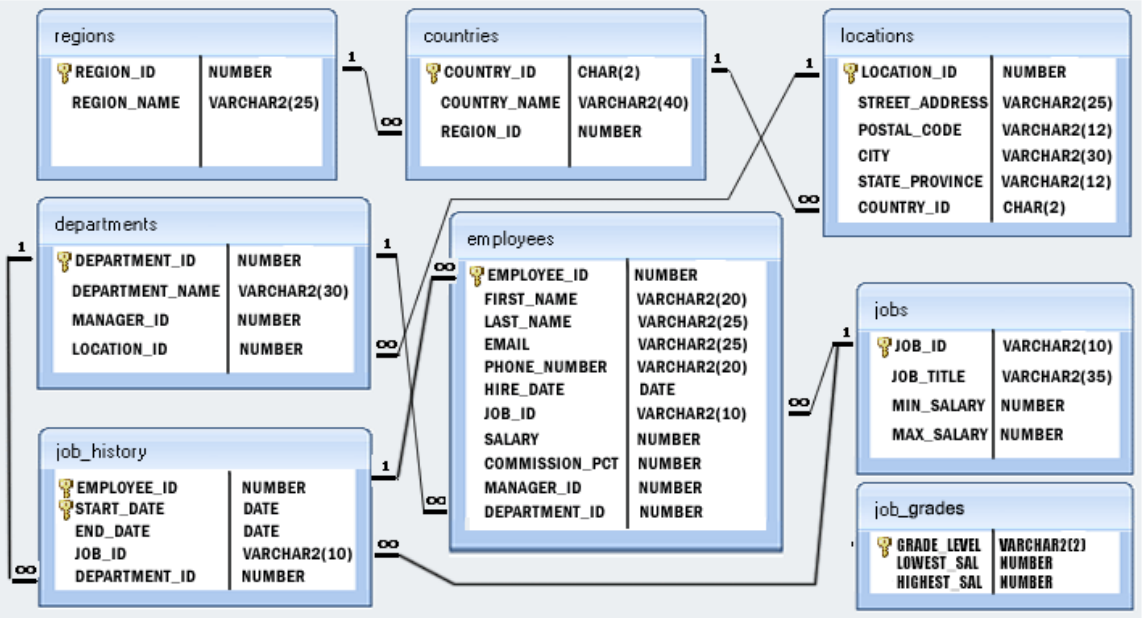
SQL

What is DataBase Management System (DBMS)?

- In simple words, **Data** can be facts related to any object in consideration. For example, your name. Age, height, etc.
- **Database** is a systematic collection of Data. Databases support storage and manipulation of data. They make data management easy. For example, online phone directory, or Facebook keeping all members data.
- Database Management System **(DBMS)** is a collection of programs which enables its users to access the database, to manipulate data, report / representations of data. It also helps to control access to the database.



RDBMS (Relational Database Management System)

- Data is organized into tables that are related to each other.
- Tables are related to each other using Primary and foreign keys
- What type of database system do you have expertise with? RDBMS, such as MySQL and Oracle.
- All RDMS use **SQL** language.
- Non-Relational Databases are all in **Key** & **Value** format.

SQL?

- Structured Query Language (SQL) used for managing and manipulating data in DB.
- Provide statements for a variety of tasks
  - Querying data
  - Inserting, updating ,deleting rows in a table
  - Creating, replacing, altering, and dropping objects
  - Controlling access to the database and its objects
  - Database consistency and integrity

What are the components of SQL statements?

SQL is a hybrid language since includes 4 different languages

- **DML (Data Manipulation Language)**
  - DML statements affect records in a table. These are basic operations we perform on **data** such as **selecting** a few records from a table, **inserting** new records, **deleting** unnecessary records, and **updating/modifying** existing records.
  - SELECT, INSERT, UPDATE, DELETE
- **DDL (Data Definition Language)**
  - DDL statements are used to alter/**modify** a **database** or **table structure** and schema. These statements handle the design and storage of database objects.
  - CREATE, DROP, TRUNCATE, ALTER
- **DCL (Data Control Language)**
  - DCL statements control the **level of access** that users have on database objects.
- **TCL (Transaction Control Language)**
  - TCL statements allow you to control and manage transactions to maintain the integrity of data within SQL statements.

What is Query in SQL?

