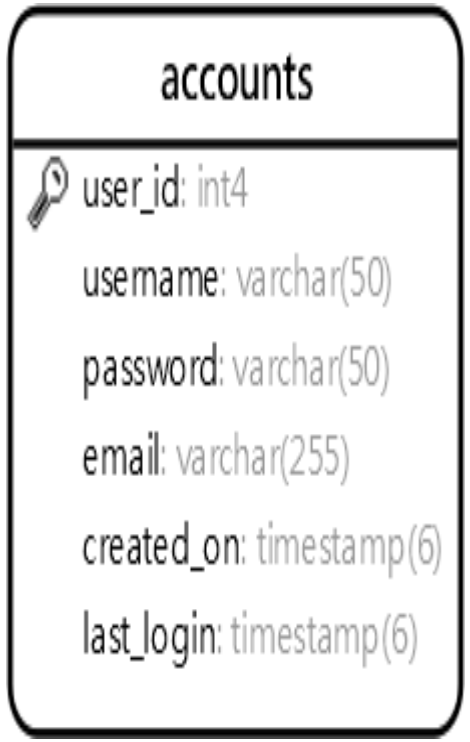
PostgreSQL CREATE TABLE syntax

- CREATE TABLE [IF NOT EXISTS] table_name (
- column1 datatype(length) column_contraint,
- column2 datatype(length) column_contraint,
- column3 datatype(length) column_contraint,
- table_constraints
-);

PostgreSQL

- PostgreSQL includes the following column constraints:
- NOT NULL – Ensures that values in a column cannot be NULL.
- UNIQUE – Ensures the values in a column unique across the rows within the same table.
- PRIMARY KEY – A primary key column uniquely identify rows in a table. A table can have one and only one primary key. The primary key constraint allows you to define the primary key of a table.
- CHECK – A CHECK constraint ensures the data must satisfy a boolean expression.
- FOREIGN KEY – Ensures values in a column or a group of columns from a table exists in a column or group of columns in another table. Unlike the primary key, a table can have many foreign keys.

- CREATE TABLE accounts (
  user_id serial PRIMARY KEY,
  username VARCHAR (50) UNIQUE NOT NULL,
  password VARCHAR (50) NOT NULL,
  email VARCHAR (255) UNIQUE NOT NULL,
  created_on TIMESTAMP NOT NULL,
  last_login TIMESTAMP
);



Example,

- CREATE TABLE cars(make varchar(10))

```
INSERT INTO cars VALUES (‘HONDA’);
INSERT INTO cars VALUES (‘BMW’);
INSERT INTO cars VALUES (‘BMW’);
INSERT INTO cars VALUES (‘BMW’);
INSERT INTO cars VALUES (‘AUDI’);
INSERT INTO cars VALUES (‘AUDI’);
.
.
```

SUBSTRING, REPLACE, POSITION

- `SELECT SUBSTRING ('This is test data' FROM 1 FOR 4) test_data_extracted`

`Output -> This`

- `SELECT department REPLACE(department,'Onur','Jhon')`
- we've changed the person of Onur in the Department table to John.

- `SELECT POSTION('@' IN email)`
- @ we are examining what position it is in.

PostgreSQL Joins

- Various kinds of PostgreSQL joins including inner join, left join, right join, and full outer join.
- PostgreSQL join is used to combine columns from one (self-join) or more tables based on the values of the common columns between related tables. The common columns are typically the primary key columns of the first table and foreign key columns of the second table.
- PostgreSQL supports inner join, left join, right join, full out join, cross join, natural join, and a special kind of join called self-join.

