

SQL & Object Classes

Grade 10 → Grade 12 | IEB Information Technology

Revision Study Notes — Prepared for Study Group

SECTION 1 — SQL (Structured Query Language)

1. What is SQL?

SQL is a language used to communicate with a DATABASE. It allows you to store, retrieve, update and delete data in a structured way using tables.

Term	Definition	Analogy
Database	A collection of related tables	The whole filing cabinet
Table	Data organised in rows and columns	A single spreadsheet
Field (Column)	A category of data	A column heading
Record (Row)	One complete entry of data	One row of info
Primary Key	Unique identifier for each record	ID number — no duplicates
Foreign Key	Field that links to another table's primary key	A reference to another table

2. SQL Statement Structure

 **Key:** SQL keywords are written in UPPERCASE. Table and field names are case-sensitive depending on the system.

```
SELECT fields FROM table WHERE condition ORDER BY field;
```

3. Grade 10 — Basic SELECT Queries

A. SELECT All Records

```
SELECT * FROM Students;
-- * means ALL fields
```

B. SELECT Specific Fields

```
SELECT StudentName, Age FROM Students;
```

C. WHERE Clause — Filtering Records

```
SELECT * FROM Students WHERE Age > 16;
SELECT * FROM Students WHERE Grade = 'Grade 10';
```

Operator	Meaning	Example
=	Equal to	WHERE Age = 17
<> or !=	Not equal to	WHERE Grade <> 'Grade 9'
>	Greater than	WHERE Mark > 50
<	Less than	WHERE Mark < 40
>=	Greater than or equal	WHERE Mark >= 50

Operator	Meaning	Example
<=	Less than or equal	WHERE Mark <= 30
BETWEEN	Within a range (inclusive)	WHERE Age BETWEEN 15 AND 18
LIKE	Pattern matching	WHERE Name LIKE 'A%'
IN	Matches a list of values	WHERE Grade IN ('Gr10','Gr11')
IS NULL	Field has no value	WHERE Email IS NULL
IS NOT NULL	Field has a value	WHERE Email IS NOT NULL

D. LIKE Pattern Matching

Pattern	Meaning	Example match
'A%'	Starts with A	Alice, Andrew
'%son'	Ends with son	Jackson, Wilson
'%an%'	Contains 'an' anywhere	Danny, Frank
'_a%'	Second letter is a	Jane, Kate

 **Note:** % = any number of characters. _ = exactly ONE character.

E. AND / OR / NOT

```
SELECT * FROM Students WHERE Age > 15 AND Grade = 'Grade 11';
SELECT * FROM Students WHERE City = 'Joburg' OR City = 'Pretoria';
SELECT * FROM Students WHERE NOT Grade = 'Grade 9';
```

F. ORDER BY — Sorting Results

```
SELECT * FROM Students ORDER BY Surname ASC;
SELECT * FROM Students ORDER BY Mark DESC;
-- ASC = ascending (A-Z, 0-9) | DESC = descending (Z-A, 9-0)
```

4. Grade 11 — Aggregate Functions & GROUP BY

A. Aggregate Functions

Function	What it does	Example
COUNT(*)	Counts number of records	COUNT(*) → 30
COUNT(field)	Counts non-null values in a field	COUNT(Email) → 25
SUM(field)	Adds up all values	SUM(Mark) → 1450
AVG(field)	Calculates the average	AVG(Mark) → 62.5
MAX(field)	Finds the highest value	MAX(Mark) → 98
MIN(field)	Finds the lowest value	MIN(Mark) → 12

```
SELECT COUNT(*) FROM Students WHERE Grade = 'Grade 11';
SELECT AVG(Mark) FROM Results WHERE Subject = 'Maths';
SELECT MAX(Mark), MIN(Mark) FROM Results;
```

B. AS — Aliases (Renaming columns in output)

```
SELECT AVG(Mark) AS AverageMark FROM Results;
SELECT StudentName AS [Full Name] FROM Students;
-- Use square brackets for names with spaces
```

C. GROUP BY — Grouping Records

Used with aggregate functions to group results by a category.

```
SELECT Grade, AVG(Mark) AS Average
FROM Results
GROUP BY Grade;
```

 **Note:** Every field in the SELECT that is NOT an aggregate function MUST appear in GROUP BY.

D. HAVING — Filtering Groups

HAVING is like WHERE but for groups — used AFTER GROUP BY.

```
SELECT Grade, AVG(Mark) AS Average
FROM Results
GROUP BY Grade
HAVING AVG(Mark) > 60;
```