- Queries are **requests** made to the database management system for specific information.
- A query is a set of instructions given to the RDBMS (written in SQL) that tell the RDBMS what information you want it to retrieve for you.
  - Tons of data in DB.
  - Often hidden in a complex schema.
  - GOAL is to get the data you want with a query.
    Sample: *selecting all employees named Steven*

```
SELECT    *
FROM      employees
WHERE     first_name='Steven';
```

What are the Data Types in a query?

- Int & Integer *(whole numbers)*
- Decimal
- Varchar: *string of text*
- Boolean:
- Date: 'YYYY-MM-DD'
- Timestamp: 'YYYY-MM-DD HH:MM:SS'

STATEMENTS and KEYWORDS in SQL:

SELECT statement is used to select data from a database. The data returned is stored in a result table, called the **result set**.

```
SELECT * FROM student; (This is to select all (*) information from the student table.)
```

DISTINCT keyword can be used to return only distinct (different/unique) values. The Distinct keyword will remove all duplicate values.

WHERE clause appears right after the FROM clause of the SELECT statement.

The conditions are used to **filter** the rows returned from the SELECT statement.

```
SELECT column_1, column_2.. Column_n
FROM table_name
WHERE conditions; (Here WHERE applies filter to the result)
```

COUNT function returns the number of input rows that match a specific condition of a query.

- ➔ Similar to COUNT(*) function, the COUNT(column) function returns the number of rows returned by a SELECT clause. *How many departments do we have ?*
- ➔ We can also use COUNT with DISTINCT. However, it does not consider NULL values in the column.

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

ORDER BY clause allows you to sort the rows returned from the SELECT statement in ascending or descending order based on criteria specified.

- ➔ If we sort the result set by **multiple columns,** use a comma to separate between two columns.
- ➔ ASC : to sort the result set in ascending order (A-Z, 0-9)
- ➔ DESC : to sort the result set in descending order (Z-A, 9-0)

```
SELECT column_1, column_2.. Column_n
FROM table_name
ORDER BY column_1 ASC/DESC;
```
(column_1 will be ordered based on what we choose ASC/DESC)

BETWEEN operator is used to match a value against a range of values.

- ➔ If the value is greater than or equal to the low value and less than or equal to the high value, the expression returns true, or vice versa.
- ➔ We can rewrite the BETWEEN operator by using the greater than or equal (>=) or less than or equal (<=) operators as the following statement:

```
SELECT *
FROM employees
WHERE salary BETWEEN 60000 AND 150000;
```

IN operator is used with the WHERE clause to check if a value matches any value **in a list of values**

- ➔ The list of values is not limited to a list of numbers or strings but also a result set of a SELECT statement as shown in the following query:

```
Value IN(SELECT value FROM tbl_name)
```

- ➔ Just like with BETWEEN, you can use **NOT** to adjust an **IN** statement (NOT IN)

```
SELECT employee_id, first_name, last_name, salary
FROM employees
WHERE employee_id IN( 101, 105, 117, 123, 125)
ORDER BY salary DESC;
```

LIKE is used in the cases you cannot remember the exact match/search keyword but can provide partial text

- ➔ Suppose the store manager asks you to find a customer that he does not remember the name exactly.
- ➔ He just remembers that customer's first name begins with something like "Jen"
- ➔ We may find the customer in the customer table by looking at the first name column to see if there is any value that begins with Jen.
- ➔ The query returns rows whose values in the first name column **begin with Jen** and may be followed by **any sequence of characters**.
- ➔ This technique is called **pattern matching**.

```
SELECT first name, last name
FROM employees
WHERE first name LIKE 'JEN'%        % takes the whatever after 'Jen' with single quotes ''
```

AGGREGATE FUNCTIONS:

- ❖ **MIN** performs the action for multiple rows at once and returns **a single result**. It will check all of the rows and show the **minimum one**.

```
SELECT MIN (salary)    salary is the column name
FROM employees;
```

- ❖ **MAX** checks all the rows and shows the **maximum** one.

```
SELECT MAX (salary)
FROM employees;
```

- ❖ **AVG** adds all rows and gets the **average** value.

```
SELECT AVG (salary)
FROM employees;
```

- ❖ **ROUND** the result with the given decimal.

```
SELECT ROUND (AVG(salary), 2)        2 is to define the number of decimals we want to see
FROM employees;
```