PostgreSQL Joins

SELECT * FROM a
INNER JOIN b ON a.key = b.key

SELECT * FROM a
LEFT JOIN b ON a.key = b.key

SELECT * FROM a
RIGHT JOIN b ON a.key = b.key

SELECT * FROM a
LEFT JOIN b ON a.key = b.key
WHERE b.key IS NULL

SELECT * FROM a
RIGHT JOIN b ON a.key = b.key
WHERE a.key IS NULL

SELECT * FROM a
FULL JOIN b ON a.key = b.key

SELECT * FROM a
FULL JOIN b ON a.key = b.key
WHERE a.key IS NULL OR b.key IS NULL

Setting up sample tables,

```sql
CREATE TABLE basket_a (
    a INT PRIMARY KEY,
    fruit_a VARCHAR (100) NOT NULL
);

CREATE TABLE basket_b (
    b INT PRIMARY KEY,
    fruit_b VARCHAR (100) NOT NULL
);

INSERT INTO basket_a (a, fruit_a)
VALUES
    (1, 'Apple'),
    (2, 'Orange'),
    (3, 'Banana'),
    (4, 'Cucumber');

INSERT INTO basket_b (b, fruit_b)
VALUES
    (1, 'Orange'),
    (2, 'Apple'),
    (3, 'Watermelon'),
    (4, 'Pear');
```

Suppose you have two tables called basket_a and basket_b that store fruits:

- The tables have some common fruits such as apple and orange.
- The following statement returns data from the basket_a table:

	a integer	fruit_a character varying (100)
1	1	Apple
2	2	Orange
3	3	Banana
4	4	Cucumber

- And the following statement returns data from the basket_b table:

	b integer	fruit_b character varying (100)
1	1	Orange
2	2	Apple
3	3	Watermelon
4	4	Pear

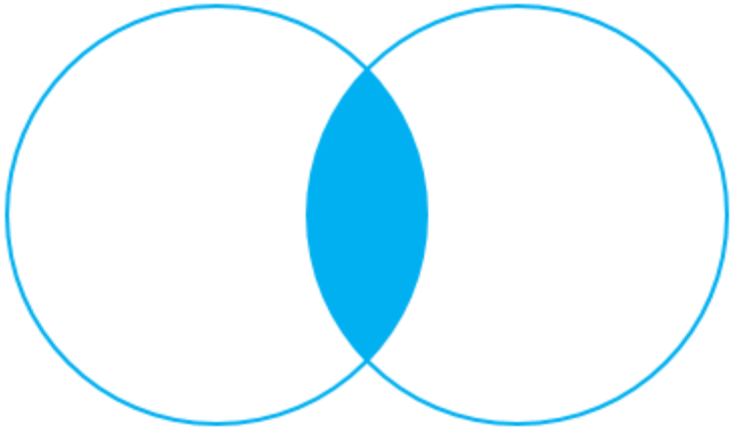
PostgreSQL inner join

- The following statement joins the first table (basket_a) with the second table (basket_b) by matching the values in the fruit_a and fruit_b columns:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
INNER JOIN basket_b
    ON fruit_a = fruit_b;
```

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange

- The following Venn diagram illustrates the inner join:



INNER JOIN

PostgreSQL left join

- The following statement uses the left join clause to join the basket_a table with the basket_b table. In the left join context, the first table is called the left table and the second table is called the right table.

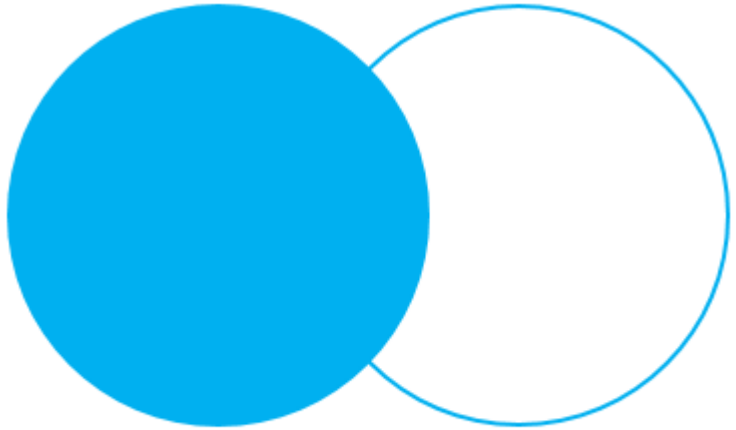
```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
   ON fruit_a = fruit_b;
```

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]

- The left join starts selecting data from the left table. It compares values in the fruit_a column with the values in the fruit_b column in the basket_b table.

- If these values are equal, the left join creates a new row that contains columns of both tables and adds this new row to the result set. (see the row #1 and #2 in the result set).

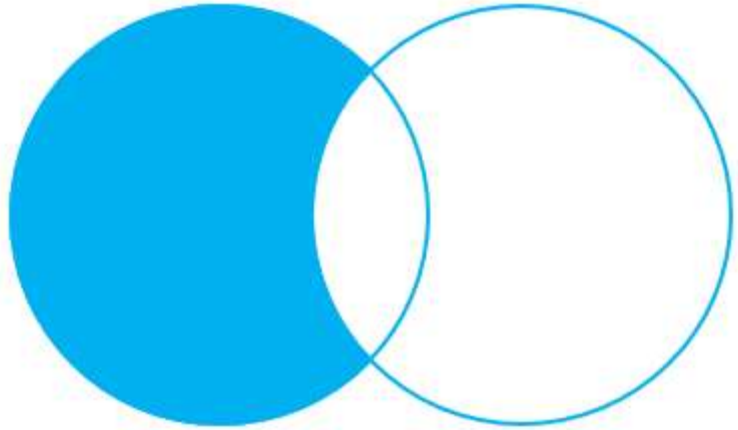
- In case the values do not equal, the left join also creates a new row that contains columns from both tables and adds it to the result set. However, it fills the columns of the right table (basket_b) with null. (see the row #3 and #4 in the result set).



LEFT OUTER JOIN

To select rows from the left table that do not have matching rows in the right table, you use the left join with a WHERE clause. For example:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
LEFT JOIN basket_b
    ON fruit_a = fruit_b
WHERE b IS NULL;
```