WHERE vs HAVING	WHERE	HAVING
Filters	Individual records	Groups of records
Used with	Any query	GROUP BY only
Position	Before GROUP BY	After GROUP BY
Can use aggregates?	No	Yes

E. DISTINCT — Remove Duplicates

```
SELECT DISTINCT City FROM Students;
-- Only shows each city once, no repeats
```

F. TOP / LIMIT — Restrict Number of Results

```
SELECT TOP 5 * FROM Results ORDER BY Mark DESC;
-- Gets the top 5 highest marks (SQL Server / Access)
```

5. Grade 12 — Joins & Multiple Tables

A JOIN combines rows from two or more tables based on a related field (usually Primary Key + Foreign Key).

A. INNER JOIN — Only Matching Records

Returns records that have matching values in BOTH tables.

```
SELECT Students.StudentName, Results.Mark
FROM Students
INNER JOIN Results ON Students.StudentID = Results.StudentID;
```

✓ **Tip:** INNER JOIN is the most common join in IEB exams. If no match exists in either table, that record is excluded.

B. LEFT JOIN — All from Left + Matches from Right

Returns ALL records from the LEFT table and matching records from the RIGHT table. Non-matches show NULL.

```
SELECT Students.StudentName, Results.Mark
FROM Students
LEFT JOIN Results ON Students.StudentID = Results.StudentID;
```

C. Full JOIN Query Structure

```
SELECT t1.field, t2.field FROM table1 INNER JOIN table2 ON t1.key = t2.key
WHERE ... ORDER BY ...;
```

D. Joining 3 Tables

```
SELECT Students.Name, Subjects.SubjectName, Results.Mark
FROM Students
INNER JOIN Results ON Students.StudentID = Results.StudentID
INNER JOIN Subjects ON Results.SubjectID = Subjects.SubjectID
WHERE Results.Mark > 50
ORDER BY Students.Name ASC;
```

6. Data Manipulation — INSERT, UPDATE, DELETE

INSERT — Add a new record

```
INSERT INTO Students (StudentID, StudentName, Age)
VALUES (101, 'John Smith', 17);
```

UPDATE — Change existing data

```
UPDATE Students
SET Age = 18
WHERE StudentID = 101;
```

⚠ **Note:** Always include WHERE in UPDATE — without it, ALL records will be changed!

DELETE — Remove a record

```
DELETE FROM Students WHERE StudentID = 101;
```

⚠ **Note:** Always include WHERE in DELETE — without it, ALL records will be deleted!

7. SQL — Full Query Writing Order

Clause	Purpose	Order Written	Order Executed
SELECT	Choose fields to display	1st	5th
FROM	Choose the table(s)	2nd	1st
JOIN ... ON	Combine tables	3rd	2nd
WHERE	Filter records	4th	3rd
GROUP BY	Group results	5th	4th
HAVING	Filter groups	6th	5th
ORDER BY	Sort results	7th	6th
TOP / LIMIT	Restrict output	8th	7th

8. SQL Quick Reference — Exam Keywords

Phrase in question	SQL to use
Show all records	SELECT * FROM ...
Show only certain fields	SELECT field1, field2 FROM ...
Filter by condition	WHERE condition
Sort alphabetically A-Z	ORDER BY field ASC
Sort highest to lowest	ORDER BY field DESC
Count how many	COUNT(*)
Find the average	AVG(field)
Find the highest	MAX(field)
Find the lowest	MIN(field)
Total of all values	SUM(field)
Group by category	GROUP BY field
Filter after grouping	HAVING condition
No duplicate values	SELECT DISTINCT
Combine two tables	INNER JOIN ... ON
Records containing 'Smith'	WHERE Name LIKE '%Smith%'
Records starting with 'A'	WHERE Name LIKE 'A%'
Add a new record	INSERT INTO ... VALUES ...
Change a value	UPDATE ... SET ... WHERE
Remove a record	DELETE FROM ... WHERE

SECTION 2 — OBJECT CLASSES IN JAVA

9. Core OOP Concepts — Theory

Concept	Definition	Real-world analogy
Class	A blueprint/template that defines fields and methods	Blueprint for a house
Object	An instance (copy) created from a class	An actual house built from the blueprint
Field	A variable that stores data about the object	The number of rooms in the house
Method	A function that defines what the object can do	Opening the front door
Encapsulation	Hiding data inside a class, controlling access through methods	A black box — you use it but can't see inside
Inheritance	A child class extends a parent class and gets its features	A sports car inherits features of a car
Polymorphism	Same method name behaves differently depending on the object	A 'speak' method — dog barks, cat meows

10. Access Modifiers

Modifier	Symbol (diagram)	Who can access it?
public	+	Any class anywhere
private	-	Only inside the same class
protected	#	Same class + any child (subclass) class



Key: Fields should almost ALWAYS be private. Methods that other classes need should be public.

11. Class Diagram → Java Code

Class Diagram Notation

Diagram notation	Java equivalent
- name : string	private String name;
- age : integer	private int age;
- price : real	private double price;
- active : boolean	private boolean active;
- initial : char	private char initial;
+ RATE = 0.15 : real (static)	public static final double RATE = 0.15;
- total : integer (static, underlined)	private static int total;
+ Constructor(n:string, a:integer)	public ClassName(String n, int a)
+ getName() : string	public String getName()
+ setName(n : string)	public void setName(String n)
+ toString() : string	public String toString()

12. Full Object Class — Template

Below is the standard structure of a Java object class with ALL components:

```
public class Student {
```

```
    // — FIELDS (always private) —————
    private String name;
    private int age;
    private double mark;
    private static int studentCount = 0;          // static field
    public static final double PASS MARK = 50.0; // constant
```

```
    // — CONSTRUCTOR —————
    public Student(String n, int a, double m) {
        name = n;
        age = a;
        mark = m;
        studentCount++; // increment static field
    }
```

```
    // — ACCESSOR METHODS (getters) —————
    public String getName() {
        return name;
    }
```

```
    public int getAge() {
        return age;
    }
```

```
    public double getMark() {
        return mark;
    }
```

```
    // — MUTATOR METHODS (setters) —————
    public void setName(String n) {
        name = n;
    }
```

```
    public void setMark(double m) {
        mark = m;
    }
```

```
    // — STATIC METHOD —————
    public static int getStudentCount() {
        return studentCount;
    }
```

```
    // — HELPER METHOD (private) —————
    private String getGrade() {
        if (mark >= 80) return "Distinction";
```

```

    else if (mark >= 50) return "Pass";
    else return "Fail";
}

```

```

// — toString METHOD —————
public String toString() {
    return name + " | Age: " + age + " | Mark: " + mark
        + " | Grade: " + getGrade();
}
}

```

13. Method Types — Summary

Method Type	Returns?	Parameters?	Access	Naming
Constructor	Nothing (no type)	Yes — sets fields	public	Same as class
Accessor (getter)	Yes — field's type	No	public	get + FieldName
Mutator (setter)	void	Yes — new value	public	set + FieldName
toString	String	No	public	toString
Helper method	Maybe	Maybe	private	Any name
Static method	Maybe	Maybe	public	get + StaticField

14. Inheritance — Parent & Child Classes

Parent (Superclass)

```
public class Vehicle {  
    protected String brand;    // protected = accessible in child  
    private int speed;  
}
```

```
    public Vehicle(String b, int s) {  
        brand = b;  
        speed = s;  
    }
```

```
    public String getBrand() { return brand; }  
    public int getSpeed()    { return speed; }
```

```
    public String toString() {  
        return brand + " at " + speed + " km/h";  
    }  
}
```

Child (Subclass)

```
public class Car extends Vehicle { // 'extends' = inheritance  
    private int numDoors;  
}
```

```
    // Child constructor — must call super() FIRST  
    public Car(String b, int s, int d) {  
        super(b, s);    // calls parent constructor  
        numDoors = d;  
    }
```

```
    public int getNumDoors() { return numDoors; }
```

```
    // Override parent toString  
    @Override  
    public String toString() {  
        return super.toString() + " | Doors: " + numDoors;  
    }  
}
```

Inheritance Rule	Detail
extends keyword	Child class inherits from parent
super(...)	Calls the parent constructor — must be FIRST line
super.method()	Calls a method from the parent class
@Override	Annotation — shows you're redefining a parent method
protected fields	Accessible in child class directly
private fields	NOT directly accessible — use getters from parent
Child constructor	Must include ALL parent parameters + its own

15. Application Class — Using Objects

Creating Objects and Using Methods

```

public class Main {
    public static void main(String[] args) {

        // Create an object
        Student s1 = new Student("Alice", 17, 85.5);
        Student s2 = new Student("Bob", 16, 42.0);

        // Use accessor methods
        System.out.println(s1.getName());    // Alice
        System.out.println(s1.getMark());    // 85.5

        // Use mutator methods
        s2.setMark(55.0);

        // toString is called automatically with println
        System.out.println(s1);
        System.out.println(s2);

        // Static method – called on class, not object
        System.out.println(Student.getStudentCount()); // 2
    }
}

```

16. Array of Objects

Declaring and Using an Object Array

```

Student[] students = new Student[5];    // array of 5 Students

students[0] = new Student("Alice", 17, 85.0);
students[1] = new Student("Bob", 16, 60.0);

// Loop through array
for (int i = 0; i < students.length; i++) {
    System.out.println(students[i].toString());
}

// Find the highest mark
double max = students[0].getMark();
for (int i = 1; i < students.length; i++) {
    if (students[i].getMark() > max) {
        max = students[i].getMark();
    }
}
System.out.println("Highest: " + max);

```

17. Common OOP Theory Questions & Answers

Question	Answer
What is encapsulation?	Hiding fields inside a class (making them private) and controlling access through public methods (getters/setters)
Why make fields private?	To protect data from being accidentally changed by other classes — improves security and prevents errors
What is a constructor?	A special method that creates an object and assigns values to its fields — called automatically with 'new'
Difference: accessor vs mutator?	Accessor (getter) RETURNS a field value. Mutator (setter) CHANGES a field value (void, has parameter)
What is method overloading?	Two methods in the same class with the SAME name but DIFFERENT parameters
What is method overriding?	A child class redefines a method from the parent class with the SAME name and parameters
What is a static field?	A field that belongs to the CLASS not an object — shared by ALL objects of that class
What is a helper method?	A private method that breaks a complex task into smaller pieces — only used inside the class
What does super() do?	Calls the parent class constructor from the child class — must be the first line
What is polymorphism?	When the same method behaves differently depending on which class's object is calling it

18. Class Diagram Quick Reference

Symbol	Meaning
+	public
-	private
#	protected
Underlined	static
UPPERCASE name	static constant
() after name	it is a method
: string / integer / real	return type or field type
(param : type)	method parameter
Arrow (child → parent)	inheritance relationship

19. Java Data Types — Class Diagram to Java

Class Diagram type	Java type	Example value
string	String	"Hello"
integer	int	42
real	double	3.14
boolean	boolean	true / false
char	char	'A'
string[]	String[]	new String[5]
integer[]	int[]	new int[10]

QUICK REFERENCE — ALL AT A GLANCE

SQL Cheat Sheet

Operation	SQL Syntax
Select all	SELECT * FROM TableName;
Select fields	SELECT f1, f2 FROM TableName;
Filter	WHERE condition
AND condition	WHERE a = 1 AND b = 2
OR condition	WHERE a = 1 OR b = 2
Pattern match	WHERE name LIKE 'A%'
Range	WHERE age BETWEEN 10 AND 20
Sort A-Z	ORDER BY field ASC
Sort Z-A	ORDER BY field DESC
Count	SELECT COUNT(*) FROM ...
Average	SELECT AVG(field) FROM ...
Max/Min	SELECT MAX(field), MIN(field) FROM ...
Group	GROUP BY field
Filter group	HAVING AVG(field) > 50
Join tables	INNER JOIN t2 ON t1.id = t2.id
No duplicates	SELECT DISTINCT field FROM ...
Add record	INSERT INTO t (f1,f2) VALUES (v1,v2);
Edit record	UPDATE t SET f = v WHERE condition;
Delete record	DELETE FROM t WHERE condition;

Java Object Class Cheat Sheet

What you need	Java code pattern
Private field	private String name;
Static field	private static int count = 0;
Constant	public static final double RATE = 0.1;
Constructor	public ClassName(String n) { name = n; }
Getter	public String getName() { return name; }
Setter	public void setName(String n) { name = n; }
toString	public String toString() { return name + ...; }
Helper	private String helper() { ... }
Inheritance	public class Child extends Parent { ... }
Call parent constructor	super(param1, param2);
Override method	@Override public String toString() { ... }
Create object	ClassName obj = new ClassName(args);
Call method	obj.getField(); or obj.setField(val);
Static call	ClassName.getStaticField();
Object array	ClassName[] arr = new ClassName[n];

Good luck in your revision session tomorrow! 🚀📖