- ❖ **SUM** adds all rows and shows the **SUM** value.

```
SELECT SUM (salary)
FROM employees;
```

- ❖ **GROUP BY** clause divides the rows returned from the SELECT statement **into groups**. For each group, you can apply an aggregate function, for example:
  - ➢ calculating the sum of items
  - ➢ count the number of items in the groups.

```
SELECT column_1 aggregate function (column_2)
FROM table_name
GROUP BY column_1;
```

- **HAVING** clause is used in conjunction with the **GROUP BY** clause to filter group rows that do not satisfy a specified condition.
  Difference between **HAVING** and **WHERE**:
  - The **HAVING** statement sets the condition for group rows created by the GROUP BY clause **after the GROUP BY clause applies** while **WHERE** clause sets the condition for individual rows before GROUP BY clause applies.

```
SELECT column_1 aggregate function (column_2)
FROM table_name
GROUP BY column_1
HAVING condition;
```

CREATE A TABLE	
CREATE TABLE table name (column name TYPE column constraint);	CREATE TABLE students(student_id serial primary key, first_name varchar(30) not null, last_name varchar(30) not null, phone_number bigint null);
INSERT: *After creating a new table, we insert new rows into the table.*	
INSERT INTO table(column1,column2) VALUES (value1,value2,..), (value1,value2,..);	INSERT INTO students(first_name, last_name, phone_number) VALUES ('Mike', 'Smith', 9739739739);
UPDATE: *To change the values of the columns.*	
UPDATE table_name SET column1 = value1, column2 = value2 , ... WHERE condition;	UPDATE students SET first_name = 'Jamal' WHERE student_id =1;
DELETE: *To delete rows in a table. (If you omit the WHERE clause, all rows in the table are deleted)*	
DELETE FROM table_name WHERE condition;	DELETE FROM students WHERE student_id = 1;
ALTER: *To change existing table structure, you use ALTER TABLE statement.*	
ALTER TABLE table_name action;	ALTER TABLE students RENAME TO st; ALTER TABLE st RENAME COLUMN phone_number TO phone; ALTER TABLE st DROP COLUMN phone;
DROP: *To remove existing table from the database, you use the DROP TABLE statement as shown following:*	
DROP TABLE table_name;	DROP TABLE st;
TRUNCATE: *To Remove all data from the table* ***but not delete the table****. Lastly we can DROP TABLE.*	
TRUNCATE TABLE table_name;	TRUNCATE TABLE st;

CREATE A TABLE

1. First, we specify the name of the new table and alter the CREATE TABLE clause.
2. Next, we list the column name, its data type, and column constraint.
3. PostgreSQL column Constraints;
    a. NOT NULL the value of the column cannot be NULL
    b. UNIQUE the value of the column must be unique across the whole table
    c. PRIMARY KEY it is the combination of NOT NULL and UNIQUE constraints

Primary Key:

- Unique value to identify other data. It can be any type (int, string). It's important to find a unique primary key for identification.

Surrogate Key:

- It is a system generated value with no business meaning that it is used to uniquely identify a record in a table. It can be made up of one or multiple columns.

Natural Key:

- It is a column or set of columns that already exists in a table and uniquely identifies a record in the table. Since they are attributes of the entity, they obviously have business meaning. For example, SSN, State_ID, Student_ID, etc.

Foregn Key:

- It is used to **link** two tables together. It is sometimes called a referencing key. It is a column or combination of columns of which values match a **Primary Key** in a different table.

Composite Key:

- It is a combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined, uniqueness is guaranteed but when it is taken individually it does not guarantee uniqueness.

SQL Practice Website:

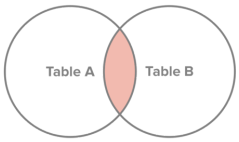
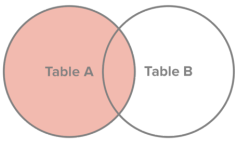
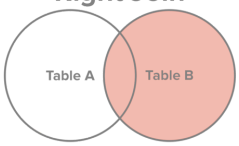
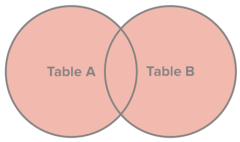
https://livesql.oracle.com/apex/f?p=590:1000

JOINS and SET Operators

SQL JOIN is a SQL instruction to combine data from two sets of data (i.e. tables). SQL is a special purpose programming language designed for managing information in a relational database management system (RDMS). The word **RELATIONAL** here is key! It specifies that the database management system is organized in such a way that there are clear relations defined between different sets of data.

JOIN Types:

- There are four basic types of SQL joins: **inner**, **left**, **right**, and **full**.
- Let's say we have two sets of data in our relational database: table A and table B, with some sort of relation specified by **primary** and **foreign keys**. The result of joining these tables together can be visually represented by the following diagrams:

Inner Join	Left Join	Right Join	Full Join
			
Select all records from Table A and Table B, where the join condition is met.	Select all records from Table A, along with records from Table B for which the join condition is met (if at all).	Select all records from Table B, along with records from Table A for which the join condition is met (if at all).	Select all records from Table A and Table B, regardless of whether the join condition is met or not.

JOIN WITH SAMPLES:

CUSTOMER TABLE

	customer_id integer	first_name character vary	last_name character varyi	address_id integer
1	1	Mary	Smith	5
2	2	Patricia	Johnson	[null]
3	3	Linda	Williams	7
4	4	Barbara	Jones	8
5	5	Elizabeth	Brown	[null]

ADDRESS TABLE

	address_id integer	address character varying (50)	phone integer
1	5	1913 Hanoi Way	28303384
2	7	692 Joliet Street	44847719
3	8	1566 Inegl Manor	70581400
4	10	1795 Santiago	86045262
5	11	900 Santiago	16571220

INNER JOIN:
It produces only the set of records that match in both Customer and Address Tables. It will return the records **at the intersection** of the two tables.

	customer_id integer	first_name character va	last_name character var	address character varying (5	phone integer
1	1	Mary	Smith	1913 Hanoi Way	28303384
2	3	Linda	Williams	692 Joliet Street	44847719
3	4	Barbara	Jones	1566 Inegl Manor	70581400

```
SELECT customer_id,first_name,last_name,address,phone
FROM customer
INNER JOIN address ON customer.address_id = address.address_id;
```

LEFT JOIN:
It produces a complete set of records from Customer Table, with matching records (where available) in Address Table. If there is no match, the right side will contain null.

	customer_id integer	first_name character var	last_name character va	address character varying (5	phone integer
1	1	Mary	Smith	1913 Hanoi Way	28303384
2	2	Patricia	Johnson	[null]	[null]
3	3	Linda	Williams	692 Joliet Street	44847719
4	4	Barbara	Jones	1566 Inegl Manor	70581400
5	5	Elizabeth	Brown	[null]	[null]

```
SELECT customer_id,first_name,last_name,address,phone
FROM customer
LEFT JOIN address ON customer.address_id = address.address_id;
```

RIGHT JOIN:
It produces a complete set of records from Address Table, with matching records (where available) in Customer Table. If there is no match, the right side will contain null.

	customer_id integer	first_name character va	last_name character va	address character varying (5	phone integer
1	1	Mary	Smith	1913 Hanoi Way	28303384
2	3	Linda	Williams	692 Joliet Street	44847719
3	4	Barbara	Jones	1566 Inegl Manor	70581400
4	[null]	[null]	[null]	900 Santiago	16571220
5	[null]	[null]	[null]	1795 Santiago	86045262

```
SELECT customer_id,first_name,last_name,address,phone
FROM customer
RIGHT JOIN address ON customer.address_id = address.address_id;
```

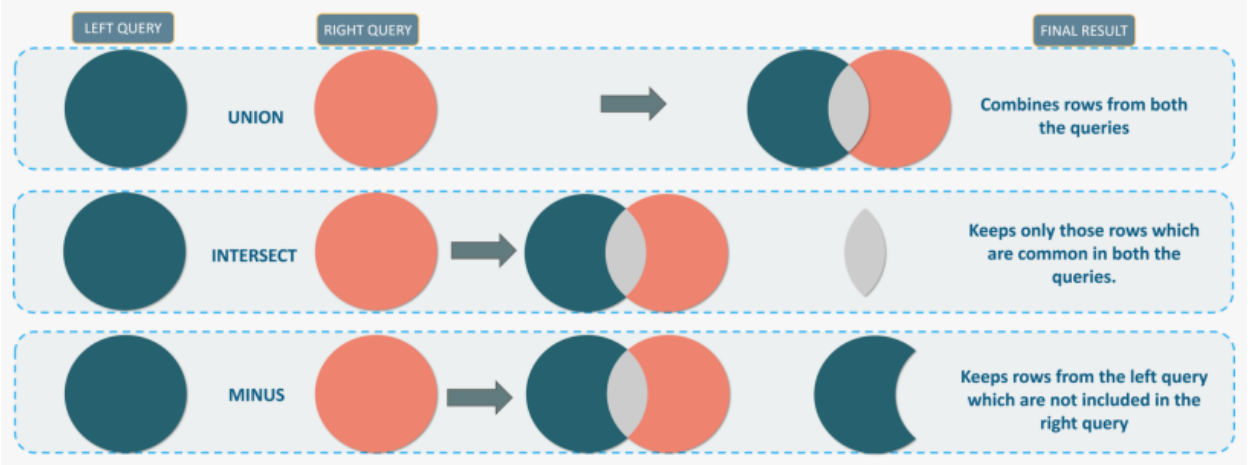
FULL JOIN:
For a list of all records from both tables, we use full join.

	customer_id integer	first_name character var	last_name character vary	address character varying (50	phone integer
1	1	Mary	Smith	1913 Hanoi Way	28303384
2	2	Patricia	Johnson	[null]	[null]
3	3	Linda	Williams	692 Joliet Street	44847719
4	4	Barbara	Jones	1566 Inegl Manor	70581400
5	5	Elizabeth	Brown	[null]	[null]
6	[null]	[null]	[null]	900 Santiago	16571220
7	[null]	[null]	[null]	1795 Santiago	86045262

```
SELECT customer_id,first_name,last_name,address,phone
FROM customer
FULL JOIN address ON customer.address_id = address.address_id;
```

SET Operators:

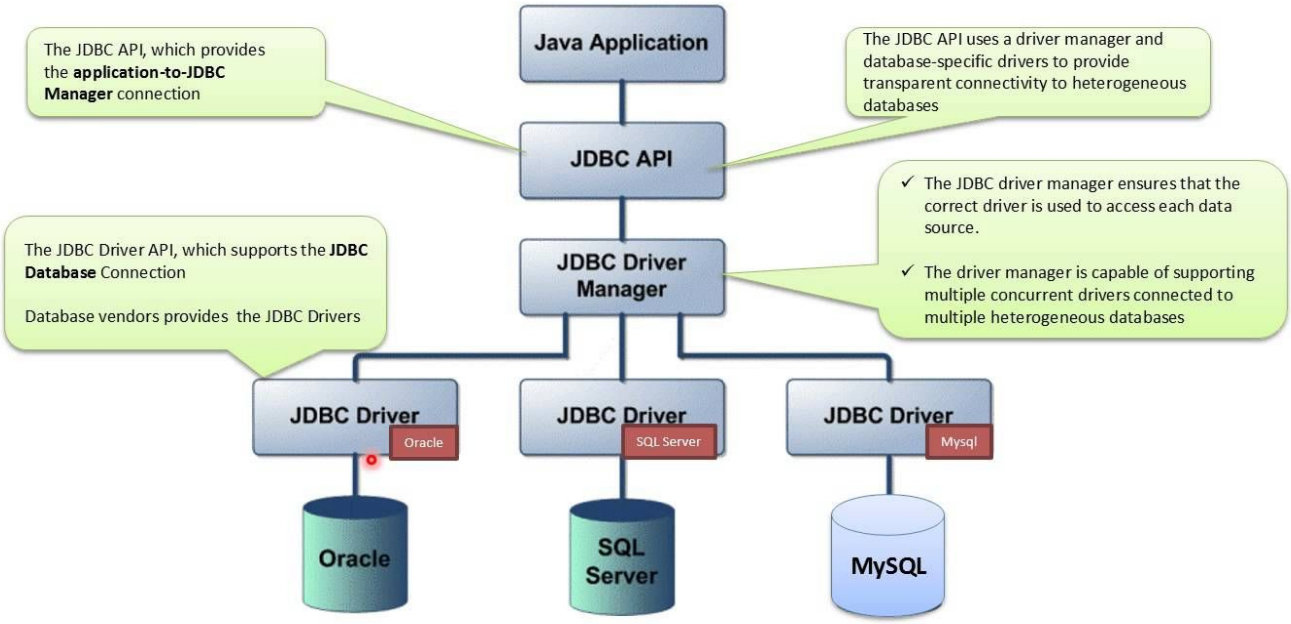
- SQL supports a few SET operations which can be performed on the table data. SET operators are used to join **the results of two (or more) SELECT statements**. There will be more than one SELECT statement BUT only one **combined table** will be generated.
- The **result sets** of all queries must have the same number of columns.
- In every result set, the **data type** of each column must match the data type of its corresponding column in the first result set.
- The column names or aliases must be found out by the first select statement.



UNION: Combine two or more result sets into a single set, without duplicates. **UNION ALL:** Combine two or more result sets into a single set, WITH duplicates.	SELECT column1,column2... FROM table_name; **UNION** SELECT column1,column2... FROM table_name;
INTERSECT: Takes the data from both result sets which are in common with no duplicates and data arranged in sorted order.	SELECT SALARY FROM employees WHERE DEPARTMENT_ID = 10 **INTERSECT** SELECT SALARY FROM employees WHERE DEPARTMENT_ID = 20
MINUS (EXCEPT): Displays the rows which are present in the first query but absent in the second query, with no duplicates and data arranged in ascending order by default.	SELECT column1 FROM table_name1; **MINUS** SELECT column1 FROM table_name2;
Table A: 10,9,8,7,6,5 --- Table B: 7,8,9 10,9,8,7,6,5 **UNION** 7,8,9 → **5,6,7,8,9,10** 10,9,8,7,6,5 **UNION ALL** 7,8,9 → **10,9,8,7,6,57,8,9**	**Table A: 10,9,8,7,6,5 --- Table B: 7,8,9** 10,9,8,7,6,5 **INTERSECT** 7,8,9 → **7,8,9** 10,9,8,7,6,5 **MINUS** 7,8,9 → **10,6,5**

JDBC

Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases.



JDBC Maven Dependency:

```
<dependency>
  <groupId>oracle</groupId>
  <artifactId>ojdbc6</artifactId>
  <version>11.2.0.3</version>
</dependency>
```

3 Important Classes/Steps in JDBC

1. **Connection** helps the java project to connect to the database.
2. **Statement** helps to write and execute SQL queries.
3. **ResultSet** is a Data Structure where we can store the data coming from the database.

Connection String in Detail:

String connection_str =
"jdbc:oracle:thin:@ec2-54-160-198-253.compute-1.amazonaws.com:1521:xe";

jdbc : protocol
oracle : sub-protocol to define database vendors (This might be SQL, MySQL, etc.)
thin : oracle driver name, thin driver
(depends on database itself, for MySQL you don't need this)
host : this is the address of the database server YOUR IP!
Where is your database server? We created our database in the cloud.
port : the door we will get into the database
xe : database service name - uniquely identify our database we're trying to reach

JDBC Set Up:

1. Creating Connection object based on the connection string.

   Connection connection = **DriverManager.*getConnection***(connection_str, db_userName, db_passWord);

2. Once we have a connection object, now we can create a **statement object** from it.

   Statement statement = connection.createStatement();

3. Now, we can use the Statement object to run a query and store the result in a **ResultSet** object.

   ResultSet resultSet = statement.executeQuery(**"SELECT * FROM COUNTRIES"**);

JDBC Methods and Navigation:

Navigation:

- If we want to move backward to access the previous row, We need to set-up the resultSet with "ResultSet.***TYPE_SCROLL_INSENSITIVE*** and ResultSet.***CONCUR_UPDATABLE***".
- They will give us the most updated data and also will help to move backward and forward on the table.
- This is a one time set-up, you just need to know what it is but no detail is required for the interviews.

```
Statement statement = connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
                                                 ResultSet.CONCUR_UPDATABLE);
```

Methods:

- resultSet.next(); : goes to the NEXT row where the cursor is.
- resultSet.previous(); : goes to the previous row where the cursor stopped.
- resultSet.absolute(2); : moves the cursor to a row that you specified
- resultSet.first(); : goes to the first row regardless of where the cursor is
- resultSet.last(); : goes to the last row regardless of where the cursor is
- resultSet.beforeFirst(); : moves the cursor to the location right before first row
- resultSet.afterLast(); : move the cursor to the location right after last row

1. resultSet.closed(); 2. statement.close(); 3. connection.close();	Connection, statement, and ResultSet are considered as resources. Once you finish working with them, you need to close them. It's like closing your book after you finish reading it. **The order is important!**

What is MetaData?

- MetaData is data about data.
- Metadata Programming is useful to know the capabilities, limitations and facilities of underlying database software and its resources
- DBC metadata programming Supports :
  - DatabaseMetadata
  - ResultSetMetaData

`DatabaseMetaData metaData = connection.getMetaData();`
 	
`System.out.println("User: " + metaData.getUserName());`
`System.out.println("Database Type: "+metaData.getDatabaseProductName());`
`System.out.println("Database Version: "+metaData.getDatabaseProductVersion());`
`System.out.println("Driver Name: "+metaData.getDriverName());`
`System.out.println("Driver Version: "+metaData.getDriverVersion());`
	`User: HR`
	`Database Type: Oracle`
	`Database Version: Oracle Database 11g Express Edition Release 11.2.0.2.0 - Production`
	`Driver Name: Oracle JDBC driver`
	`Driver Version: 12.1.0.1.0`

`ResultSet result = statement.executeQuery("Select * from employees");`
 	
`ResultSetMetaData rsMetadata = result.getMetaData();`
 	
`System.out.println("Columns count: " + rsMetadata.getColumnCount());`
`System.out.println("Column Name: " + rsMetadata.getColumnName(1));`
	`Columns count: 11`
	`Column Name: EMPLOYEE_ID`

SQL Interview Questions:

Do you know SQL?

- Yes, I am very comfortable with writing SQL Queries and DDL and DML commands.
- Currently working with an Oracle database that is running in AMAZON CLOUD SERVER.
- DDL (Data definition language) : CREATE , ALTER, DROP, TRUNCATE
- DML(Data manipulation language): SELECT, DELETE, INSERT, UPDATE

What kind of Database Testing Are You Doing?

- I am mostly doing Database validations.
- I make changes or insert data (create loan) in the front end and validate in the database. Data in the front end matches the DB.
- I also make changes using RESTapi and verify that changes are successful in Database as well.

Do you have any experience with SQL?

- Yes, I have worked with relational databases and I am very comfortable with DDL and DML commands.

Have You Done Any Backend/DataBase testing?

- Yes, I have lots of experience working with databases.
- And I am very comfortable with writing SQL queries.
- I have experience with working on Relational Databases like Oracle, MySQL, SQL Server

Have you worked with non-relational databases?

- I don't have hands on experience, but I know that it is like JSON format
- Database and I have good experience with working with JSON files.
- And I am a quick learner

How can you find 3 HIGHEST PAID employees?

```
SELECT salary, first_name, last_name
FROM employees
ORDER BY salary DESC LIMIT 3;
```

How can you find DUPLICATE NAMES in employees?

```
SELECT first_name, COUNT(first_name)
FROM employees
GROUP BY first_name
HAVING (COUNT(first_name>1));
```

How can you find employees whose salaries are below the average?

```
SELECT first_name, salary
FROM employees
WHERE <= (SELECT AVG (salary) FROM employees);
```

How can you find MAXIMUM salaries in EACH DEPARTMENT?

```
SELECT first_name, MAX(salary)
FROM department d LEFT JOIN employee e ON (d.department_id = e.department_id)
GROUP BY department_id;
```

How can you find the LOWEST Salaries?

```
SELECT first_name, last_name, salary, job_id
FROM employees
WHERE salary =(SELECT MIN (salary) FROM employees);
```

How can you find the SECOND HIGHEST Salary?

```
SELECT MAX (salary)
FROM employees
WHERE salary NOT IN (SELECT MAX(salary) FROM employees);
or
WHERE salary < (SELECT MAX(salary) FROM employees);
```

Get all Employees FIRST_NAME contains 'a' case insensitive

```
SELECT first_name, last_name
FROM employees
WHERE first_name LIKE '%A%' or  first_name LIKE '%a%';
```

- LIKE A% starts with A
- LIKE %A ends with A
- LIKE '%A%' contains A

Count How Many First_Names start with 'A'

```
SELECT COUNT (*) AS CNT // this is to name the column header which will give the result
FROM employees
WHERE first_name LIKE '%A%'
```

Get the count of employees with the salary between 10000 and 20000

```
SELECT COUNT (*)
FROM employees
WHERE salary BETWEEN 10000 AND 20000
```