LEFT OUTER JOIN – only rows from the left table

The output is:

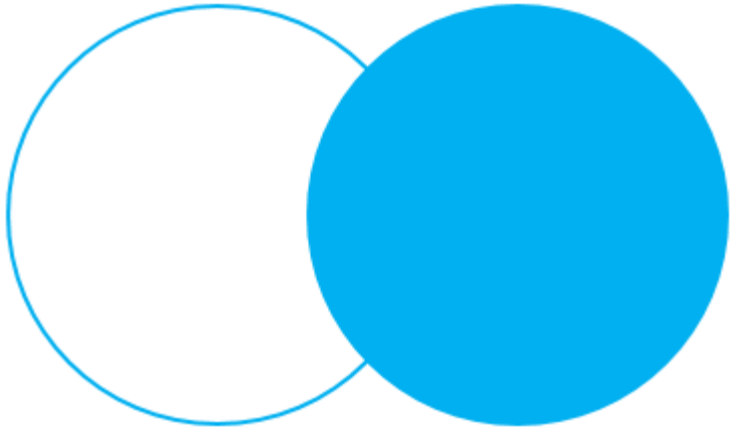
	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	3	Banana	[null]	[null]
2	4	Cucumber	[null]	[null]

PostgreSQL right join

- The right join is a reversed version of the left join. The right join starts selecting data from the right table. It compares each value in the fruit_b column of every row in the right table with each value in the fruit_a column of every row in the fruit_a table.
- If these values are equal, the right join creates a new row that contains columns from both tables.
- In case these values are not equal, the right join also creates a new row that contains columns from both tables. However, it fills the columns in the left table with NULL.

The following statement uses the right join to join the basket_a table with the basket_b table:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
RIGHT JOIN basket_b ON fruit_a = fruit_b;
```



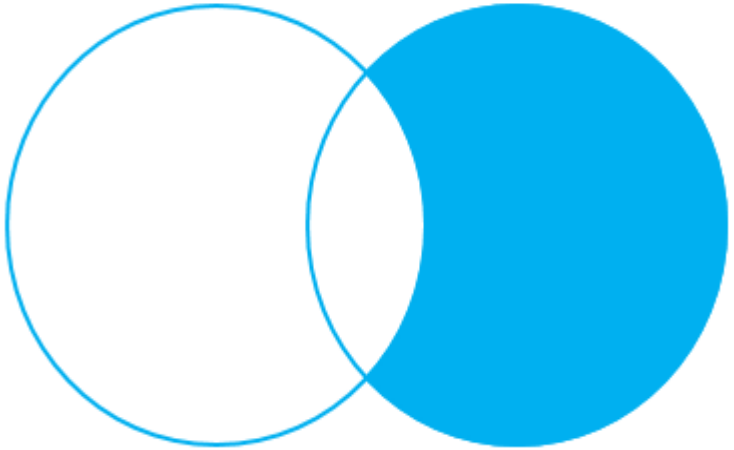
RIGHT OUTER JOIN

Here is the output:

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	2	Orange	1	Orange
2	1	Apple	2	Apple
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear

Similarly, you can get rows from the right table that do not have matching rows from the left table by adding a WHERE clause as follows:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
RIGHT JOIN basket_b
   ON fruit_a = fruit_b
WHERE a IS NULL;
```



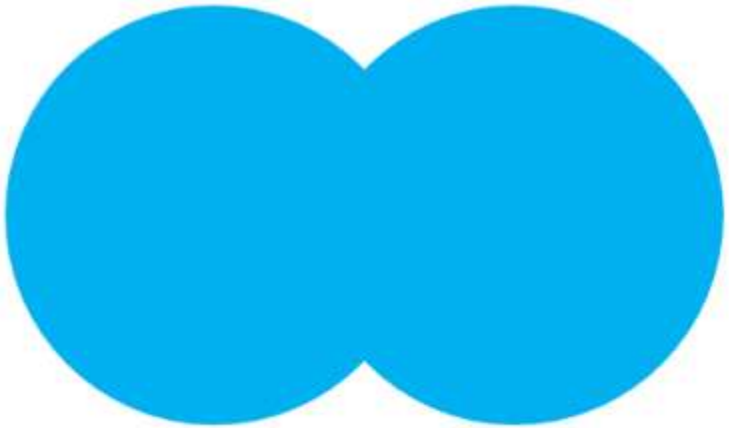
RIGHT OUTER JOIN – only rows from the right table

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	[null]	[null]	3	Watermelon
2	[null]	[null]	4	Pear

PostgreSQL full outer join

- The full outer join or full join returns a result set that contains all rows from both left and right tables, with the matching rows from both sides if available. In case there is no match, the columns of the table will be filled with NULL.

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
FULL OUTER JOIN basket_b
    ON fruit_a = fruit_b;
```



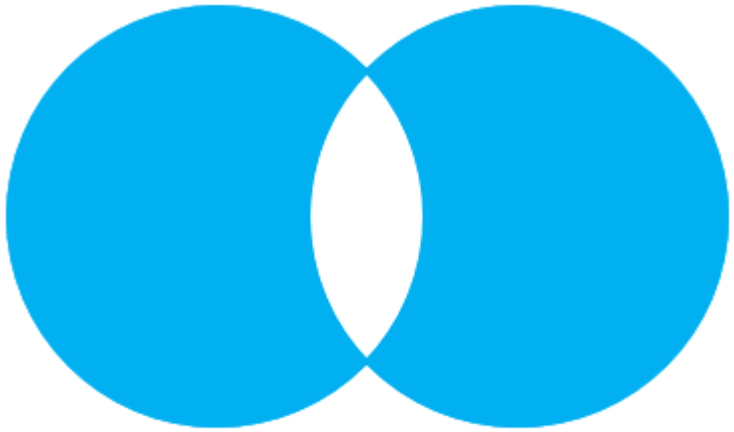
FULL OUTER JOIN

Output:

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]
5	[null]	[null]	3	Watermelon
6	[null]	[null]	4	Pear

To return rows in a table that do not have matching rows in the other, you use the full join with a WHERE clause like this:

```
SELECT
    a,
    fruit_a,
    b,
    fruit_b
FROM
    basket_a
FULL JOIN basket_b
   ON fruit_a = fruit_b
WHERE a IS NULL OR b IS NULL;
```



FULL OUTER JOIN – only rows unique to both tables

Here is the result:

	a integer	fruit_a character varying (100)	b integer	fruit_b character varying (100)
1	3	Banana	[null]	[null]
2	4	Cucumber	[null]	[null]
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear