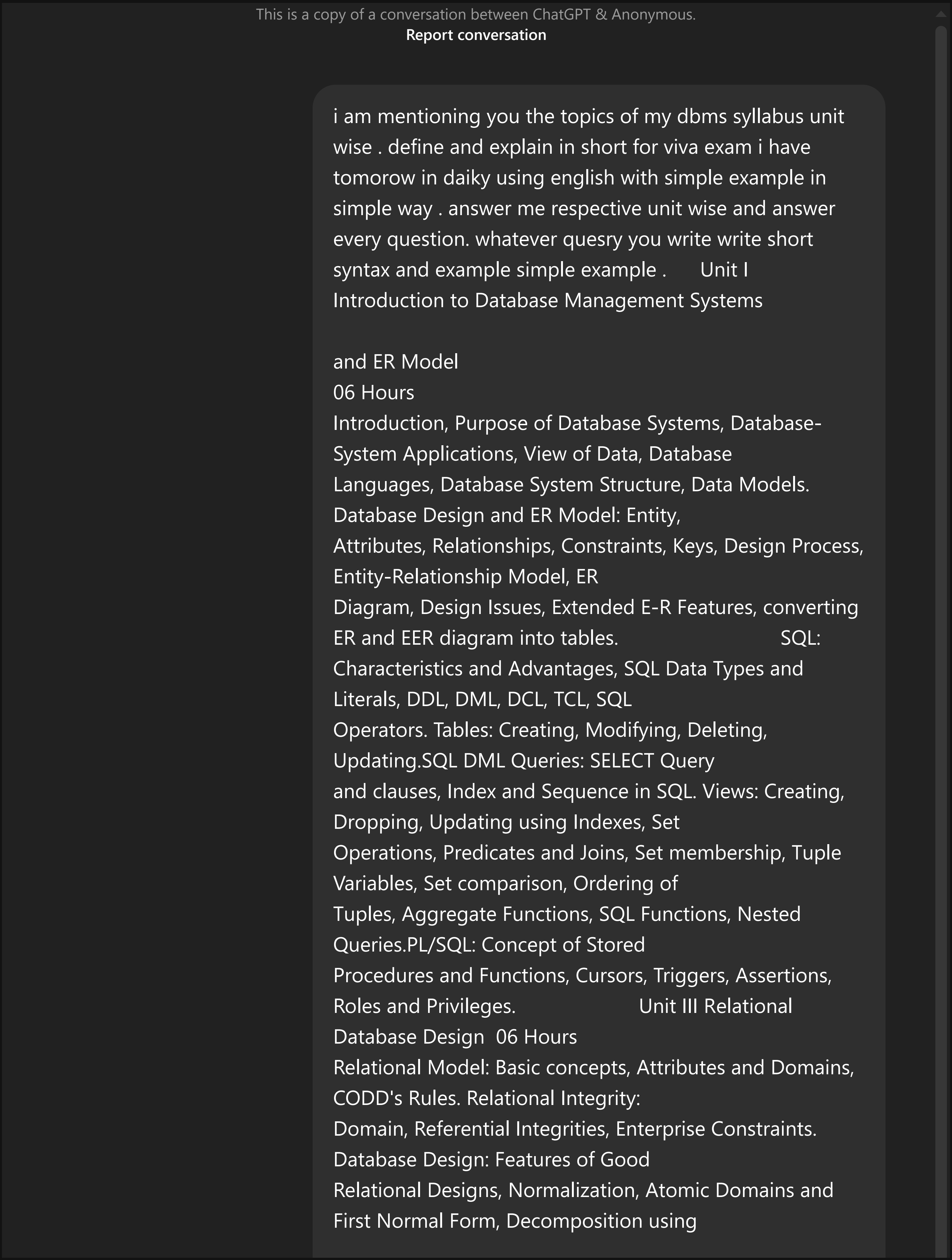
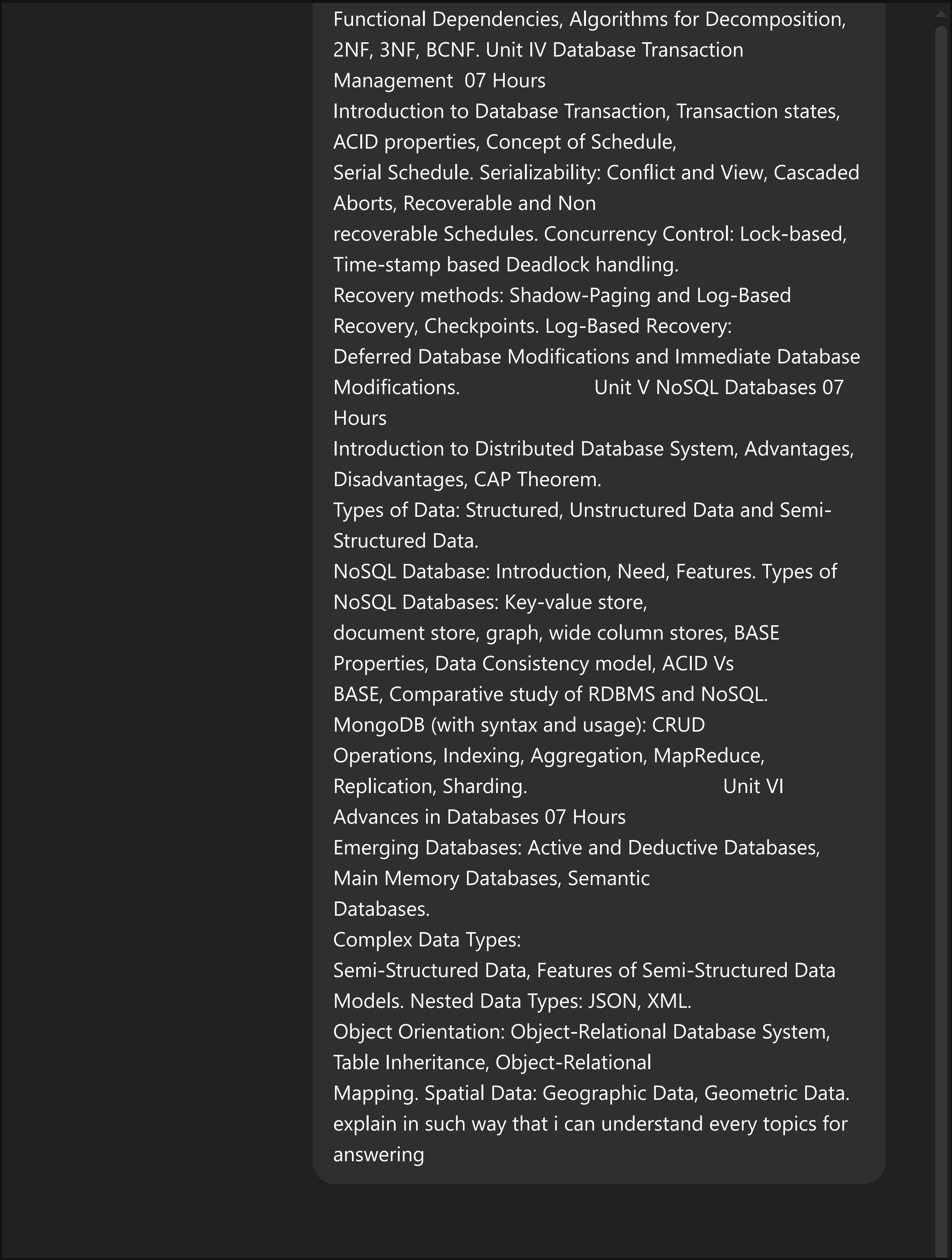


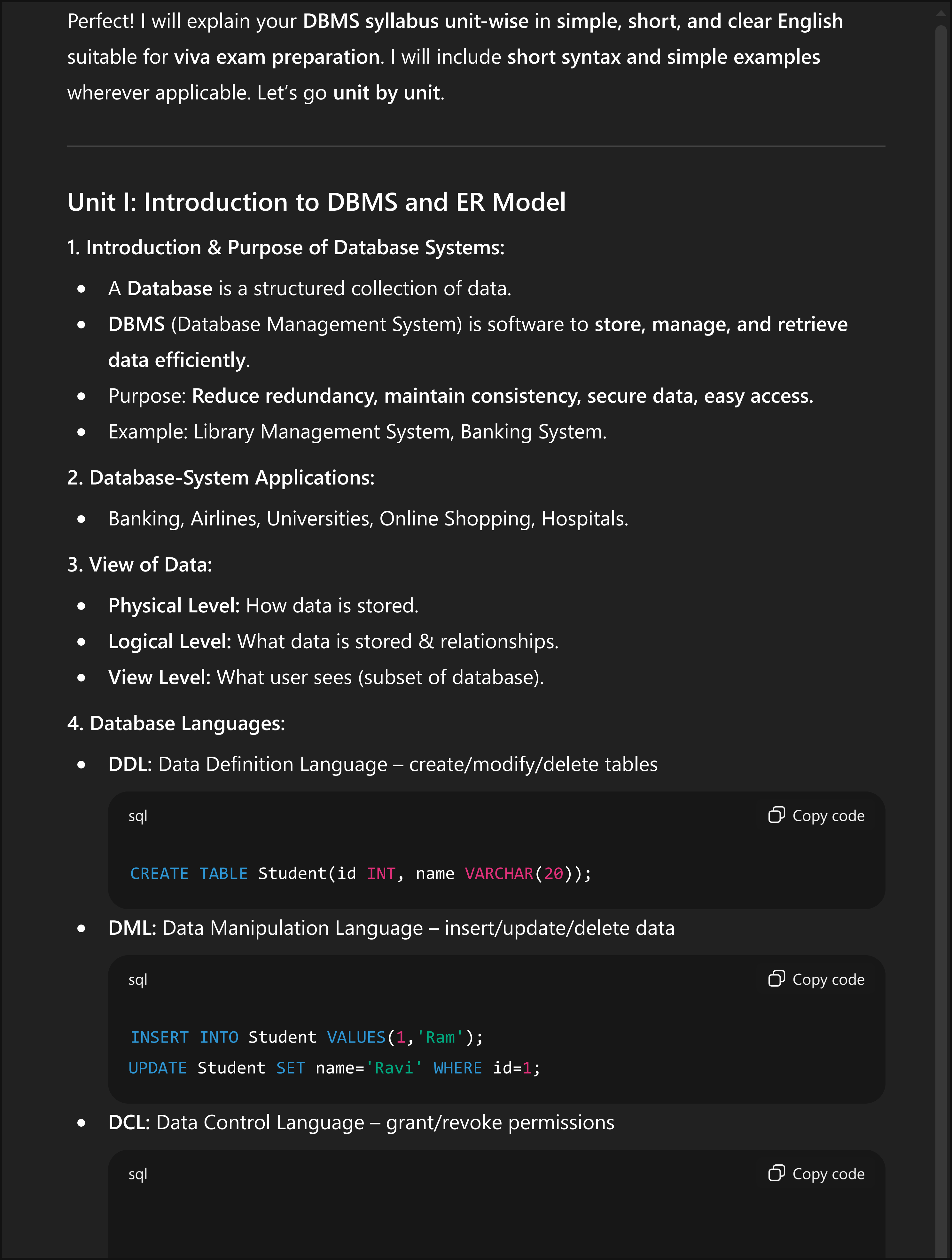
This is a copy of a conversation between ChatGPT & Anonymous.
Report conversation

i am mentioning you the topics of my dbms syllabus unit wise . define and explain in short for viva exam i have tomorow in daiky using english with simple example in simple way . answer me respective unit wise and answer every question. whatever quesry you write write short syntax and example simple example .   Unit I
Introduction to Database Management Systems

and ER Model
06 Hours
Introduction, Purpose of Database Systems, Database-System Applications, View of Data, Database
Languages, Database System Structure, Data Models.
Database Design and ER Model: Entity,
Attributes, Relationships, Constraints, Keys, Design Process, Entity-Relationship Model, ER
Diagram, Design Issues, Extended E-R Features, converting ER and EER diagram into tables.   SQL:
Characteristics and Advantages, SQL Data Types and Literals, DDL, DML, DCL, TCL, SQL
Operators. Tables: Creating, Modifying, Deleting, Updating.SQL DML Queries: SELECT Query
and clauses, Index and Sequence in SQL. Views: Creating, Dropping, Updating using Indexes, Set
Operations, Predicates and Joins, Set membership, Tuple Variables, Set comparison, Ordering of
Tuples, Aggregate Functions, SQL Functions, Nested Queries.PL/SQL: Concept of Stored
Procedures and Functions, Cursors, Triggers, Assertions, Roles and Privileges.   Unit III Relational Database Design 06 Hours
Relational Model: Basic concepts, Attributes and Domains, CODD's Rules. Relational Integrity:
Domain, Referential Integrities, Enterprise Constraints. Database Design: Features of Good
Relational Designs, Normalization, Atomic Domains and First Normal Form, Decomposition using



Functional Dependencies, Algorithms for Decomposition, 2NF, 3NF, BCNF. Unit IV Database Transaction Management 07 Hours
Introduction to Database Transaction, Transaction states, ACID properties, Concept of Schedule,
Serial Schedule. Serializability: Conflict and View, Cascaded Aborts, Recoverable and Non
recoverable Schedules. Concurrency Control: Lock-based, Time-stamp based Deadlock handling.
Recovery methods: Shadow-Paging and Log-Based Recovery, Checkpoints. Log-Based Recovery:
Deferred Database Modifications and Immediate Database Modifications. Unit V NoSQL Databases 07 Hours
Introduction to Distributed Database System, Advantages, Disadvantages, CAP Theorem.
Types of Data: Structured, Unstructured Data and Semi-Structured Data.
NoSQL Database: Introduction, Need, Features. Types of NoSQL Databases: Key-value store,
document store, graph, wide column stores, BASE Properties, Data Consistency model, ACID Vs
BASE, Comparative study of RDBMS and NoSQL.
MongoDB (with syntax and usage): CRUD
Operations, Indexing, Aggregation, MapReduce, Replication, Sharding. Unit VI
Advances in Databases 07 Hours
Emerging Databases: Active and Deductive Databases, Main Memory Databases, Semantic
Databases.
Complex Data Types:
Semi-Structured Data, Features of Semi-Structured Data Models. Nested Data Types: JSON, XML.
Object Orientation: Object-Relational Database System, Table Inheritance, Object-Relational
Mapping. Spatial Data: Geographic Data, Geometric Data.
explain in such way that i can understand every topics for answering



Perfect! I will explain your **DBMS syllabus unit-wise** in **simple, short, and clear English** suitable for **viva exam preparation**. I will include **short syntax and simple examples** wherever applicable. Let's go **unit by unit**.

Unit I: Introduction to DBMS and ER Model

1. Introduction & Purpose of Database Systems:

- A **Database** is a structured collection of data.
- **DBMS** (Database Management System) is software to **store, manage, and retrieve data efficiently**.
- Purpose: **Reduce redundancy, maintain consistency, secure data, easy access.**
- Example: Library Management System, Banking System.

2. Database-System Applications:

- Banking, Airlines, Universities, Online Shopping, Hospitals.

3. View of Data:

- **Physical Level:** How data is stored.
- **Logical Level:** What data is stored & relationships.
- **View Level:** What user sees (subset of database).

4. Database Languages:

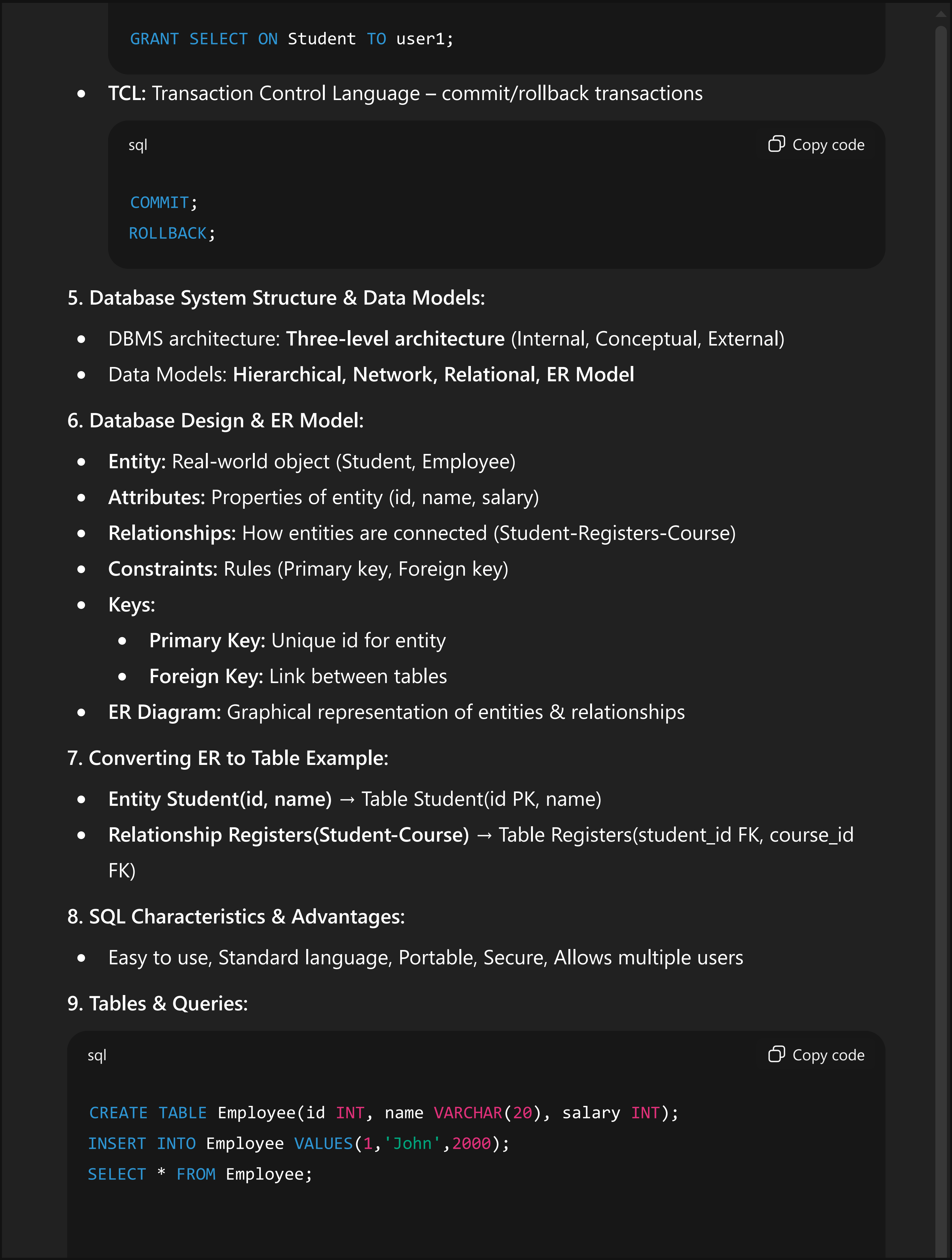
- **DDL:** Data Definition Language – create/modify/delete tables

```sql
CREATE TABLE Student(id INT, name VARCHAR(20));
```

- **DML:** Data Manipulation Language – insert/update/delete data

```sql
INSERT INTO Student VALUES(1,'Ram');
UPDATE Student SET name='Ravi' WHERE id=1;
```

- **DCL:** Data Control Language – grant/revoke permissions



```sql
GRANT SELECT ON Student TO user1;
```

- **TCL:** Transaction Control Language – commit/rollback transactions

```sql
COMMIT;
ROLLBACK;
```

5. Database System Structure & Data Models:

- DBMS architecture: **Three-level architecture** (Internal, Conceptual, External)
- Data Models: **Hierarchical, Network, Relational, ER Model**

6. Database Design & ER Model:

- **Entity:** Real-world object (Student, Employee)
- **Attributes:** Properties of entity (id, name, salary)
- **Relationships:** How entities are connected (Student-Registers-Course)
- **Constraints:** Rules (Primary key, Foreign key)
- **Keys:**
  - **Primary Key:** Unique id for entity
  - **Foreign Key:** Link between tables
- **ER Diagram:** Graphical representation of entities & relationships

7. Converting ER to Table Example:

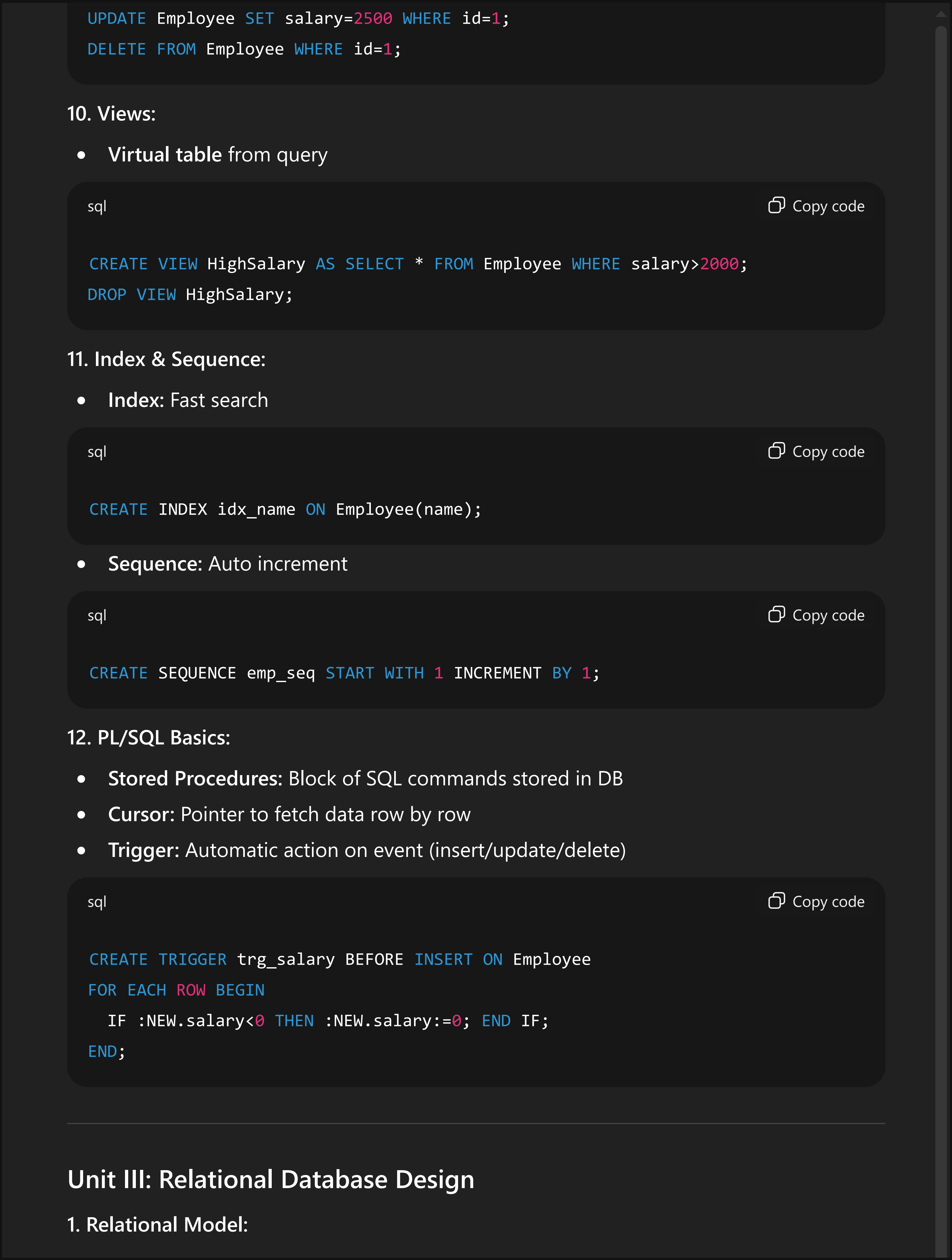
- **Entity Student(id, name)** → Table Student(id PK, name)
- **Relationship Registers(Student-Course)** → Table Registers(student_id FK, course_id FK)

8. SQL Characteristics & Advantages:

- Easy to use, Standard language, Portable, Secure, Allows multiple users

9. Tables & Queries:

```sql
CREATE TABLE Employee(id INT, name VARCHAR(20), salary INT);
INSERT INTO Employee VALUES(1,'John',2000);
SELECT * FROM Employee;
```



```sql
UPDATE Employee SET salary=2500 WHERE id=1;
DELETE FROM Employee WHERE id=1;
```

10. Views:

- **Virtual table** from query

```sql
CREATE VIEW HighSalary AS SELECT * FROM Employee WHERE salary>2000;
DROP VIEW HighSalary;
```

11. Index & Sequence:

- **Index:** Fast search

```sql
CREATE INDEX idx_name ON Employee(name);
```

- **Sequence:** Auto increment

```sql
CREATE SEQUENCE emp_seq START WITH 1 INCREMENT BY 1;
```

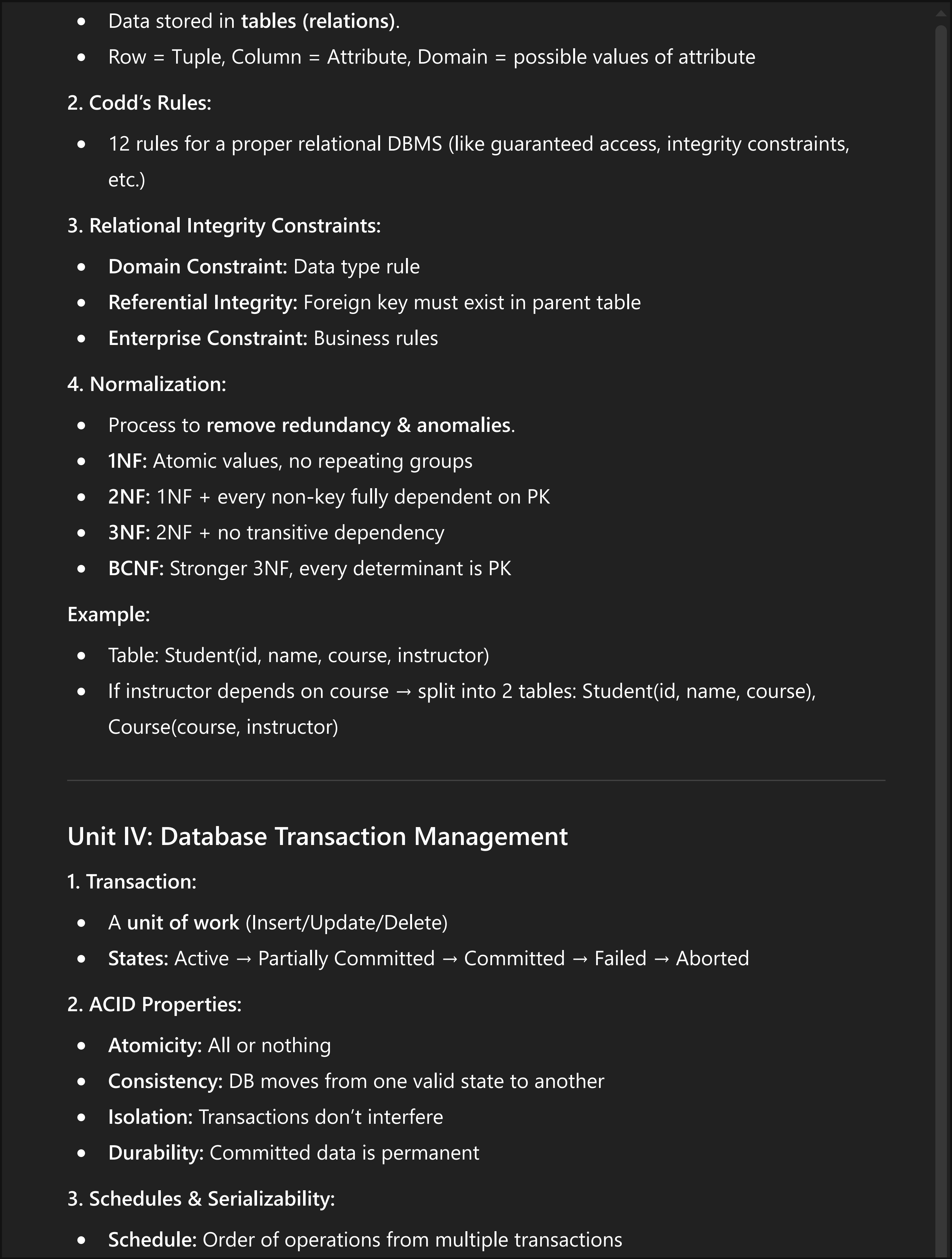
12. PL/SQL Basics:

- **Stored Procedures:** Block of SQL commands stored in DB
- **Cursor:** Pointer to fetch data row by row
- **Trigger:** Automatic action on event (insert/update/delete)

```sql
CREATE TRIGGER trg_salary BEFORE INSERT ON Employee
FOR EACH ROW BEGIN
  IF :NEW.salary<0 THEN :NEW.salary:=0; END IF;
END;
```

Unit III: Relational Database Design

1. Relational Model:



- Data stored in **tables (relations)**.
- Row = Tuple, Column = Attribute, Domain = possible values of attribute

2. Codd’s Rules:

- 12 rules for a proper relational DBMS (like guaranteed access, integrity constraints, etc.)

3. Relational Integrity Constraints:

- **Domain Constraint:** Data type rule
- **Referential Integrity:** Foreign key must exist in parent table
- **Enterprise Constraint:** Business rules

4. Normalization:

- Process to **remove redundancy & anomalies**.
- **1NF:** Atomic values, no repeating groups
- **2NF:** 1NF + every non-key fully dependent on PK
- **3NF:** 2NF + no transitive dependency
- **BCNF:** Stronger 3NF, every determinant is PK

Example:

- Table: Student(id, name, course, instructor)
- If instructor depends on course → split into 2 tables: Student(id, name, course), Course(course, instructor)

Unit IV: Database Transaction Management

1. Transaction:

- A **unit of work** (Insert/Update/Delete)
- **States:** Active → Partially Committed → Committed → Failed → Aborted

2. ACID Properties:

- **Atomicity:** All or nothing
- **Consistency:** DB moves from one valid state to another
- **Isolation:** Transactions don’t interfere
- **Durability:** Committed data is permanent

3. Schedules & Serializability:

- **Schedule:** Order of operations from multiple transactions

- **Conflict Serializability:** Can be reordered to get same result
- **Recoverable Schedule:** No commit if dependency fails

4. Concurrency Control:

- **Lock-based:** Shared & Exclusive locks
- **Timestamp-based:** Order transactions by timestamp
- **Deadlock handling:** Wait-Die or Wound-Wait

5. Recovery Methods:

- **Shadow Paging:** Keep a shadow copy, update only after commit
- **Log-based:** Keep log for undo/redo
- **Checkpoints:** Save DB state periodically

Unit V: NoSQL Databases

1. Distributed Database System:

- DB spread across multiple locations, supports **parallel processing**
- Advantages: Scalability, Availability
- CAP Theorem: **Consistency, Availability, Partition tolerance – can only have 2**

2. Types of Data:

- Structured: Tables
- Semi-structured: JSON, XML
- Unstructured: Text, Video, Images

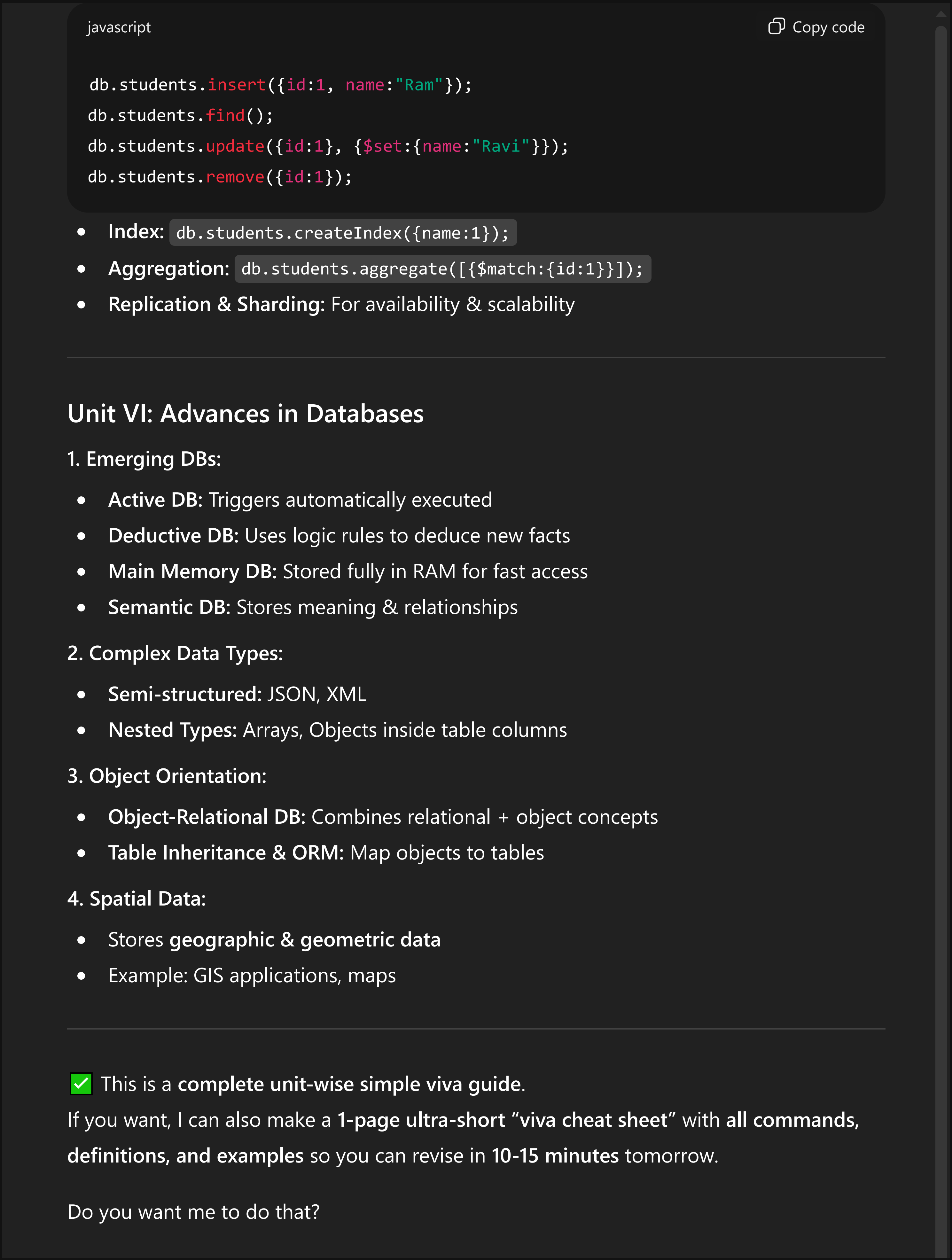
3. NoSQL DB:

- **Need:** RDBMS not scalable for Big Data
- **Types:**
  - **Key-Value Store:** Redis
  - **Document Store:** MongoDB (JSON)
  - **Graph DB:** Neo4j
  - **Wide Column Store:** Cassandra

4. BASE vs ACID:

- BASE: Basically Available, Soft state, Eventually consistent
- ACID: Strict consistency

5. MongoDB Example:



```javascript
db.students.insert({id:1, name:"Ram"});
db.students.find();
db.students.update({id:1}, {$set:{name:"Ravi"}});
db.students.remove({id:1});
```

- **Index:** `db.students.createIndex({name:1});`
- **Aggregation:** `db.students.aggregate([{$match:{id:1}}]);`
- **Replication & Sharding:** For availability & scalability

Unit VI: Advances in Databases

1. Emerging DBs:

- **Active DB:** Triggers automatically executed
- **Deductive DB:** Uses logic rules to deduce new facts
- **Main Memory DB:** Stored fully in RAM for fast access
- **Semantic DB:** Stores meaning & relationships

2. Complex Data Types:

- **Semi-structured:** JSON, XML
- **Nested Types:** Arrays, Objects inside table columns

3. Object Orientation:

- **Object-Relational DB:** Combines relational + object concepts
- **Table Inheritance & ORM:** Map objects to tables

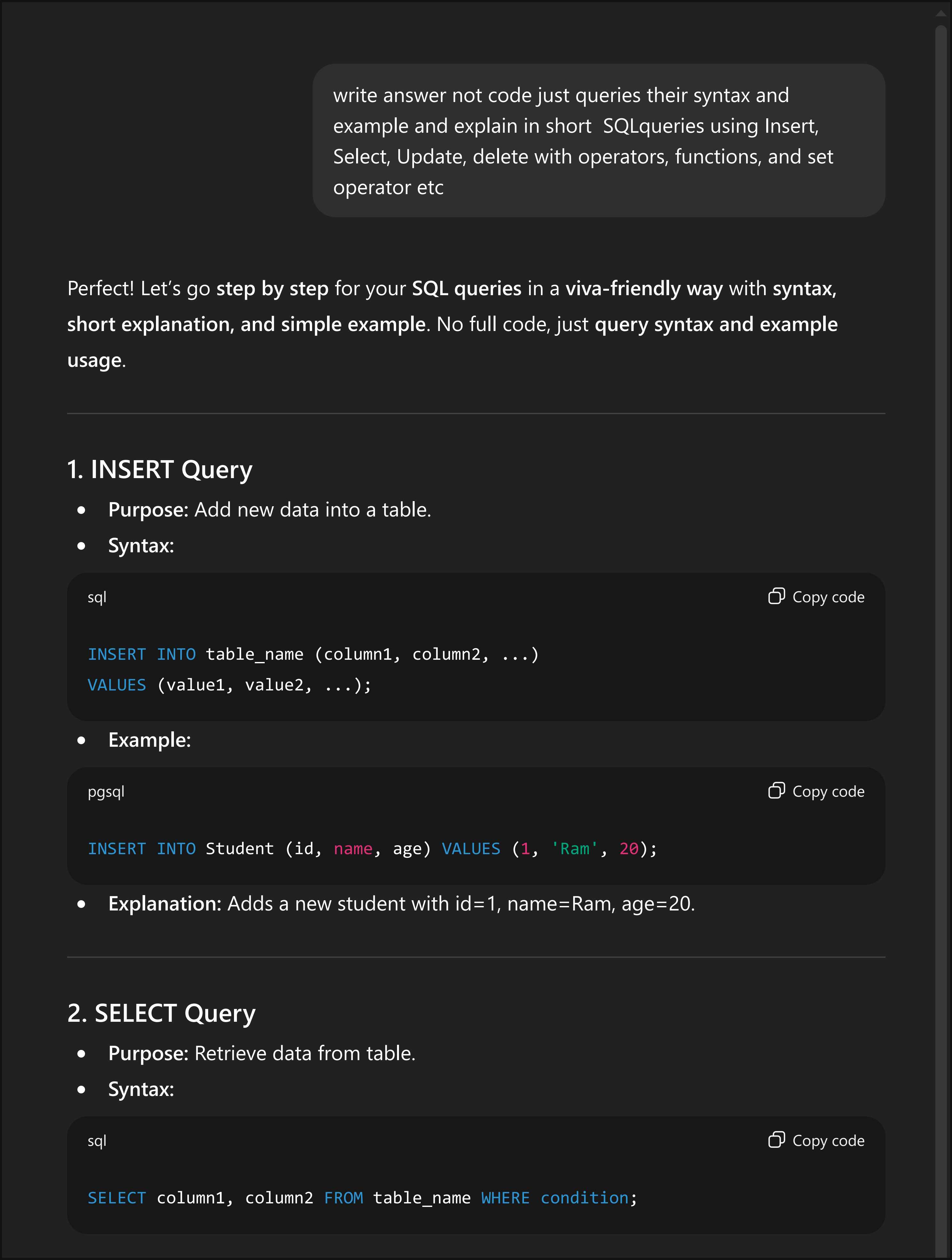
4. Spatial Data:

- Stores **geographic & geometric data**
- Example: GIS applications, maps

✅ This is a **complete unit-wise simple viva guide**.

If you want, I can also make a **1-page ultra-short “viva cheat sheet”** with **all commands, definitions, and examples** so you can revise in **10-15 minutes** tomorrow.

Do you want me to do that?



write answer not code just queries their syntax and example and explain in short SQLqueries using Insert, Select, Update, delete with operators, functions, and set operator etc

Perfect! Let's go **step by step** for your **SQL queries** in a **viva-friendly way** with **syntax, short explanation, and simple example**. No full code, just **query syntax and example usage**.

---

1. INSERT Query

- **Purpose:** Add new data into a table.
- **Syntax:**

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

- **Example:**

```pgsql
INSERT INTO Student (id, name, age) VALUES (1, 'Ram', 20);
```

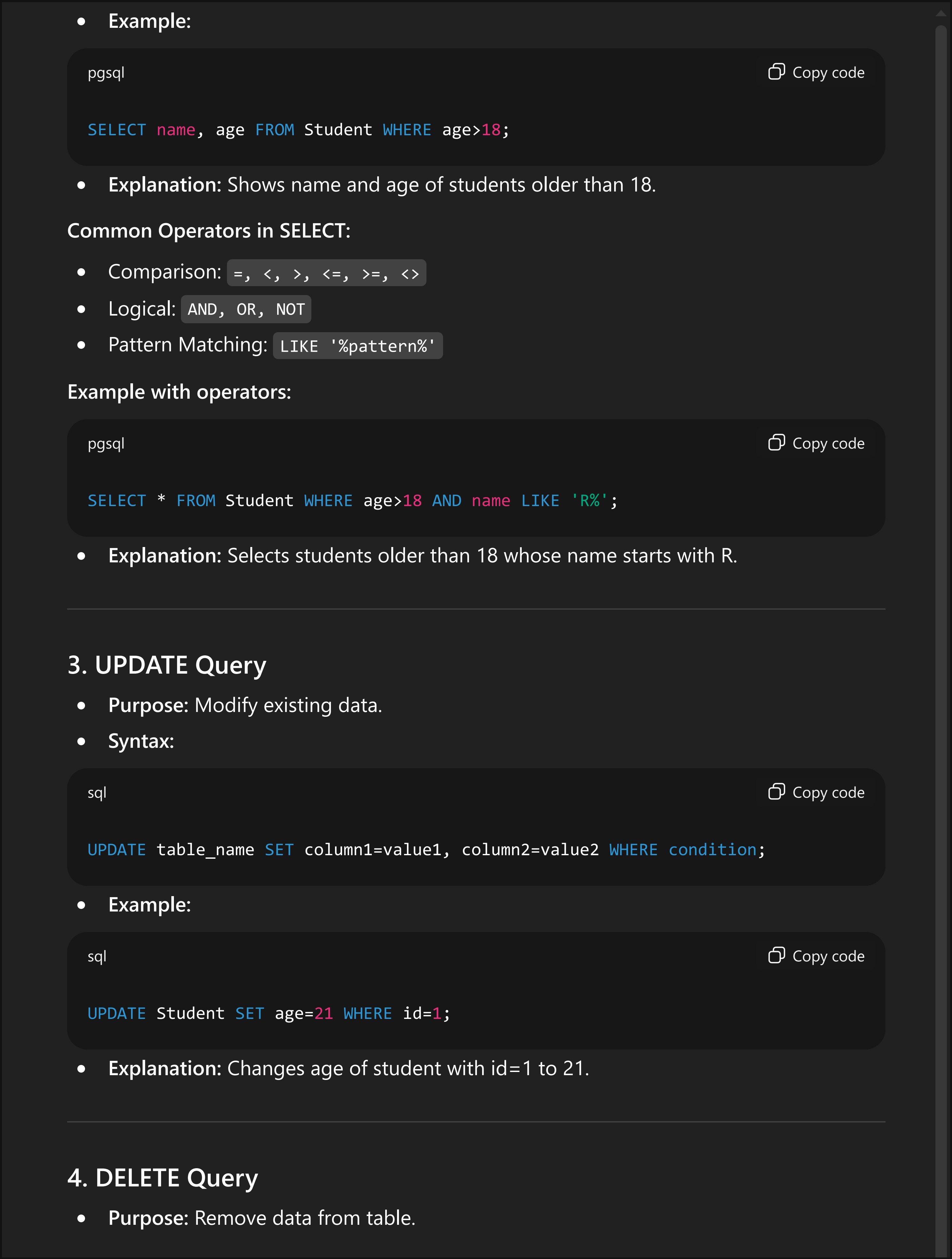
- **Explanation:** Adds a new student with id=1, name=Ram, age=20.

---

2. SELECT Query

- **Purpose:** Retrieve data from table.
- **Syntax:**

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



- **Example:**

pgsql

```
SELECT name, age FROM Student WHERE age>18;
```

- **Explanation:** Shows name and age of students older than 18.

Common Operators in SELECT:

- Comparison: `=, <, >, <=, >=, <>`
- Logical: `AND, OR, NOT`
- Pattern Matching: `LIKE '%pattern%'`

Example with operators:

pgsql

```
SELECT * FROM Student WHERE age>18 AND name LIKE 'R%';
```

- **Explanation:** Selects students older than 18 whose name starts with R.

---

3. UPDATE Query

- **Purpose:** Modify existing data.
- **Syntax:**

sql

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

- **Example:**

sql

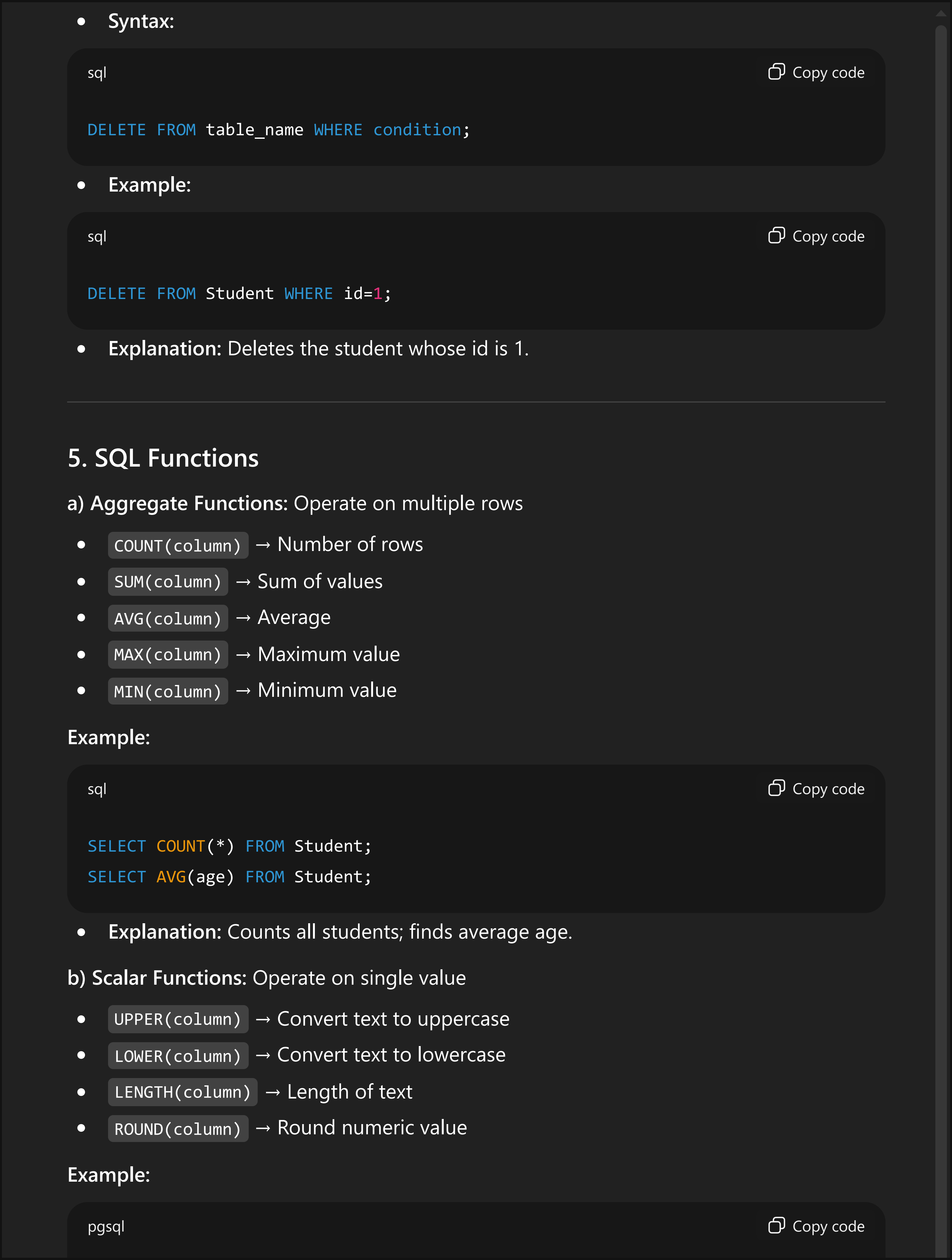
```
UPDATE Student SET age=21 WHERE id=1;
```

- **Explanation:** Changes age of student with id=1 to 21.

---

4. DELETE Query

- **Purpose:** Remove data from table.



- **Syntax:**

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

- **Example:**

```sql
DELETE FROM Student WHERE id=1;
```

- **Explanation:** Deletes the student whose id is 1.

---

5. SQL Functions

a) Aggregate Functions: Operate on multiple rows

- `COUNT(column)` → Number of rows
- `SUM(column)` → Sum of values
- `AVG(column)` → Average
- `MAX(column)` → Maximum value
- `MIN(column)` → Minimum value

Example:

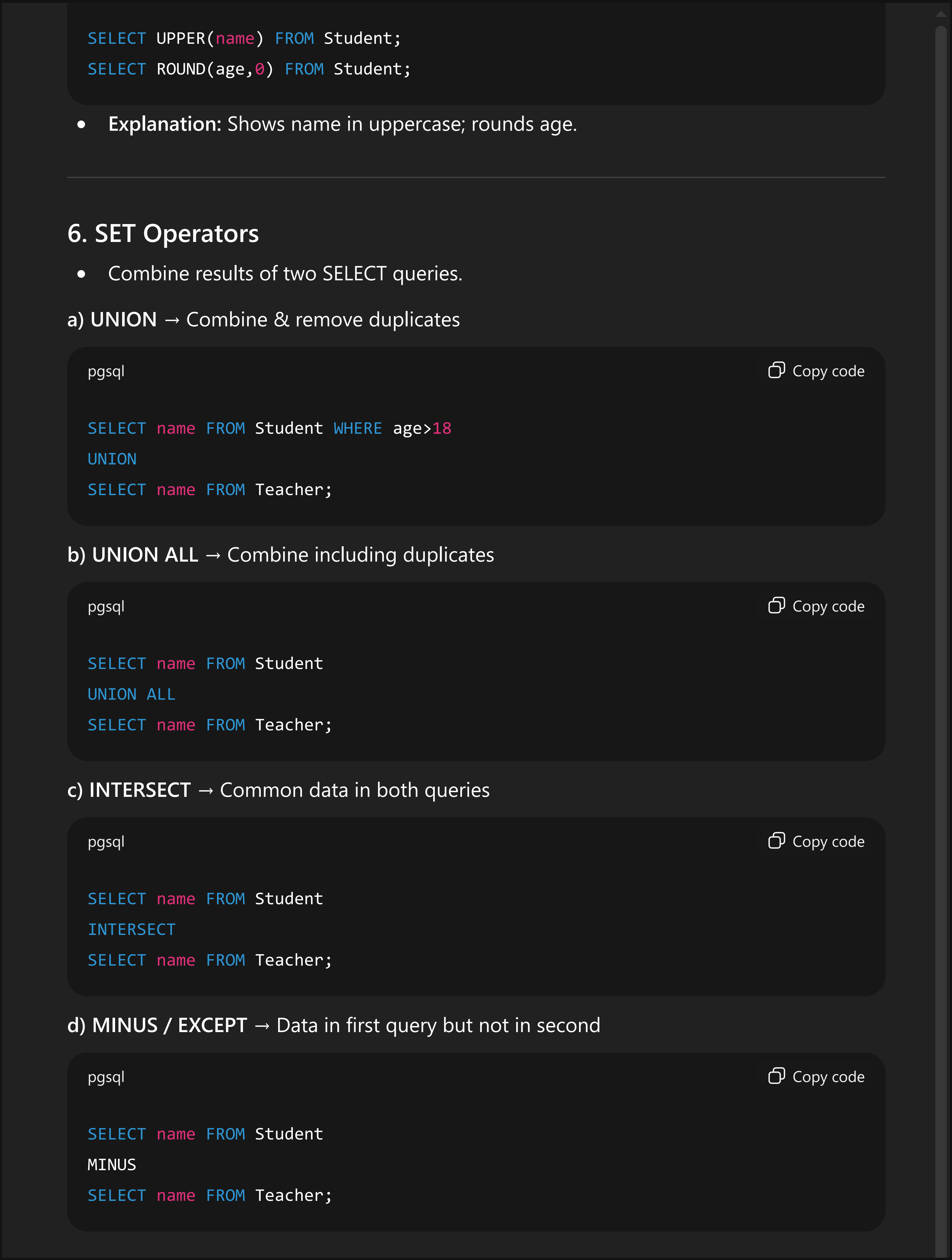
```sql
SELECT COUNT(*) FROM Student;
SELECT AVG(age) FROM Student;
```

- **Explanation:** Counts all students; finds average age.

b) Scalar Functions: Operate on single value

- `UPPER(column)` → Convert text to uppercase
- `LOWER(column)` → Convert text to lowercase
- `LENGTH(column)` → Length of text
- `ROUND(column)` → Round numeric value

Example:



```
SELECT UPPER(name) FROM Student;
SELECT ROUND(age,0) FROM Student;
```

- **Explanation:** Shows name in uppercase; rounds age.

---

6. SET Operators

- Combine results of two SELECT queries.

a) **UNION** → Combine & remove duplicates

```pgsql
SELECT name FROM Student WHERE age>18
UNION
SELECT name FROM Teacher;
```

b) **UNION ALL** → Combine including duplicates

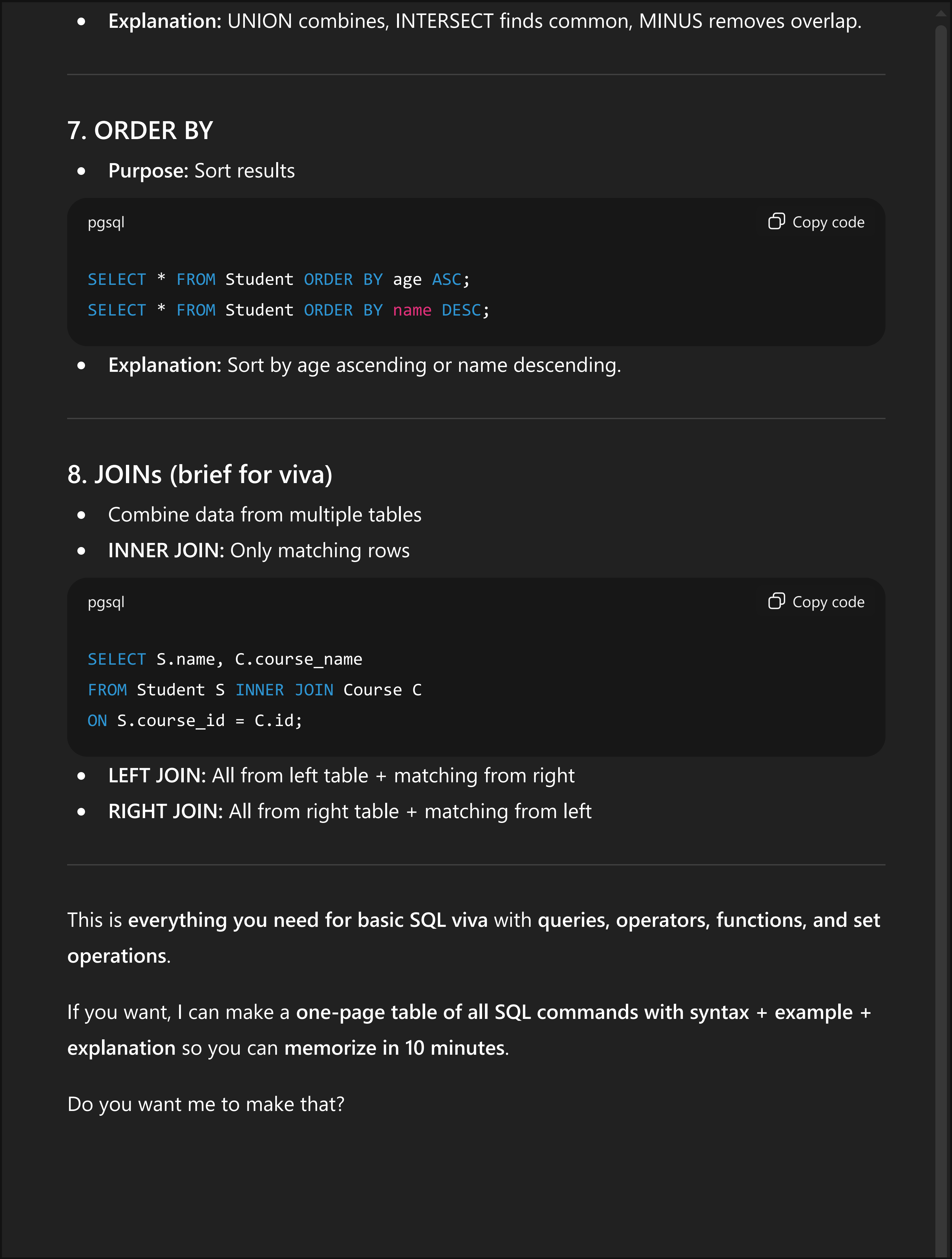
```pgsql
SELECT name FROM Student
UNION ALL
SELECT name FROM Teacher;
```

c) **INTERSECT** → Common data in both queries

```pgsql
SELECT name FROM Student
INTERSECT
SELECT name FROM Teacher;
```

d) **MINUS / EXCEPT** → Data in first query but not in second

```pgsql
SELECT name FROM Student
MINUS
SELECT name FROM Teacher;
```



- **Explanation:** UNION combines, INTERSECT finds common, MINUS removes overlap.

---

7. ORDER BY

- **Purpose:** Sort results

```pgsql
SELECT * FROM Student ORDER BY age ASC;
SELECT * FROM Student ORDER BY name DESC;
```

- **Explanation:** Sort by age ascending or name descending.

---

8. JOINs (brief for viva)

- Combine data from multiple tables
- **INNER JOIN:** Only matching rows

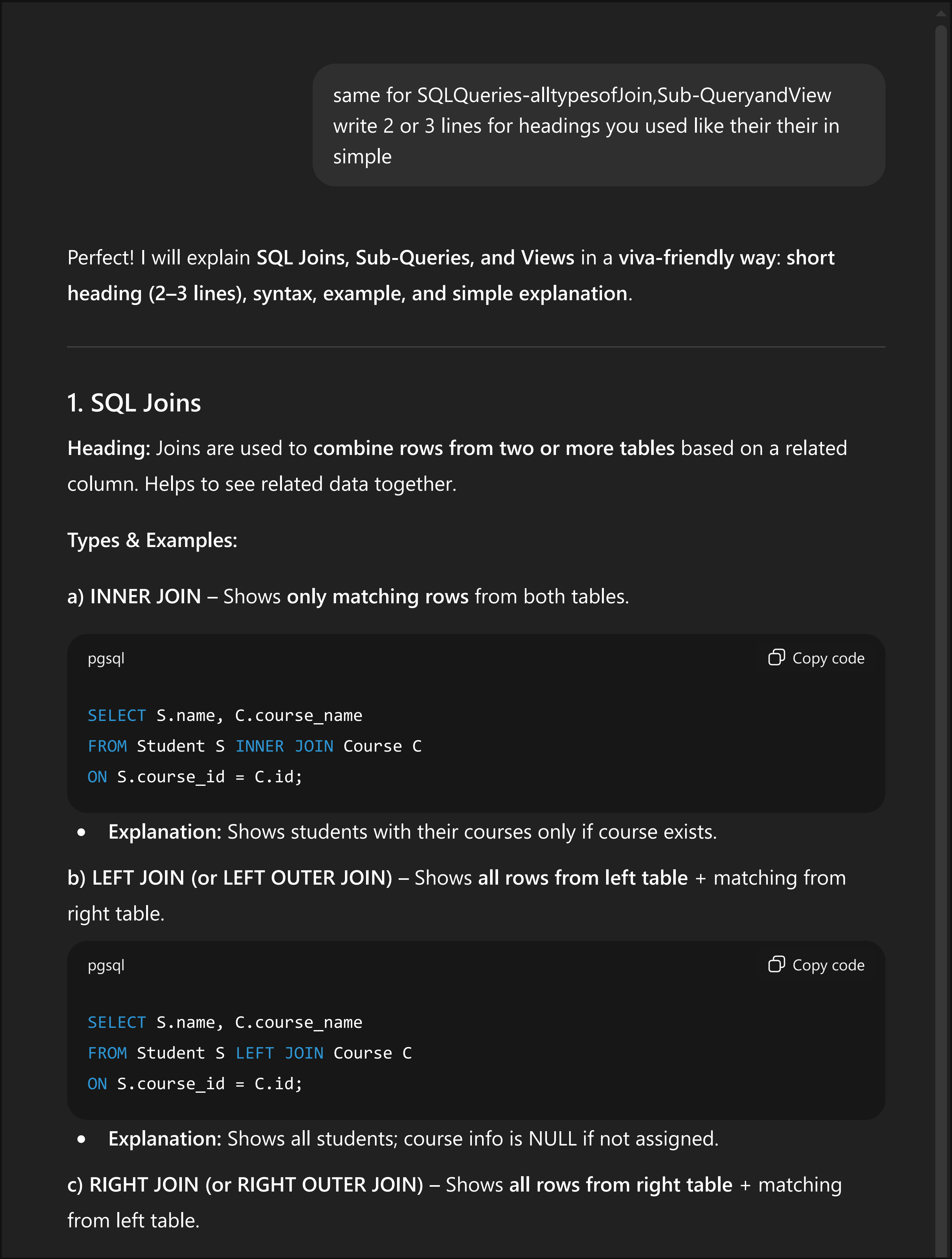
```pgsql
SELECT S.name, C.course_name
FROM Student S INNER JOIN Course C
ON S.course_id = C.id;
```

- **LEFT JOIN:** All from left table + matching from right
- **RIGHT JOIN:** All from right table + matching from left

This is **everything you need for basic SQL viva** with **queries, operators, functions, and set operations**.

If you want, I can make a **one-page table of all SQL commands with syntax + example + explanation** so you can **memorize in 10 minutes**.

Do you want me to make that?



same for SQLQueries-alltypesofJoin,Sub-QueryandView write 2 or 3 lines for headings you used like their their in simple

Perfect! I will explain **SQL Joins, Sub-Queries, and Views** in a **viva-friendly way: short heading (2–3 lines), syntax, example, and simple explanation**.

---

1. SQL Joins

Heading: Joins are used to **combine rows from two or more tables** based on a related column. Helps to see related data together.

Types & Examples:

a) INNER JOIN – Shows **only matching rows** from both tables.

```pgsql
SELECT S.name, C.course_name
FROM Student S INNER JOIN Course C
ON S.course_id = C.id;
```

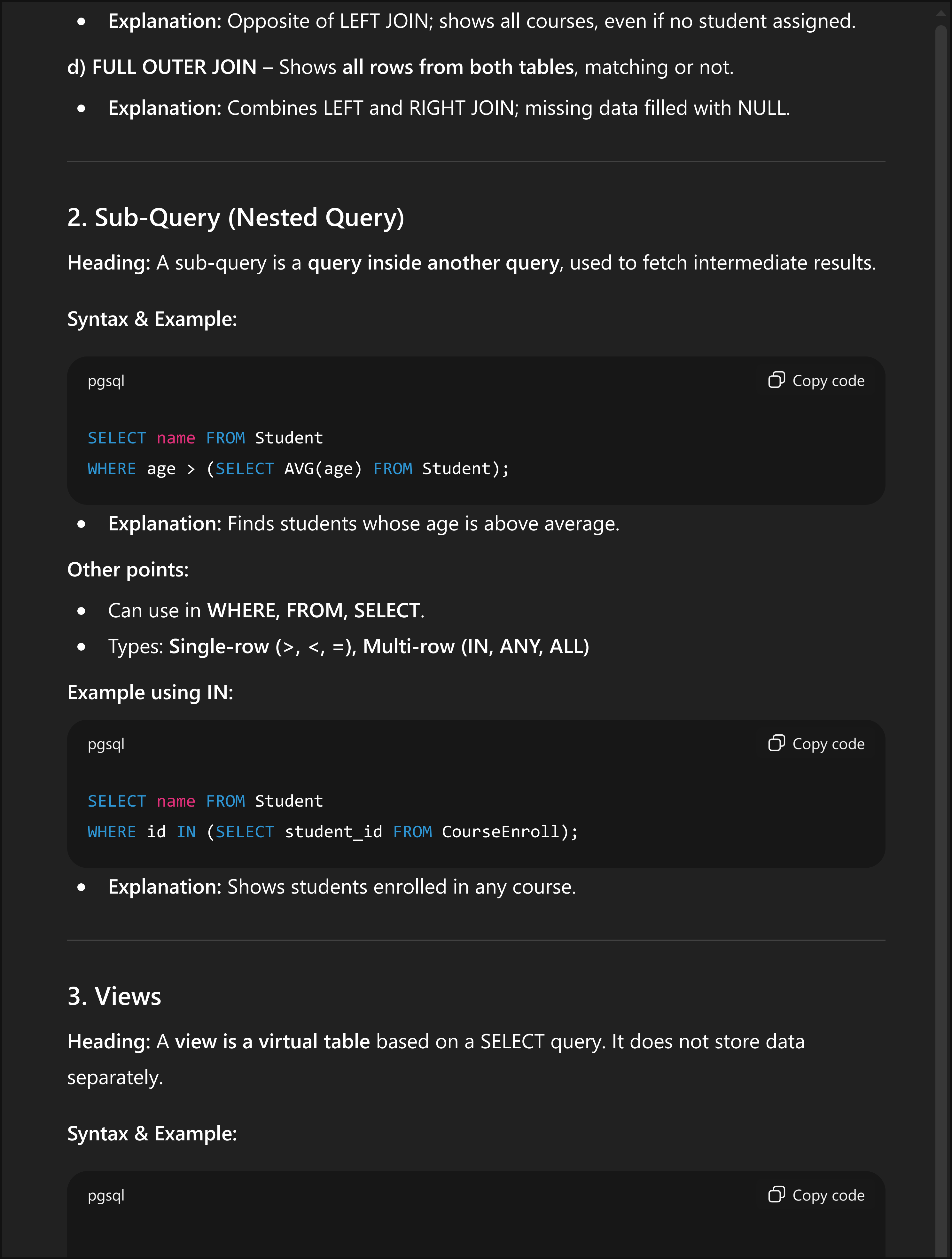
- **Explanation:** Shows students with their courses only if course exists.

b) LEFT JOIN (or LEFT OUTER JOIN) – Shows **all rows from left table** + matching from right table.

```pgsql
SELECT S.name, C.course_name
FROM Student S LEFT JOIN Course C
ON S.course_id = C.id;
```

- **Explanation:** Shows all students; course info is NULL if not assigned.

c) RIGHT JOIN (or RIGHT OUTER JOIN) – Shows **all rows from right table** + matching from left table.



- **Explanation:** Opposite of LEFT JOIN; shows all courses, even if no student assigned.

d) FULL OUTER JOIN – Shows **all rows from both tables**, matching or not.

- **Explanation:** Combines LEFT and RIGHT JOIN; missing data filled with NULL.

---

2. Sub-Query (Nested Query)

Heading: A sub-query is a **query inside another query**, used to fetch intermediate results.

Syntax & Example:

```pgsql
SELECT name FROM Student
WHERE age > (SELECT AVG(age) FROM Student);
```

- **Explanation:** Finds students whose age is above average.

Other points:

- Can use in **WHERE, FROM, SELECT**.
- Types: **Single-row** (>, <, =), **Multi-row (IN, ANY, ALL)**

Example using IN:

```pgsql
SELECT name FROM Student
WHERE id IN (SELECT student_id FROM CourseEnroll);
```

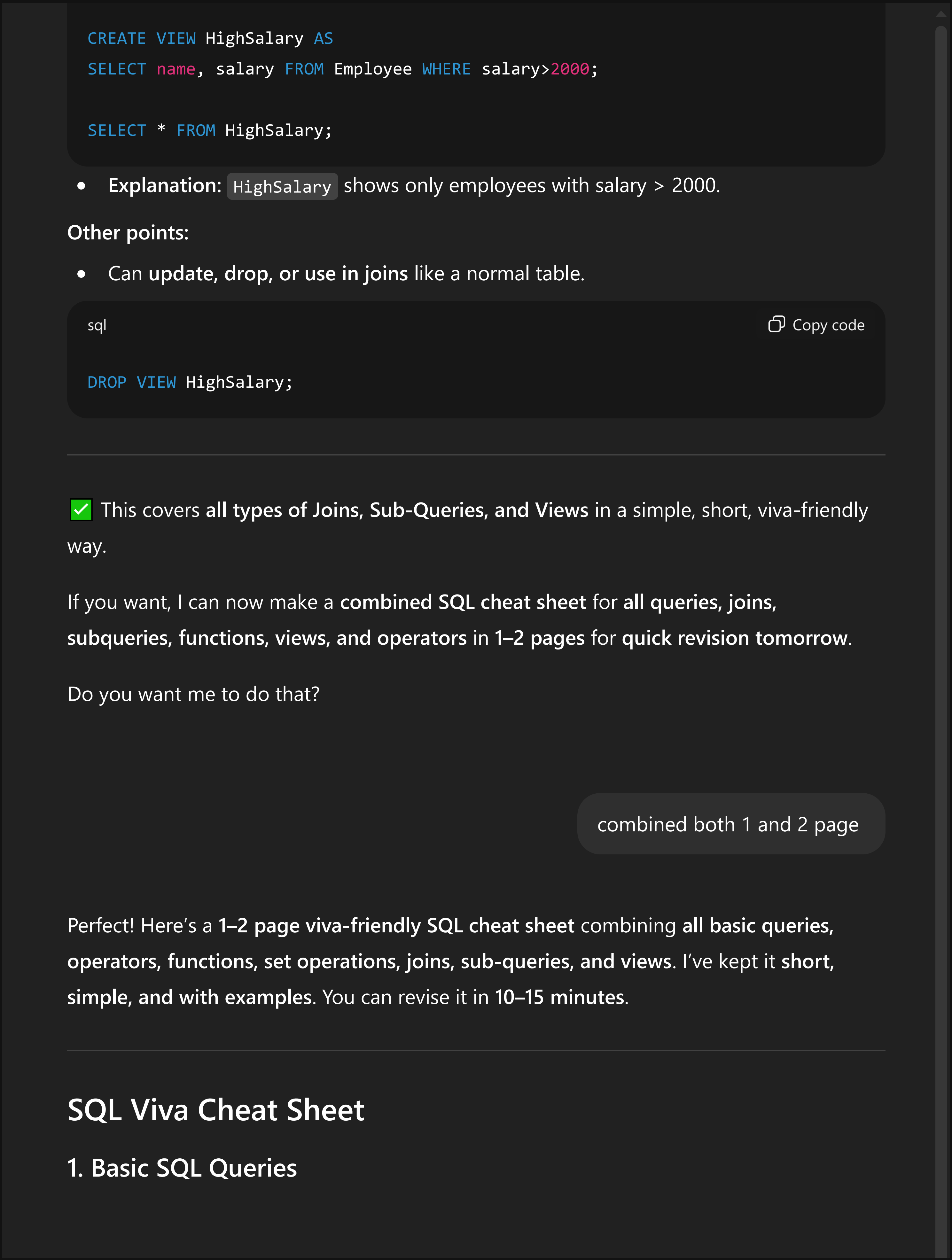
- **Explanation:** Shows students enrolled in any course.

---

3. Views

Heading: A **view is a virtual table** based on a SELECT query. It does not store data separately.

Syntax & Example:



```sql
CREATE VIEW HighSalary AS
SELECT name, salary FROM Employee WHERE salary>2000;

SELECT * FROM HighSalary;
```

- **Explanation:** `HighSalary` shows only employees with salary > 2000.

Other points:

- Can **update, drop, or use in joins** like a normal table.

```sql
DROP VIEW HighSalary;
```

✅ This covers **all types of Joins, Sub-Queries, and Views** in a simple, short, viva-friendly way.

If you want, I can now make a **combined SQL cheat sheet** for **all queries, joins, subqueries, functions, views, and operators** in **1–2 pages** for **quick revision tomorrow**.

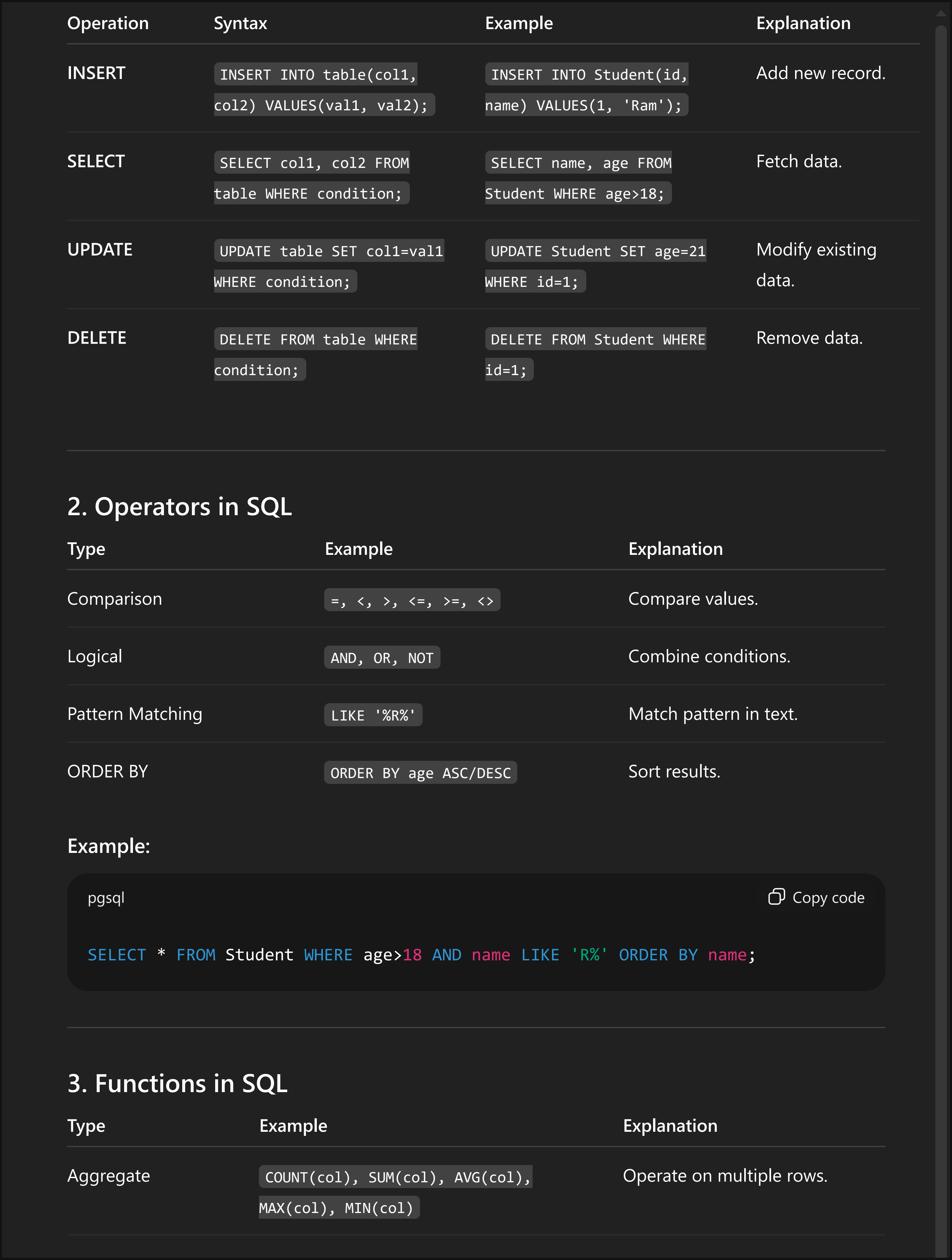
Do you want me to do that?

combined both 1 and 2 page

Perfect! Here's a **1–2 page viva-friendly SQL cheat sheet** combining **all basic queries, operators, functions, set operations, joins, sub-queries, and views**. I've kept it **short, simple, and with examples**. You can revise it in **10–15 minutes**.

SQL Viva Cheat Sheet

1. Basic SQL Queries



Operation	Syntax	Example	Explanation
INSERT	`INSERT INTO table(col1, col2) VALUES(val1, val2);`	`INSERT INTO Student(id, name) VALUES(1, 'Ram');`	Add new record.
SELECT	`SELECT col1, col2 FROM table WHERE condition;`	`SELECT name, age FROM Student WHERE age>18;`	Fetch data.
UPDATE	`UPDATE table SET col1=val1 WHERE condition;`	`UPDATE Student SET age=21 WHERE id=1;`	Modify existing data.
DELETE	`DELETE FROM table WHERE condition;`	`DELETE FROM Student WHERE id=1;`	Remove data.

---

2. Operators in SQL

Type	Example	Explanation
Comparison	`=, <, >, <=, >=, <>`	Compare values.
Logical	`AND, OR, NOT`	Combine conditions.
Pattern Matching	`LIKE '%R%'`	Match pattern in text.
ORDER BY	`ORDER BY age ASC/DESC`	Sort results.

Example:

```pgsql
SELECT * FROM Student WHERE age>18 AND name LIKE 'R%' ORDER BY name;
```

---

3. Functions in SQL

Type	Example	Explanation
Aggregate	`COUNT(col), SUM(col), AVG(col), MAX(col), MIN(col)`	Operate on multiple rows.

Type	Example	Explanation
Scalar	`UPPER(col), LOWER(col), LENGTH(col), ROUND(col)`	Operate on single value.

Example:

```pgsql
SELECT COUNT(*) FROM Student;
SELECT UPPER(name) FROM Student;
```

---

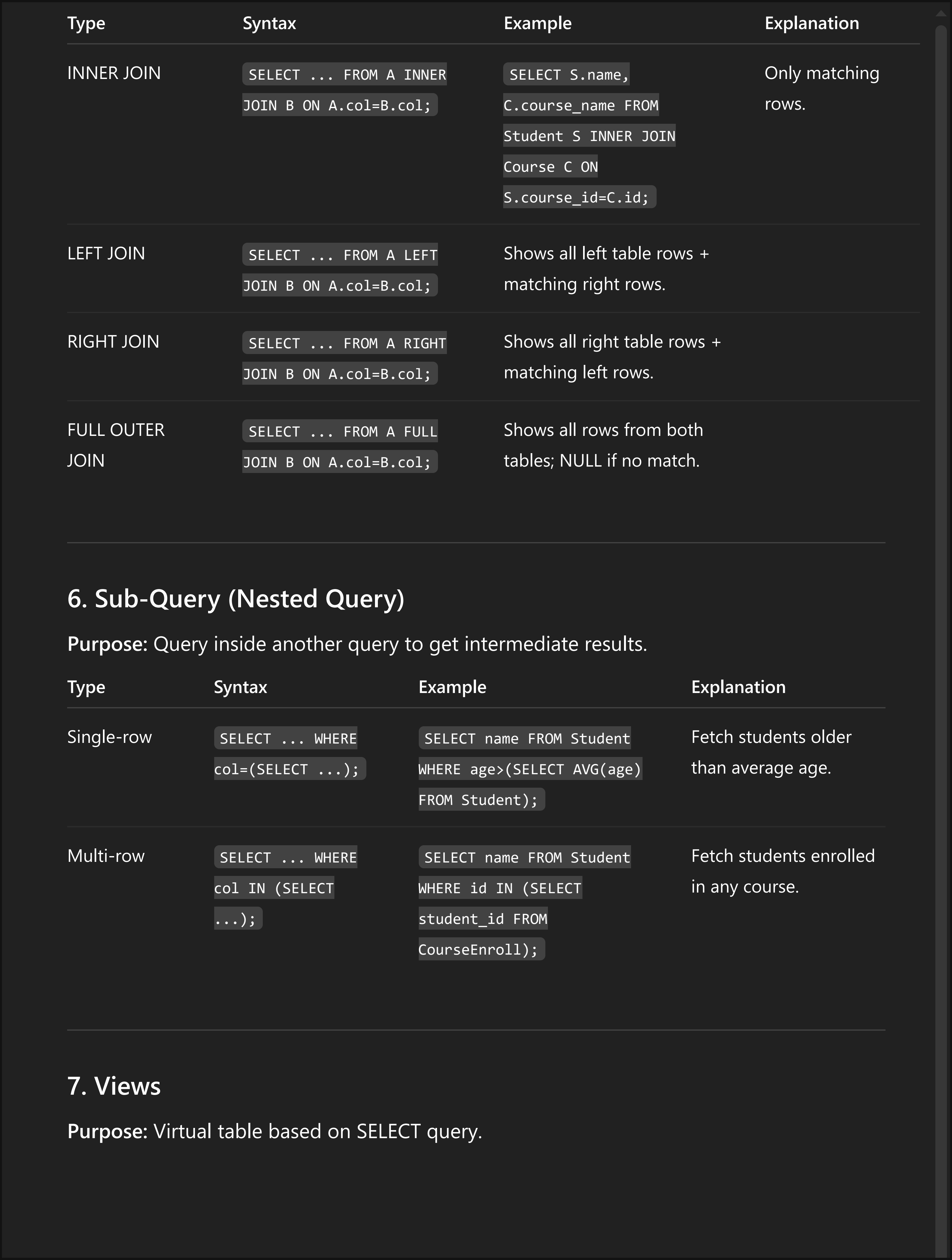
4. Set Operators

Operator	Syntax	Example	Explanation
UNION	`SELECT ... FROM ... UNION SELECT ...;`	`SELECT name FROM Student UNION SELECT name FROM Teacher;`	Combine two queries, remove duplicates.
UNION ALL	Same as UNION	Includes duplicates.	
INTERSECT	Same as above	Shows common data in both queries.	
MINUS/EXCEPT	Same	Shows data in first query but not in second.	

---

5. SQL Joins

Purpose: Combine rows from two or more tables based on related column.



Type	Syntax	Example	Explanation
INNER JOIN	`SELECT ... FROM A INNER JOIN B ON A.col=B.col;`	`SELECT S.name, C.course_name FROM Student S INNER JOIN Course C ON S.course_id=C.id;`	Only matching rows.
LEFT JOIN	`SELECT ... FROM A LEFT JOIN B ON A.col=B.col;`	Shows all left table rows + matching right rows.	
RIGHT JOIN	`SELECT ... FROM A RIGHT JOIN B ON A.col=B.col;`	Shows all right table rows + matching left rows.	
FULL OUTER JOIN	`SELECT ... FROM A FULL JOIN B ON A.col=B.col;`	Shows all rows from both tables; NULL if no match.	

---

6. Sub-Query (Nested Query)

Purpose: Query inside another query to get intermediate results.

Type	Syntax	Example	Explanation
Single-row	`SELECT ... WHERE col=(SELECT ...);`	`SELECT name FROM Student WHERE age>(SELECT AVG(age) FROM Student);`	Fetch students older than average age.
Multi-row	`SELECT ... WHERE col IN (SELECT ...);`	`SELECT name FROM Student WHERE id IN (SELECT student_id FROM CourseEnroll);`	Fetch students enrolled in any course.

---

7. Views

Purpose: Virtual table based on SELECT query.

Operation	Syntax	Example	Explanation
CREATE VIEW	`CREATE VIEW view_name AS SELECT ...;`	`CREATE VIEW HighSalary AS SELECT name, salary FROM Employee WHERE salary>2000;`	Virtual table of high salary employees.
SELECT VIEW	`SELECT * FROM view_name;`	`SELECT * FROM HighSalary;`	Fetch data from view.
DROP VIEW	`DROP VIEW view_name;`	`DROP VIEW HighSalary;`	Remove view.

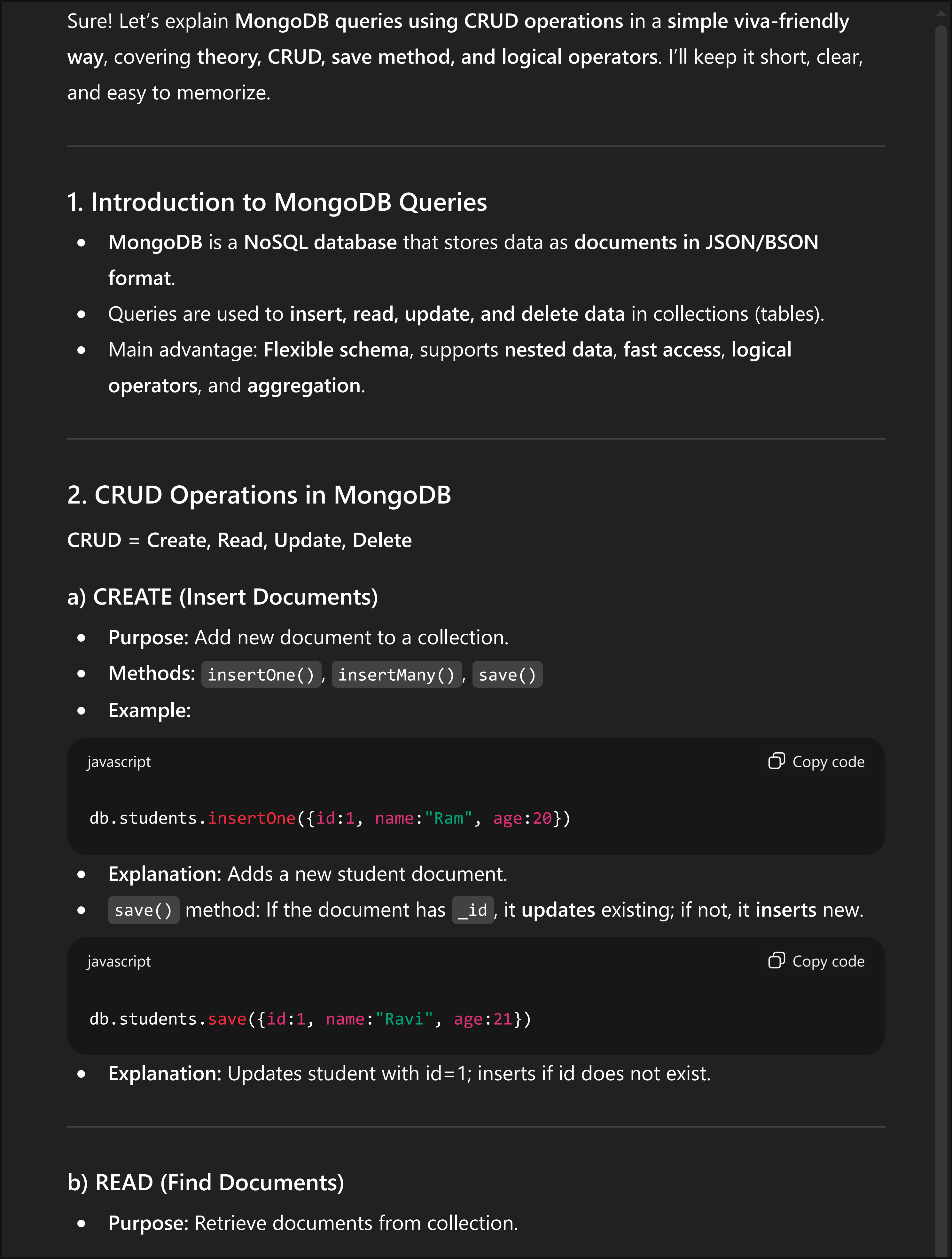
✅ This **one-page cheat sheet** covers:

- Insert, Select, Update, Delete
- Operators & Sorting
- Aggregate & Scalar functions
- Set operations (UNION, INTERSECT, MINUS)
- Joins (Inner, Left, Right, Full)
- Sub-queries (Single-row, Multi-row)
- Views (Create, Select, Drop)

If you want, I can also **make a super-condensed version** as a **2-column table with only syntax + 1-line example** so you can **glance through all SQL topics in 5 minutes** before viva.

Do you want me to do that?

explain their theory in simple way MongoDB Queries using CRUD operations. (Use CRUDoperations, SAVE method, logical operators etc)



Sure! Let's explain **MongoDB queries using CRUD operations** in a **simple viva-friendly way**, covering **theory, CRUD, save method, and logical operators**. I'll keep it short, clear, and easy to memorize.

---

1. Introduction to MongoDB Queries

- **MongoDB** is a **NoSQL database** that stores data as **documents in JSON/BSON format**.
- Queries are used to **insert, read, update, and delete data** in collections (tables).
- Main advantage: **Flexible schema**, supports **nested data**, **fast access**, **logical operators**, and **aggregation**.

---

2. CRUD Operations in MongoDB

CRUD = **Create, Read, Update, Delete**

a) CREATE (Insert Documents)

- **Purpose:** Add new document to a collection.
- **Methods:** `insertOne()`, `insertMany()`, `save()`
- **Example:**

```javascript
db.students.insertOne({id:1, name:"Ram", age:20})
```

- **Explanation:** Adds a new student document.
- `save()` method: If the document has `_id`, it **updates** existing; if not, it **inserts** new.

```javascript
db.students.save({id:1, name:"Ravi", age:21})
```

- **Explanation:** Updates student with id=1; inserts if id does not exist.

---

b) READ (Find Documents)

- **Purpose:** Retrieve documents from collection.

- **Method:** `find()`, `findOne()`
- **Example:**

```javascript
db.students.find({age:20})
db.students.findOne({name:"Ram"})
```

- **Explanation:** Finds all students with age 20; or the first student named Ram.

Logical Operators in Queries:

- `$and`, `$or`, `$not`, `$gt`, `$lt`, `$gte`, `$lte`, `$ne`
- **Example:**

```javascript
db.students.find({$and:[{age:{$gte:18}}, {age:{$lte:22}}]})
```

- **Explanation:** Finds students between age 18 and 22.

---

c) UPDATE (Modify Documents)

- **Purpose:** Change existing data.
- **Methods:** `updateOne()`, `updateMany()`, `replaceOne()`
- **Example:**

```javascript
db.students.updateOne({id:1}, {$set:{name:"Ravi"}})
```

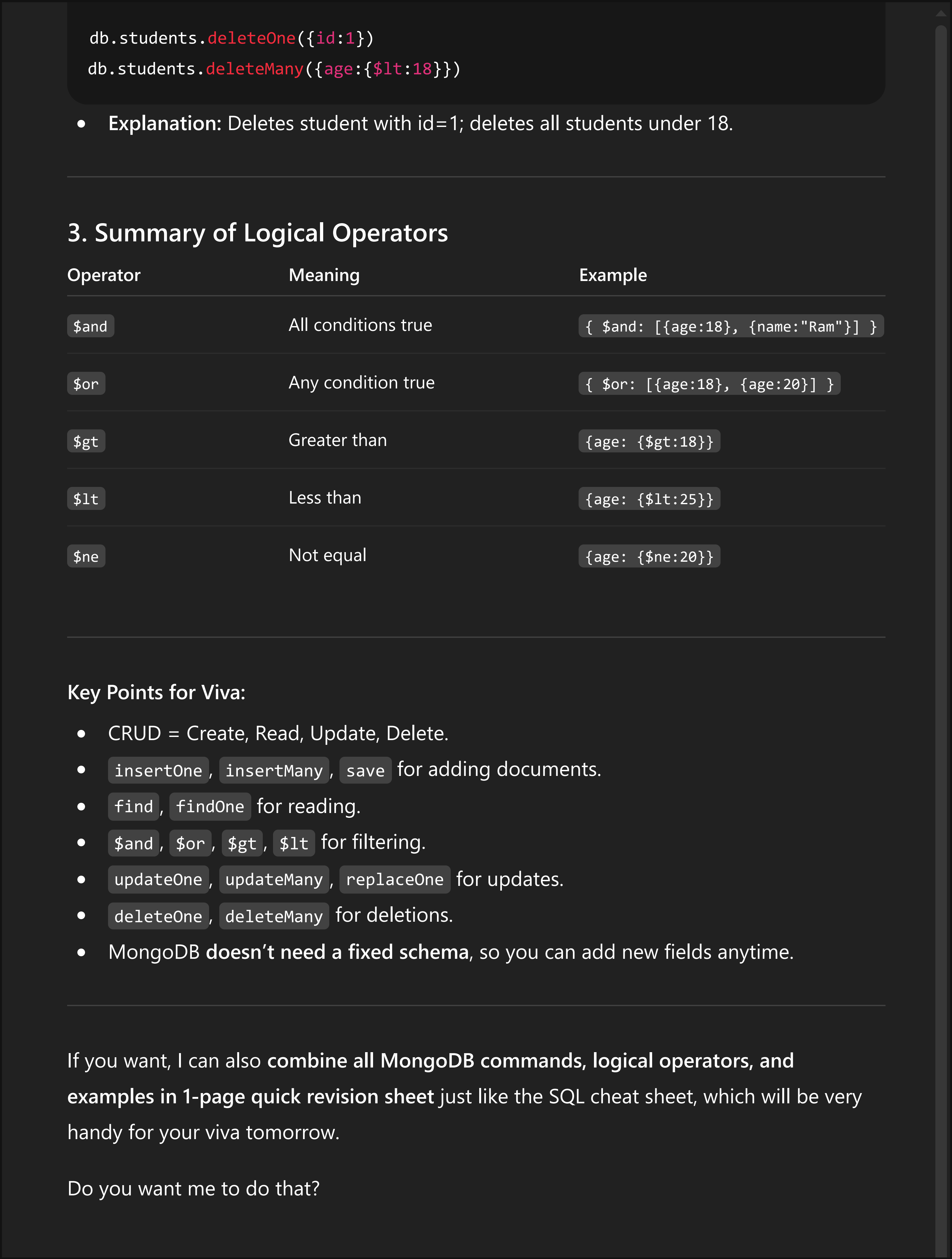
- **Explanation:** Updates name to Ravi for student with id=1.

---

d) DELETE (Remove Documents)

- **Purpose:** Remove documents from collection.
- **Methods:** `deleteOne()`, `deleteMany()`
- **Example:**

```javascript
```



```
db.students.deleteOne({id:1})
db.students.deleteMany({age:{$lt:18}})
```

- **Explanation:** Deletes student with id=1; deletes all students under 18.

---

3. Summary of Logical Operators

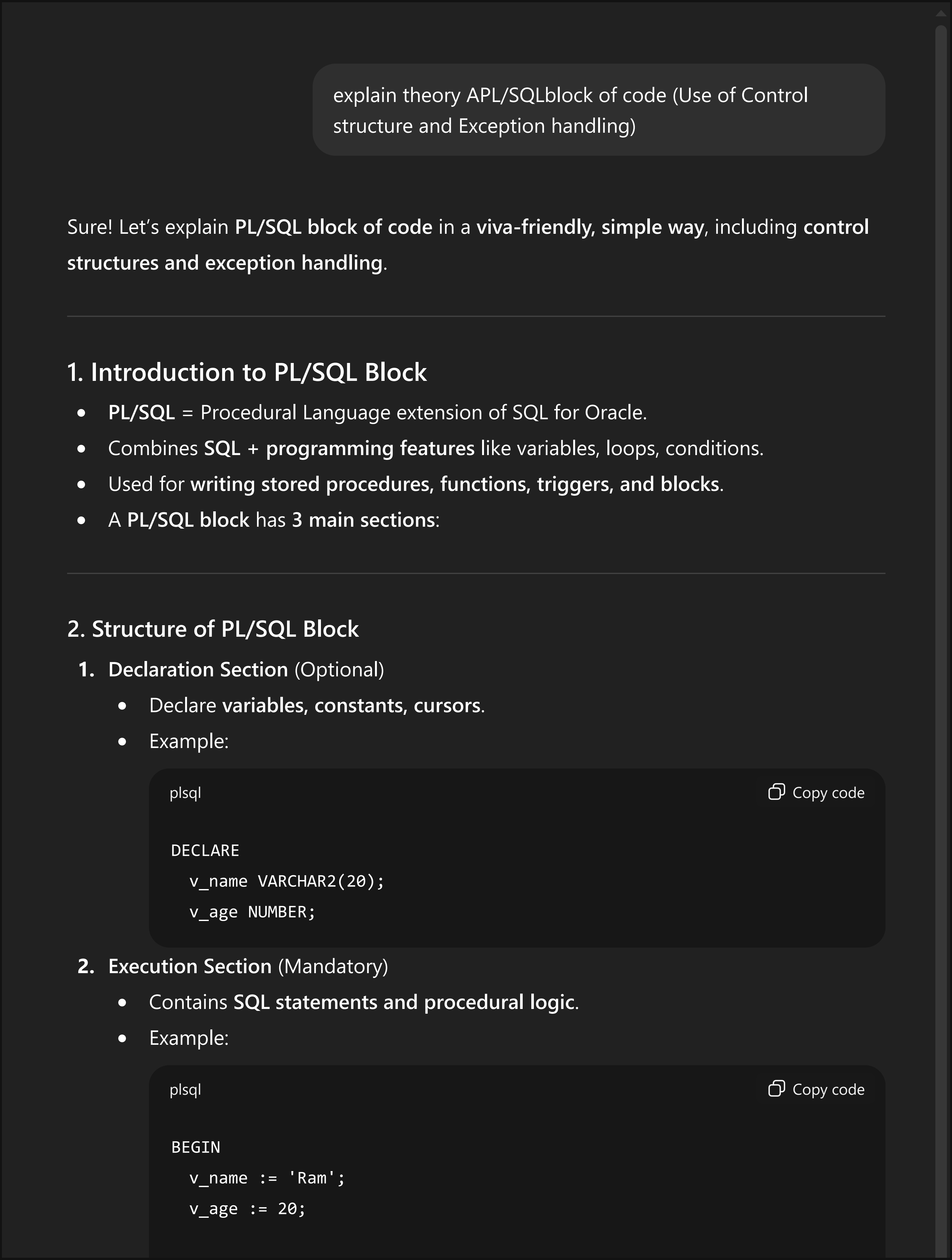
Operator	Meaning	Example
`$and`	All conditions true	`{ $and: [{age:18}, {name:"Ram"}] }`
`$or`	Any condition true	`{ $or: [{age:18}, {age:20}] }`
`$gt`	Greater than	`{age: {$gt:18}}`
`$lt`	Less than	`{age: {$lt:25}}`
`$ne`	Not equal	`{age: {$ne:20}}`

Key Points for Viva:

- CRUD = Create, Read, Update, Delete.
- `insertOne`, `insertMany`, `save` for adding documents.
- `find`, `findOne` for reading.
- `$and`, `$or`, `$gt`, `$lt` for filtering.
- `updateOne`, `updateMany`, `replaceOne` for updates.
- `deleteOne`, `deleteMany` for deletions.
- MongoDB **doesn't need a fixed schema**, so you can add new fields anytime.

If you want, I can also **combine all MongoDB commands, logical operators, and examples in 1-page quick revision sheet** just like the SQL cheat sheet, which will be very handy for your viva tomorrow.

Do you want me to do that?



explain theory APL/SQLblock of code (Use of Control structure and Exception handling)

Sure! Let's explain **PL/SQL block of code** in a **viva-friendly, simple way**, including **control structures and exception handling**.

---

1. Introduction to PL/SQL Block

- **PL/SQL** = Procedural Language extension of SQL for Oracle.
- Combines **SQL** + **programming features** like variables, loops, conditions.
- Used for **writing stored procedures, functions, triggers, and blocks**.
- A **PL/SQL block** has **3 main sections**:

---

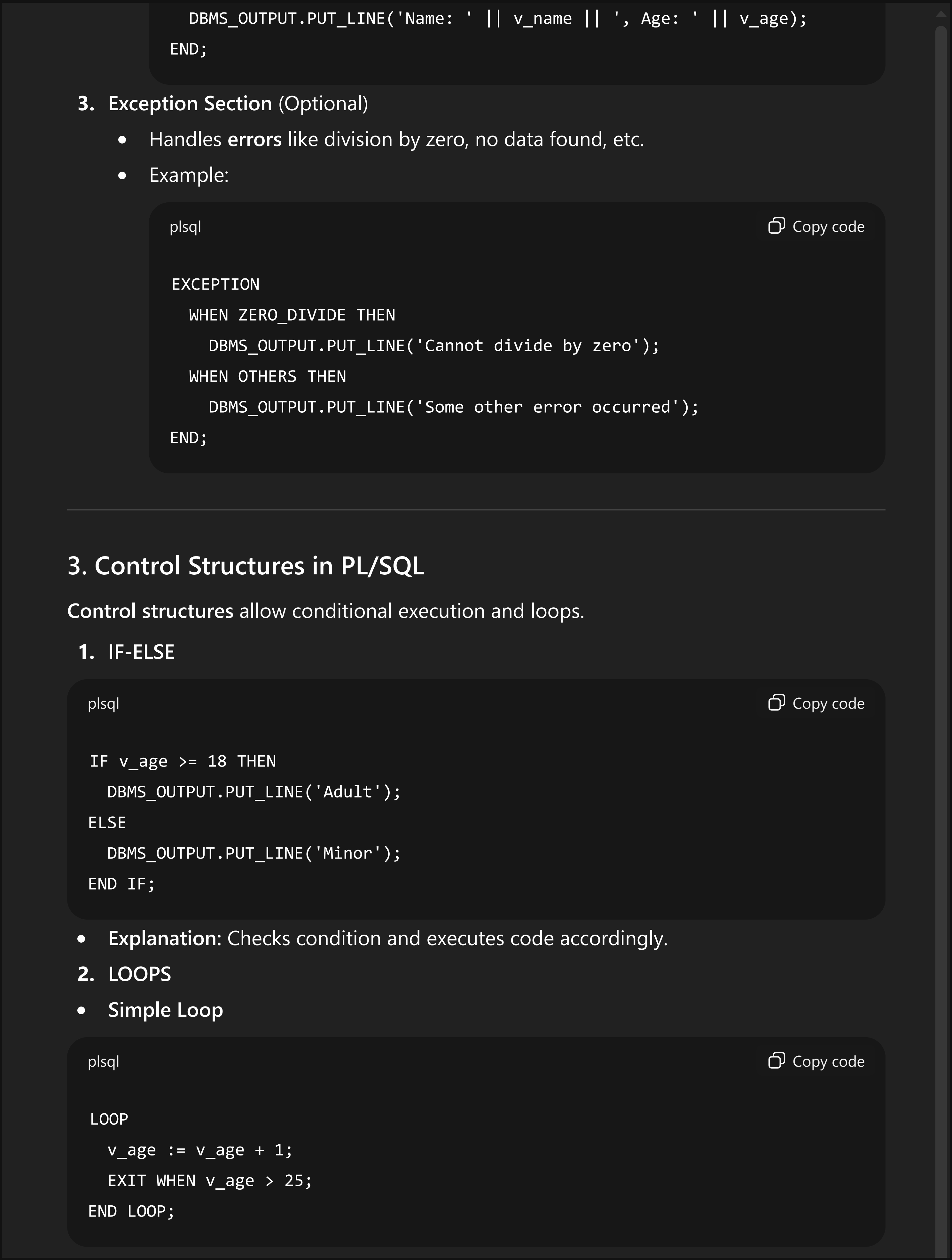
2. Structure of PL/SQL Block

1. **Declaration Section** (Optional)
    - Declare **variables, constants, cursors**.
    - Example:

```plsql
DECLARE
  v_name VARCHAR2(20);
  v_age NUMBER;
```

2. **Execution Section** (Mandatory)
    - Contains **SQL statements and procedural logic**.
    - Example:

```plsql
BEGIN
  v_name := 'Ram';
  v_age := 20;
```



```plsql
  DBMS_OUTPUT.PUT_LINE('Name: ' || v_name || ', Age: ' || v_age);
END;
```

3. **Exception Section** (Optional)
   - Handles **errors** like division by zero, no data found, etc.
   - Example:

```plsql
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Cannot divide by zero');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Some other error occurred');
END;
```

---

3. Control Structures in PL/SQL

Control structures allow conditional execution and loops.

1. **IF-ELSE**

```plsql
IF v_age >= 18 THEN
  DBMS_OUTPUT.PUT_LINE('Adult');
ELSE
  DBMS_OUTPUT.PUT_LINE('Minor');
END IF;
```

- **Explanation:** Checks condition and executes code accordingly.

2. **LOOPS**

- **Simple Loop**

```plsql
LOOP
  v_age := v_age + 1;
  EXIT WHEN v_age > 25;
END LOOP;
```

- **WHILE Loop**

```plsql
WHILE v_age <= 25 LOOP
  DBMS_OUTPUT.PUT_LINE(v_age);
  v_age := v_age + 1;
END LOOP;
```

- **FOR Loop**

```plsql
FOR i IN 1..5 LOOP
  DBMS_OUTPUT.PUT_LINE('Count: ' || i);
END LOOP;
```

---

4. Exception Handling in PL/SQL

- **Purpose:** Handle runtime errors gracefully.

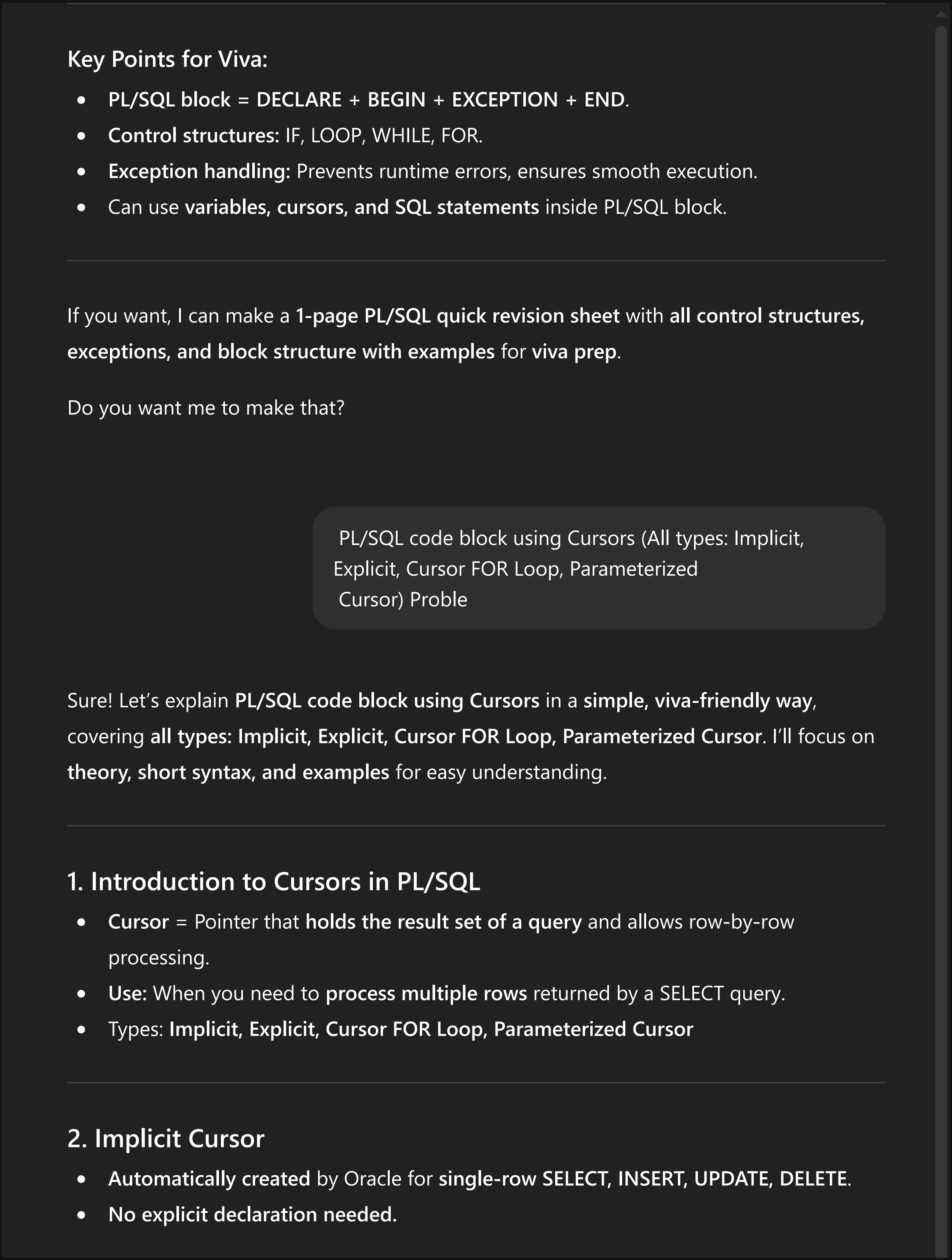
Common exceptions:

- `NO_DATA_FOUND` → Query returns no row.
- `TOO_MANY_ROWS` → Query returns multiple rows.
- `ZERO_DIVIDE` → Division by zero.
- `OTHERS` → Catch all other errors.

Example:

```plsql
BEGIN
  v_result := 10 / 0;
EXCEPTION
  WHEN ZERO_DIVIDE THEN
    DBMS_OUTPUT.PUT_LINE('Cannot divide by zero');
  WHEN OTHERS THEN
    DBMS_OUTPUT.PUT_LINE('Some error occurred');
END;
```

- **Explanation:** Prevents program crash; shows friendly error message.



Key Points for Viva:

- **PL/SQL block = DECLARE + BEGIN + EXCEPTION + END**.
- **Control structures:** IF, LOOP, WHILE, FOR.
- **Exception handling:** Prevents runtime errors, ensures smooth execution.
- Can use **variables, cursors, and SQL statements** inside PL/SQL block.

If you want, I can make a **1-page PL/SQL quick revision sheet** with **all control structures, exceptions, and block structure with examples** for **viva prep**.

Do you want me to make that?

PL/SQL code block using Cursors (All types: Implicit,
Explicit, Cursor FOR Loop, Parameterized
Cursor) Proble

Sure! Let's explain **PL/SQL code block using Cursors** in a **simple, viva-friendly way**, covering **all types: Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor**. I'll focus on **theory, short syntax, and examples** for easy understanding.

---

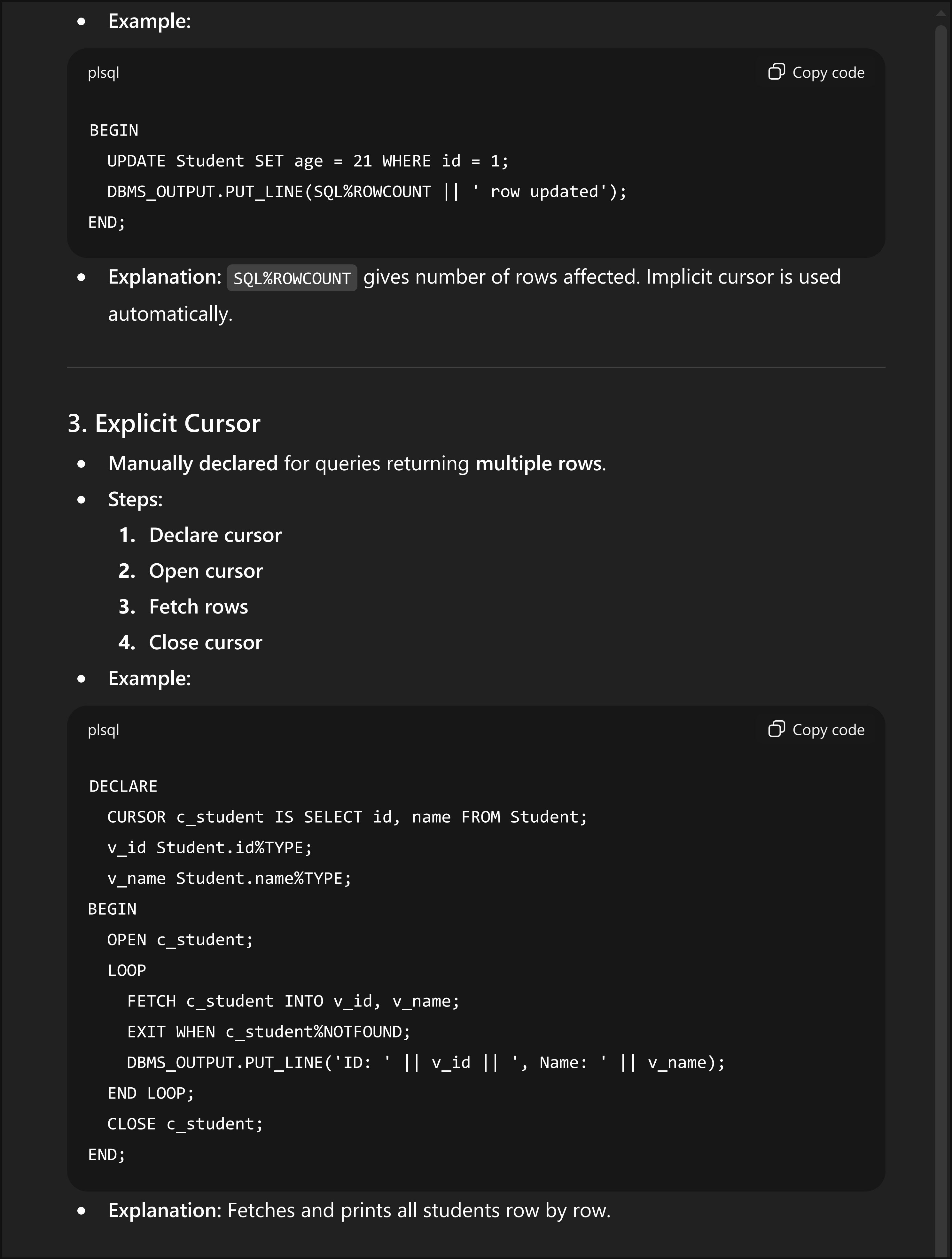
1. Introduction to Cursors in PL/SQL

- **Cursor** = Pointer that **holds the result set of a query** and allows row-by-row processing.
- **Use:** When you need to **process multiple rows** returned by a SELECT query.
- Types: **Implicit, Explicit, Cursor FOR Loop, Parameterized Cursor**

---

2. Implicit Cursor

- **Automatically created** by Oracle for **single-row SELECT, INSERT, UPDATE, DELETE**.
- **No explicit declaration needed.**



- **Example:**

```plsql
BEGIN
  UPDATE Student SET age = 21 WHERE id = 1;
  DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' row updated');
END;
```

- **Explanation:** `SQL%ROWCOUNT` gives number of rows affected. Implicit cursor is used automatically.

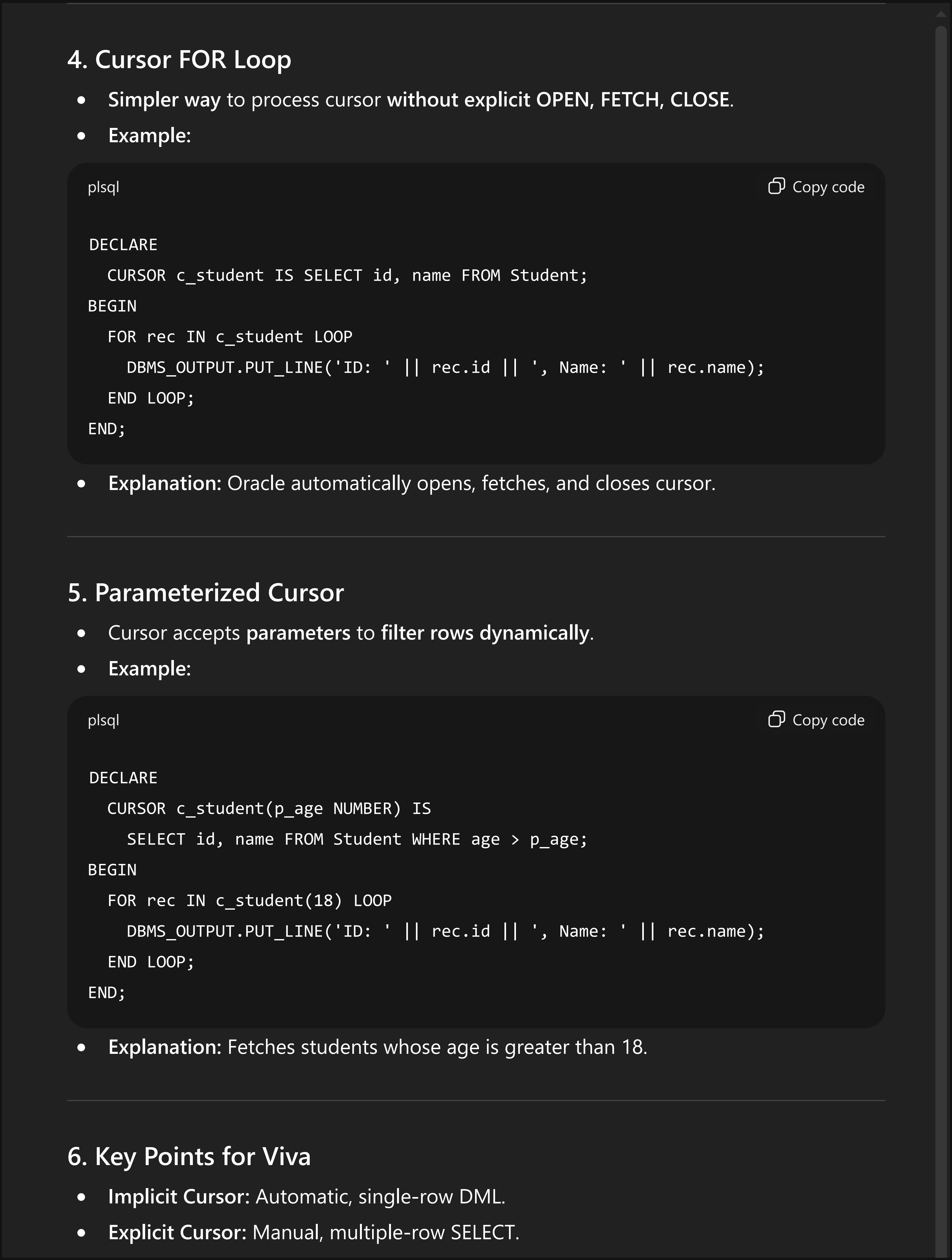
---

3. Explicit Cursor

- **Manually declared** for queries returning **multiple rows**.
- **Steps:**
  1. Declare cursor
  2. Open cursor
  3. Fetch rows
  4. Close cursor
- **Example:**

```plsql
DECLARE
  CURSOR c_student IS SELECT id, name FROM Student;
  v_id Student.id%TYPE;
  v_name Student.name%TYPE;
BEGIN
  OPEN c_student;
  LOOP
    FETCH c_student INTO v_id, v_name;
    EXIT WHEN c_student%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('ID: ' || v_id || ', Name: ' || v_name);
  END LOOP;
  CLOSE c_student;
END;
```

- **Explanation:** Fetches and prints all students row by row.



4. Cursor FOR Loop

- **Simpler way** to process cursor **without explicit OPEN, FETCH, CLOSE**.
- **Example:**

```plsql
DECLARE
  CURSOR c_student IS SELECT id, name FROM Student;
BEGIN
  FOR rec IN c_student LOOP
    DBMS_OUTPUT.PUT_LINE('ID: ' || rec.id || ', Name: ' || rec.name);
  END LOOP;
END;
```

- **Explanation:** Oracle automatically opens, fetches, and closes cursor.

---

5. Parameterized Cursor

- Cursor accepts **parameters** to **filter rows dynamically**.
- **Example:**

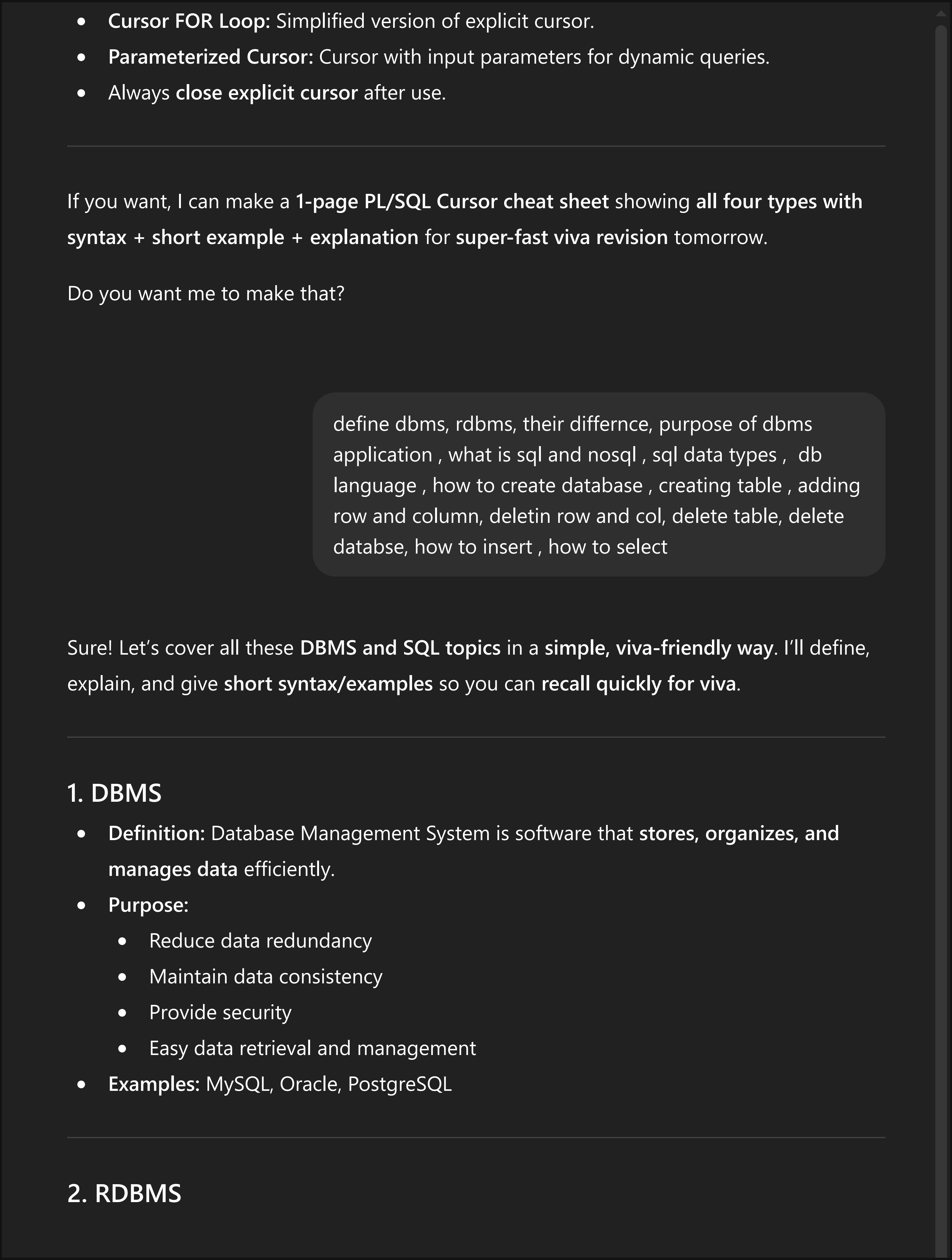
```plsql
DECLARE
  CURSOR c_student(p_age NUMBER) IS
    SELECT id, name FROM Student WHERE age > p_age;
BEGIN
  FOR rec IN c_student(18) LOOP
    DBMS_OUTPUT.PUT_LINE('ID: ' || rec.id || ', Name: ' || rec.name);
  END LOOP;
END;
```

- **Explanation:** Fetches students whose age is greater than 18.

---

6. Key Points for Viva

- **Implicit Cursor:** Automatic, single-row DML.
- **Explicit Cursor:** Manual, multiple-row SELECT.



- **Cursor FOR Loop:** Simplified version of explicit cursor.
- **Parameterized Cursor:** Cursor with input parameters for dynamic queries.
- Always **close explicit cursor** after use.

If you want, I can make a **1-page PL/SQL Cursor cheat sheet** showing **all four types with syntax + short example + explanation** for **super-fast viva revision** tomorrow.

Do you want me to make that?

define dbms, rdbms, their differnce, purpose of dbms application , what is sql and nosql , sql data types , db language , how to create database , creating table , adding row and column, deletin row and col, delete table, delete databse, how to insert , how to select

Sure! Let's cover all these **DBMS and SQL topics** in a **simple, viva-friendly way**. I'll define, explain, and give **short syntax/examples** so you can **recall quickly for viva**.

---

1. DBMS

- **Definition:** Database Management System is software that **stores, organizes, and manages data** efficiently.
- **Purpose:**
  - Reduce data redundancy
  - Maintain data consistency
  - Provide security
  - Easy data retrieval and management
- **Examples:** MySQL, Oracle, PostgreSQL

---

2. RDBMS

- **Definition:** Relational DBMS stores data in **tables (rows & columns)** and supports **relationships between tables**.
- **Examples:** MySQL, Oracle, SQL Server
- **Difference DBMS vs RDBMS:**

Feature	DBMS	RDBMS
Data Storage	File-oriented	Table-oriented
Relationships	Not supported	Supported via foreign key
Normalization	No	Yes
Redundancy	More	Less
Examples	MS Access	MySQL, Oracle

---

3. Database Applications

- **Applications:** Banking system, Online Shopping, University, Hospital Management, Airline Reservation
- **Purpose:** Efficient data handling, reduce manual work, ensure integrity and security

---

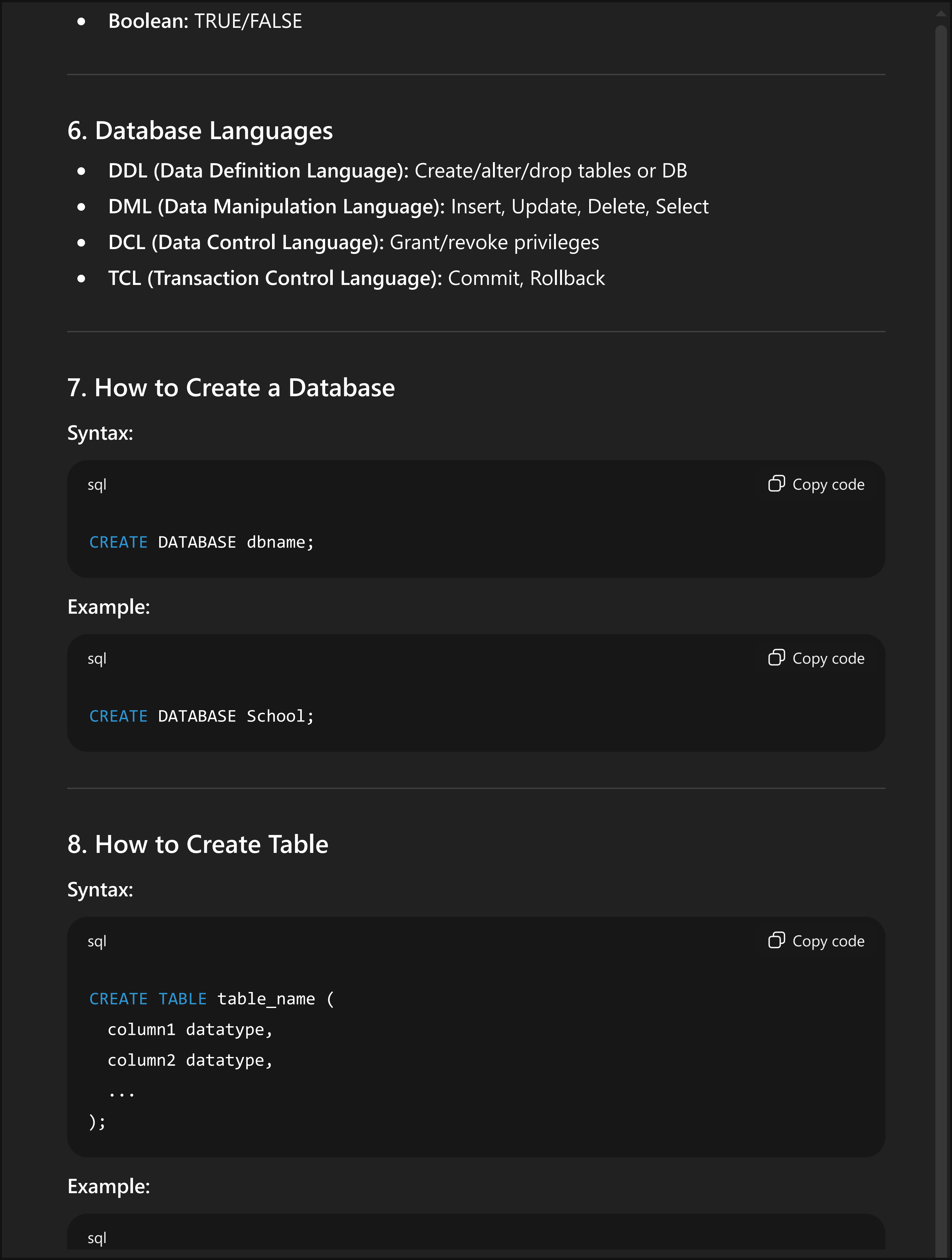
4. SQL and NoSQL

- **SQL (Structured Query Language):** Language used to **create, manage, and retrieve data in relational databases**.
- **NoSQL (Not Only SQL):** Non-relational database used for **big data & flexible schema** (JSON, key-value, document store, etc.)
- **Difference:** SQL = structured, schema-based; NoSQL = flexible, scalable

---

5. SQL Data Types

- **Numeric:** INT, FLOAT, DECIMAL
- **Character:** CHAR(size), VARCHAR(size), TEXT
- **Date/Time:** DATE, TIME, DATETIME, TIMESTAMP



- **Boolean:** TRUE/FALSE

---

6. Database Languages

- **DDL (Data Definition Language):** Create/alter/drop tables or DB
- **DML (Data Manipulation Language):** Insert, Update, Delete, Select
- **DCL (Data Control Language):** Grant/revoke privileges
- **TCL (Transaction Control Language):** Commit, Rollback

---

7. How to Create a Database

Syntax:

```sql
CREATE DATABASE dbname;
```

Example:

```sql
CREATE DATABASE School;
```

---

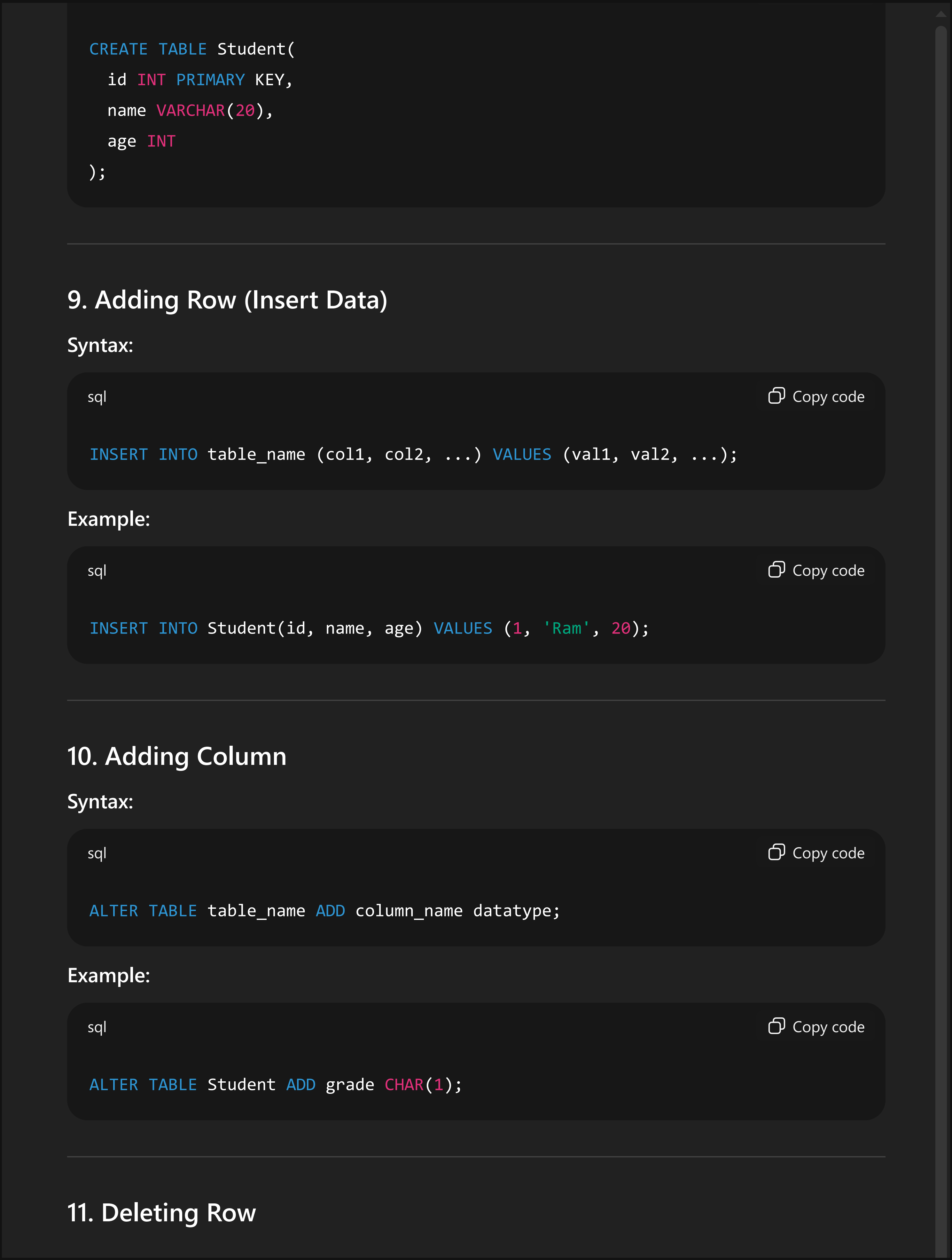
8. How to Create Table

Syntax:

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

Example:

sql



```sql
CREATE TABLE Student(
  id INT PRIMARY KEY,
  name VARCHAR(20),
  age INT
);
```

---

9. Adding Row (Insert Data)

Syntax:

```sql
INSERT INTO table_name (col1, col2, ...) VALUES (val1, val2, ...);
```

Example:

```sql
INSERT INTO Student(id, name, age) VALUES (1, 'Ram', 20);
```

---

10. Adding Column

Syntax:

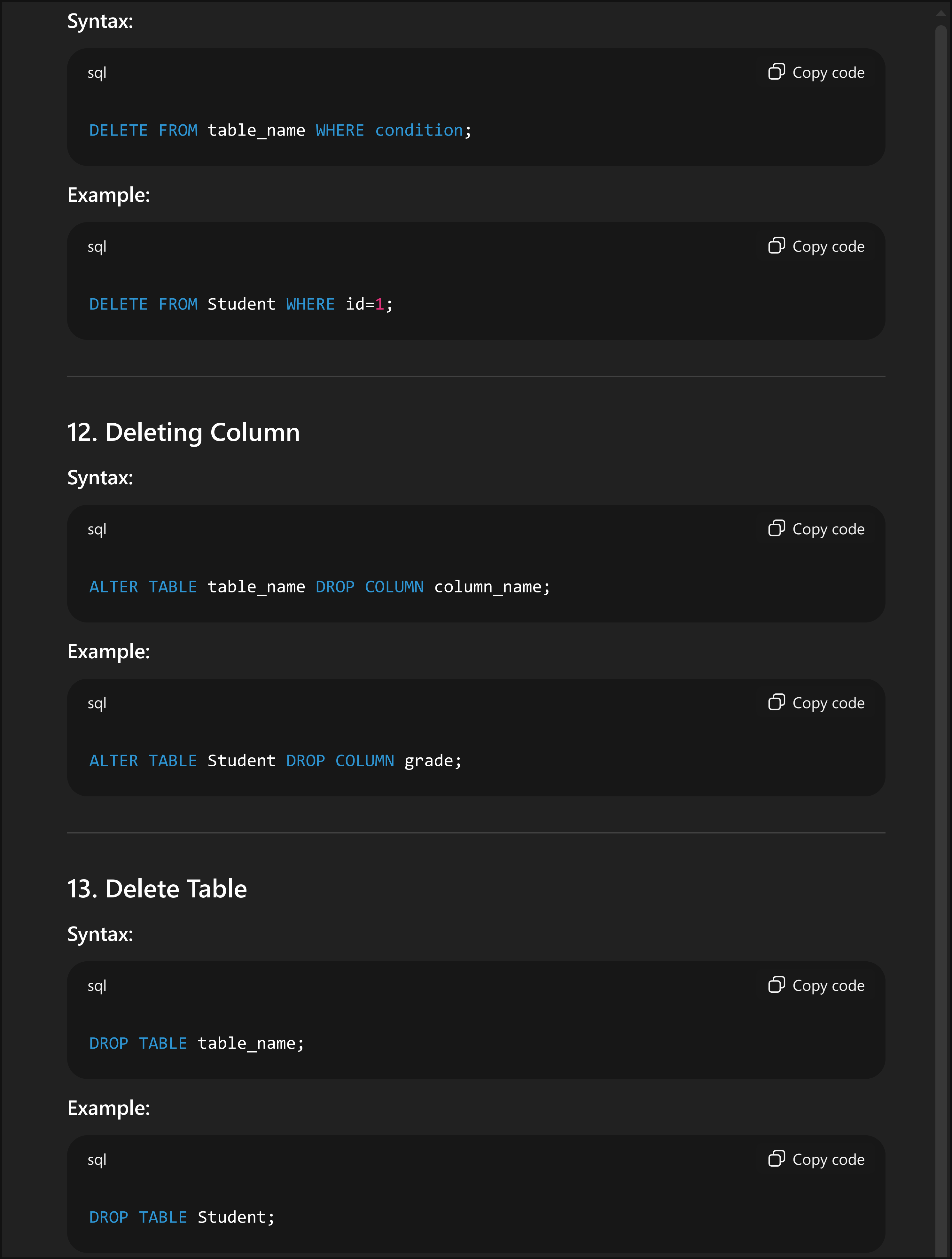
```sql
ALTER TABLE table_name ADD column_name datatype;
```

Example:

```sql
ALTER TABLE Student ADD grade CHAR(1);
```

---

11. Deleting Row



Syntax:

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

Example:

```sql
DELETE FROM Student WHERE id=1;
```

---

12. Deleting Column

Syntax:

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

Example:

```sql
ALTER TABLE Student DROP COLUMN grade;
```

---

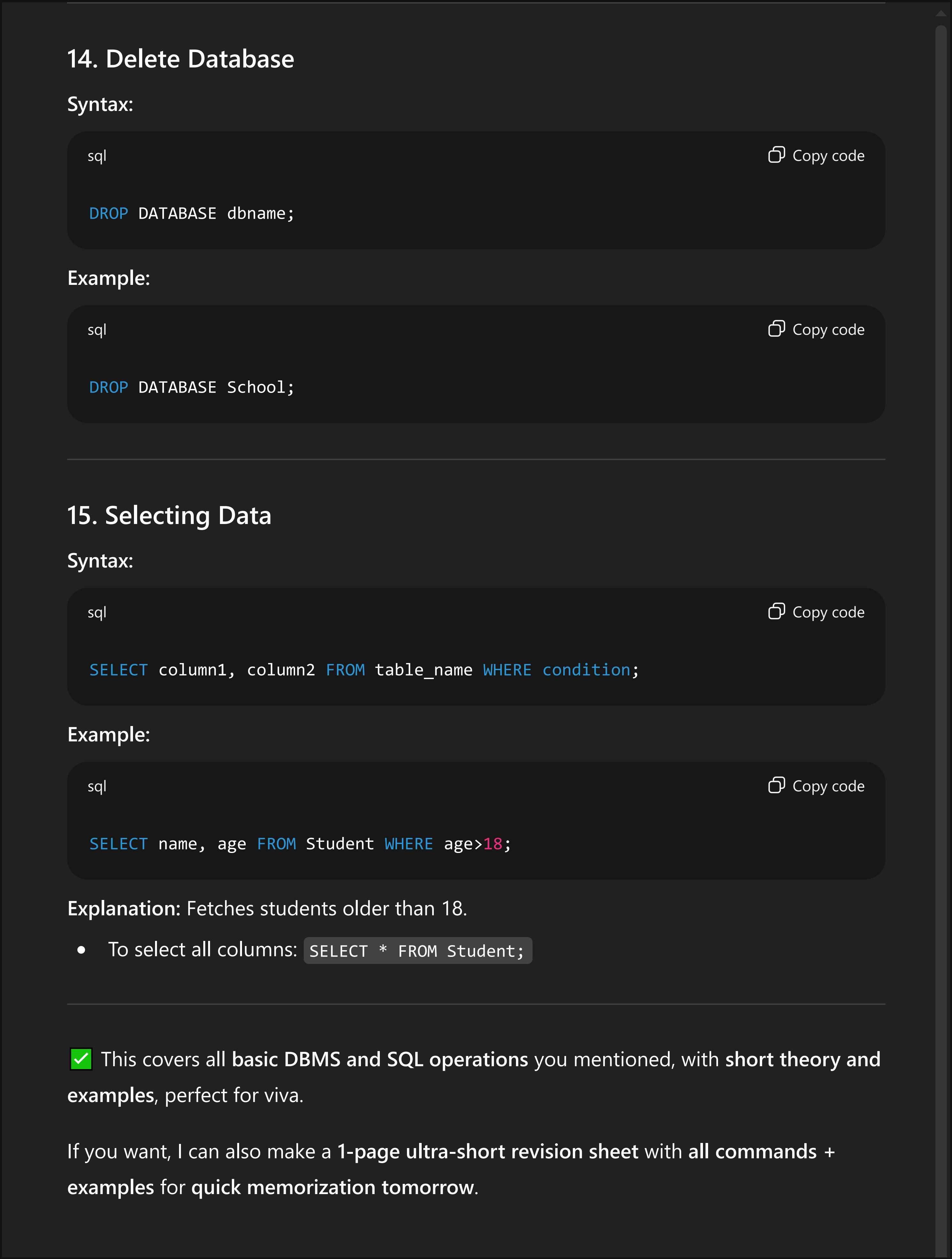
13. Delete Table

Syntax:

```sql
DROP TABLE table_name;
```

Example:

```sql
DROP TABLE Student;
```



14. Delete Database

Syntax:

```sql
DROP DATABASE dbname;
```

Example:

```sql
DROP DATABASE School;
```

---

15. Selecting Data

Syntax:

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

Example:

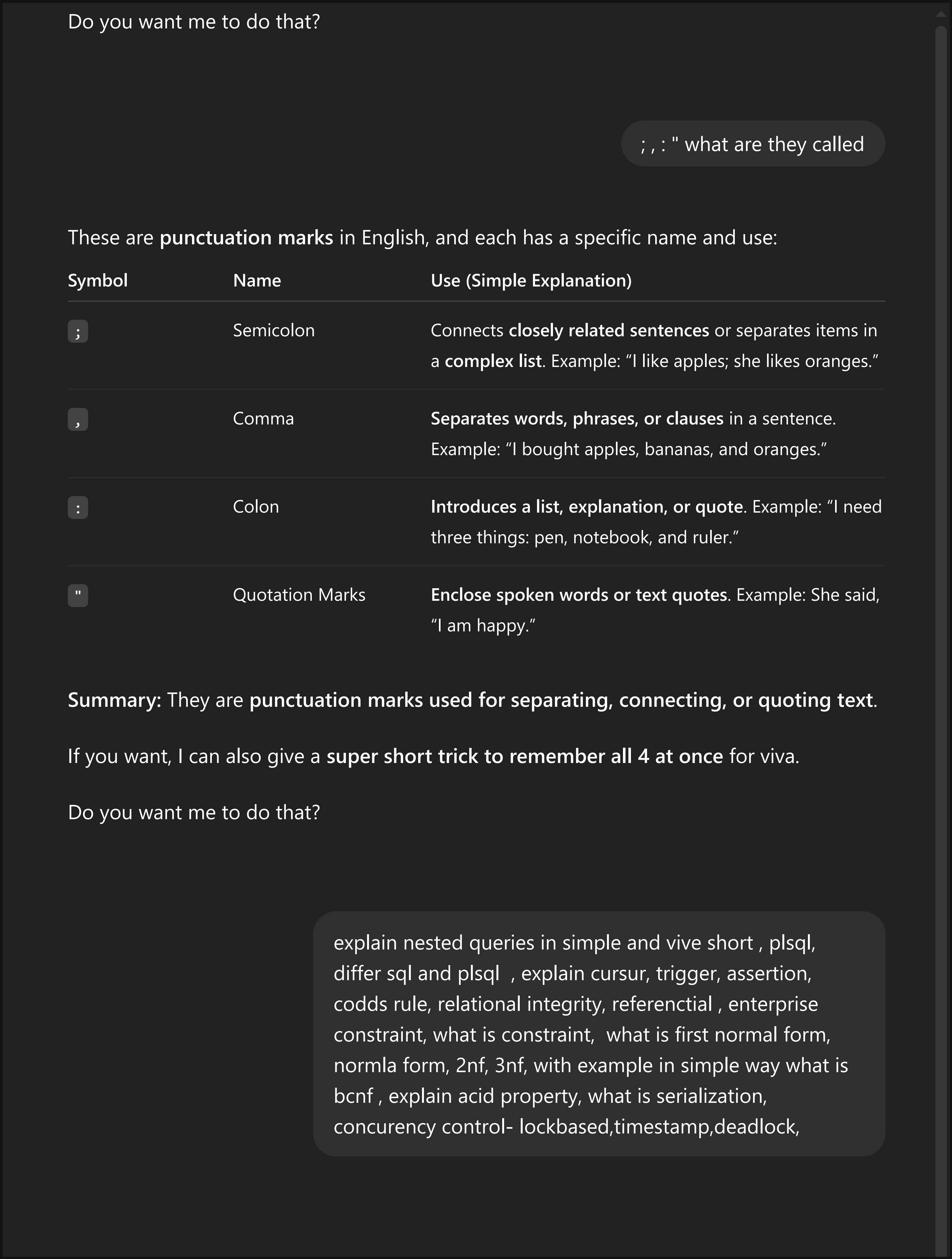
```sql
SELECT name, age FROM Student WHERE age>18;
```

Explanation: Fetches students older than 18.

- To select all columns: `SELECT * FROM Student;`

✅ This covers all **basic DBMS and SQL operations** you mentioned, with **short theory and examples**, perfect for viva.

If you want, I can also make a **1-page ultra-short revision sheet** with **all commands + examples** for **quick memorization tomorrow**.



Do you want me to do that?

; , : " what are they called

These are **punctuation marks** in English, and each has a specific name and use:

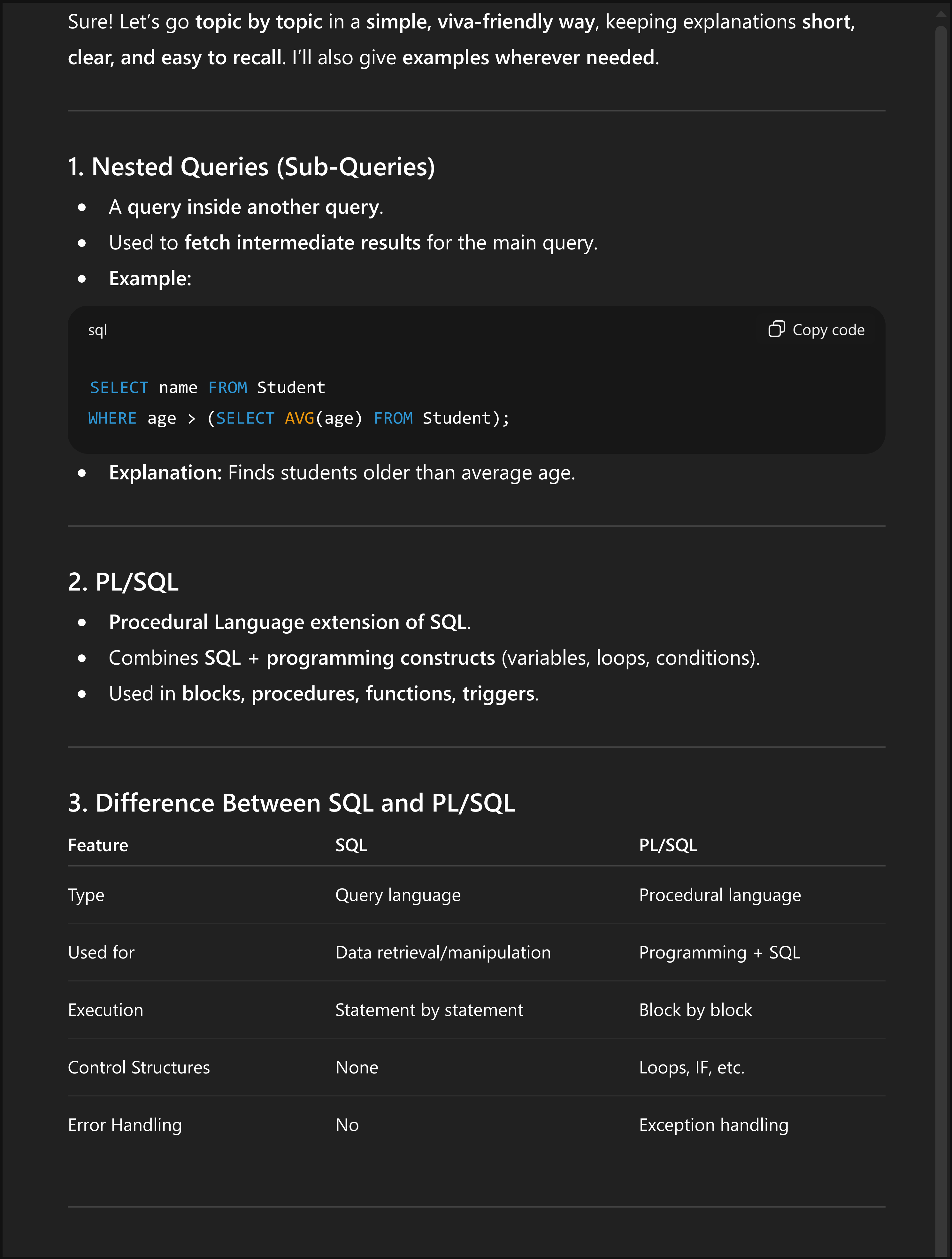
Symbol	**Name**	**Use (Simple Explanation)**
`;`	Semicolon	Connects **closely related sentences** or separates items in a **complex list**. Example: “I like apples; she likes oranges.”
`,`	Comma	**Separates words, phrases, or clauses** in a sentence. Example: “I bought apples, bananas, and oranges.”
`:`	Colon	**Introduces a list, explanation, or quote**. Example: “I need three things: pen, notebook, and ruler.”
`"`	Quotation Marks	**Enclose spoken words or text quotes**. Example: She said, “I am happy.”

Summary: They are **punctuation marks used for separating, connecting, or quoting text**.

If you want, I can also give a **super short trick to remember all 4 at once** for viva.

Do you want me to do that?

explain nested queries in simple and vive short , plsql, differ sql and plsql , explain cursur, trigger, assertion, codds rule, relational integrity, referenctial , enterprise constraint, what is constraint, what is first normal form, normla form, 2nf, 3nf, with example in simple way what is bcnf , explain acid property, what is serialization, concurency control- lockbased,timestamp,deadlock,



Sure! Let's go **topic by topic** in a **simple, viva-friendly way**, keeping explanations **short, clear, and easy to recall**. I'll also give **examples wherever needed**.

---

1. Nested Queries (Sub-Queries)

- A **query inside another query**.
- Used to **fetch intermediate results** for the main query.
- **Example:**

```sql
SELECT name FROM Student
WHERE age > (SELECT AVG(age) FROM Student);
```

- **Explanation:** Finds students older than average age.

---

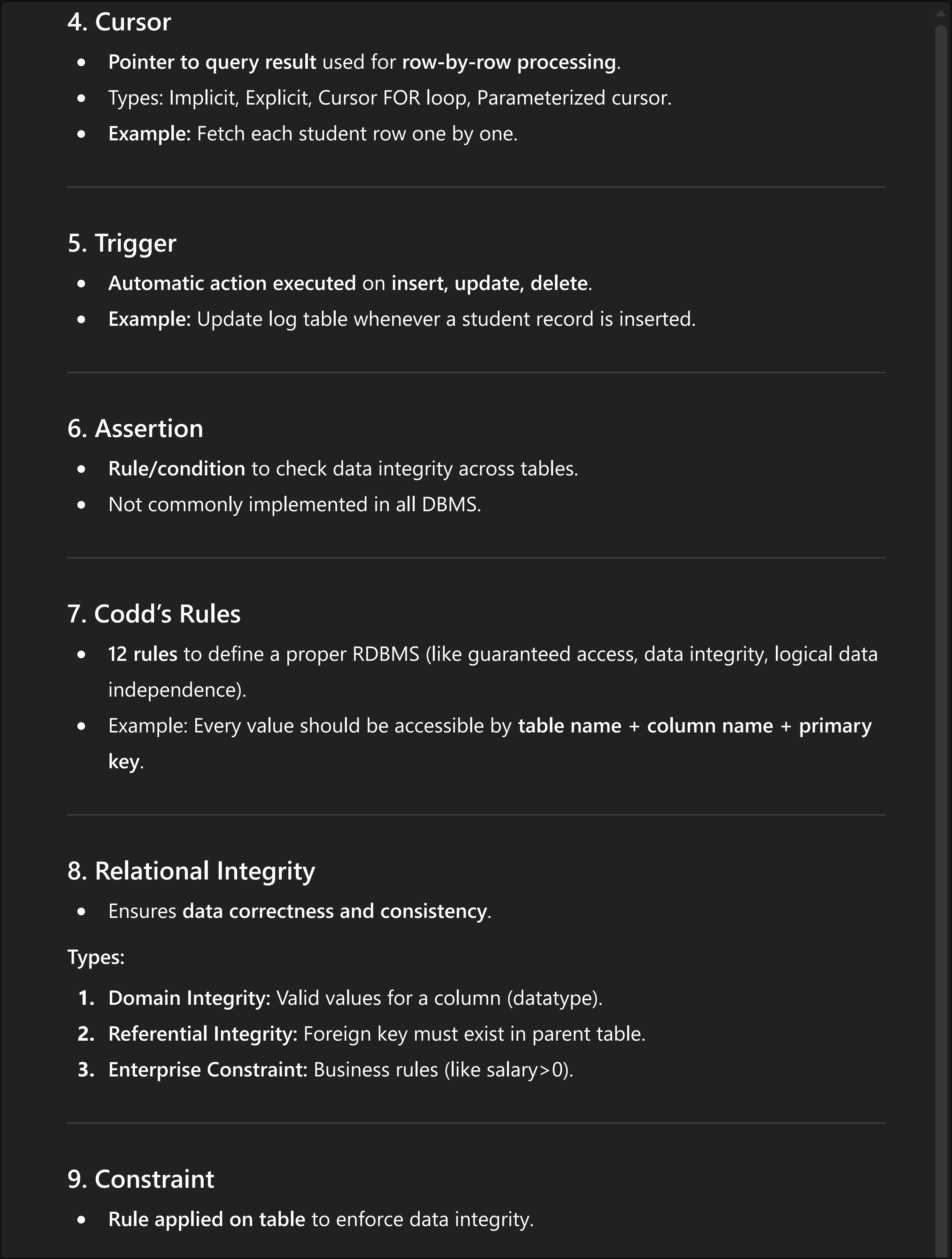
2. PL/SQL

- **Procedural Language extension of SQL**.
- Combines **SQL + programming constructs** (variables, loops, conditions).
- Used in **blocks, procedures, functions, triggers**.

---

3. Difference Between SQL and PL/SQL

Feature	SQL	PL/SQL
Type	Query language	Procedural language
Used for	Data retrieval/manipulation	Programming + SQL
Execution	Statement by statement	Block by block
Control Structures	None	Loops, IF, etc.
Error Handling	No	Exception handling



4. Cursor

- **Pointer to query result** used for **row-by-row processing**.
- Types: Implicit, Explicit, Cursor FOR loop, Parameterized cursor.
- **Example:** Fetch each student row one by one.

---

5. Trigger

- **Automatic action executed** on **insert, update, delete**.
- **Example:** Update log table whenever a student record is inserted.

---

6. Assertion

- **Rule/condition** to check data integrity across tables.
- Not commonly implemented in all DBMS.

---

7. Codd's Rules

- **12 rules** to define a proper RDBMS (like guaranteed access, data integrity, logical data independence).
- Example: Every value should be accessible by **table name + column name + primary key**.

---

8. Relational Integrity

- Ensures **data correctness and consistency**.

Types:

1. **Domain Integrity:** Valid values for a column (datatype).
2. **Referential Integrity:** Foreign key must exist in parent table.
3. **Enterprise Constraint:** Business rules (like salary>0).

---

9. Constraint

- **Rule applied on table** to enforce data integrity.

- **Examples:** PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK

---

10. Normalization & Normal Forms

Purpose: Remove redundancy & anomalies in tables.

1. **First Normal Form (1NF):**
- Columns contain **atomic values only**, no repeating groups.
- **Example:**
  | Student | Courses | → 1NF → separate each course in a row.
2. **Second Normal Form (2NF):**
- Must be in 1NF + **no partial dependency** on part of a composite primary key.
3. **Third Normal Form (3NF):**
- Must be in 2NF + **no transitive dependency** (non-key depends on another non-key).
4. **Boyce-Codd Normal Form (BCNF):**
- Stronger 3NF; **every determinant is a candidate key**.

---

11. ACID Properties

- Ensure reliable transactions:
1. **Atomicity:** All or nothing
2. **Consistency:** DB moves from one valid state to another
3. **Isolation:** Transactions do not interfere
4. **Durability:** Committed data is permanent

---

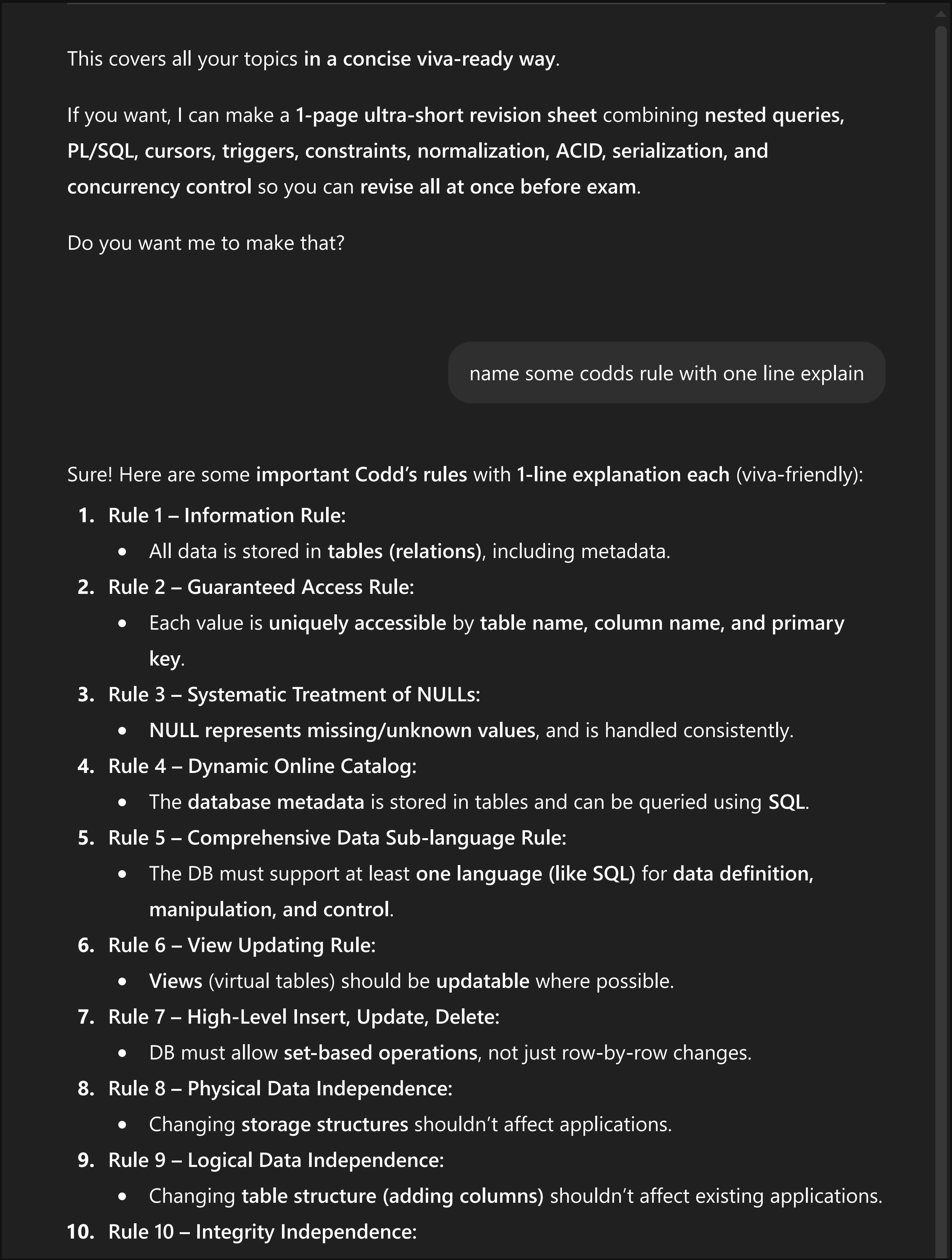
12. Serialization (Serializability)

- Ensures **transactions produce same result as if executed one by one**.
- Prevents conflicts in concurrent transactions.

---

13. Concurrency Control

1. **Lock-Based:** Transactions acquire locks (shared/exclusive) to prevent conflicts.
2. **Timestamp-Based:** Transactions ordered by timestamps to avoid conflicts.
3. **Deadlock Handling:** Detect & resolve deadlocks (e.g., wait-die, wound-wait).



This covers all your topics **in a concise viva-ready way**.

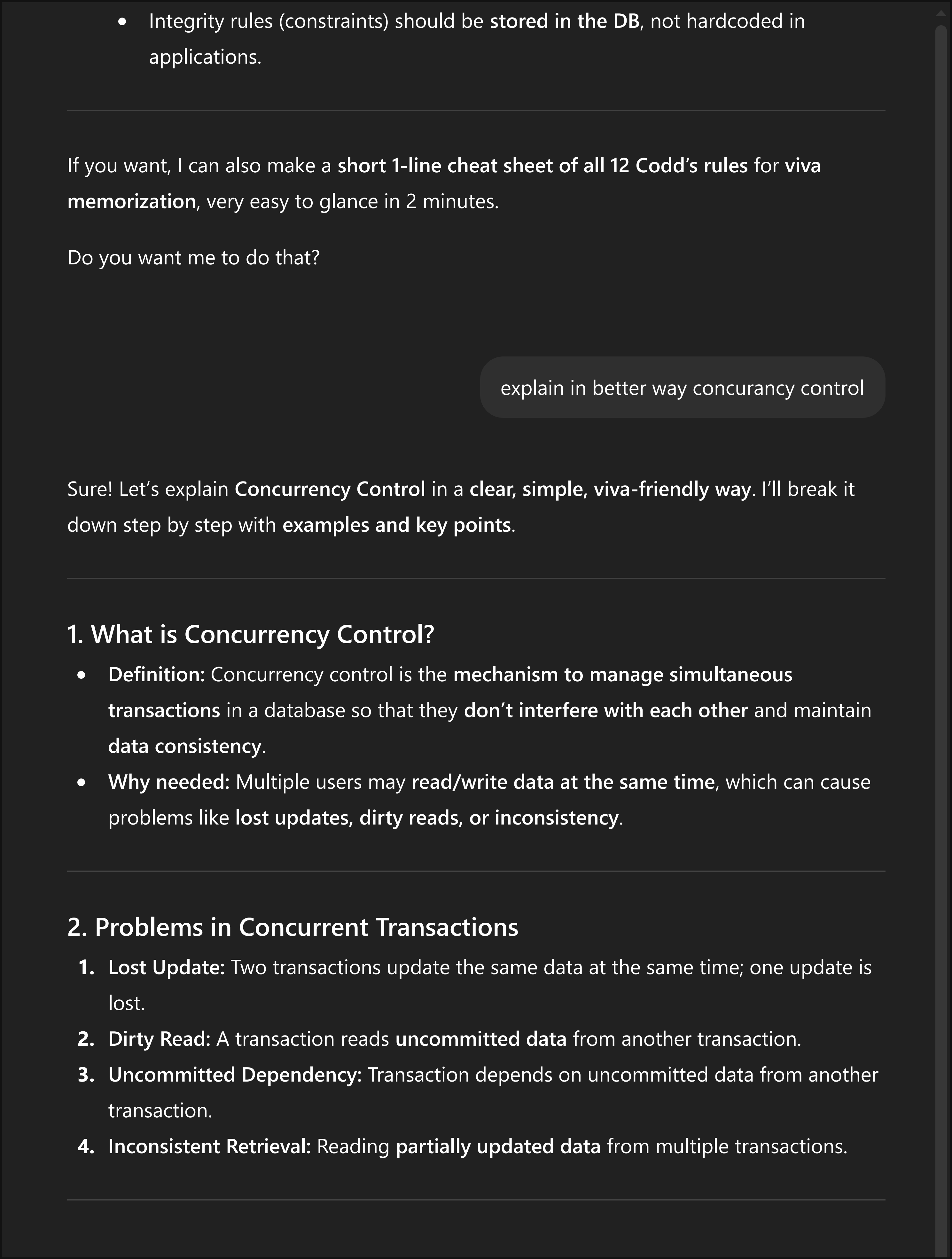
If you want, I can make a **1-page ultra-short revision sheet** combining **nested queries, PL/SQL, cursors, triggers, constraints, normalization, ACID, serialization, and concurrency control** so you can **revise all at once before exam**.

Do you want me to make that?

name some codds rule with one line explain

Sure! Here are some **important Codd's rules** with **1-line explanation each** (viva-friendly):

1. **Rule 1 – Information Rule:**
   - All data is stored in **tables (relations)**, including metadata.
2. **Rule 2 – Guaranteed Access Rule:**
   - Each value is **uniquely accessible** by **table name, column name, and primary key**.
3. **Rule 3 – Systematic Treatment of NULLs:**
   - **NULL represents missing/unknown values**, and is handled consistently.
4. **Rule 4 – Dynamic Online Catalog:**
   - The **database metadata** is stored in tables and can be queried using **SQL**.
5. **Rule 5 – Comprehensive Data Sub-language Rule:**
   - The DB must support at least **one language (like SQL)** for **data definition, manipulation, and control**.
6. **Rule 6 – View Updating Rule:**
   - **Views** (virtual tables) should be **updatable** where possible.
7. **Rule 7 – High-Level Insert, Update, Delete:**
   - DB must allow **set-based operations**, not just row-by-row changes.
8. **Rule 8 – Physical Data Independence:**
   - Changing **storage structures** shouldn't affect applications.
9. **Rule 9 – Logical Data Independence:**
   - Changing **table structure (adding columns)** shouldn't affect existing applications.
10. **Rule 10 – Integrity Independence:**



- Integrity rules (constraints) should be **stored in the DB**, not hardcoded in applications.

If you want, I can also make a **short 1-line cheat sheet of all 12 Codd's rules** for **viva memorization**, very easy to glance in 2 minutes.

Do you want me to do that?

explain in better way concurancy control

Sure! Let's explain **Concurrency Control** in a **clear, simple, viva-friendly way**. I'll break it down step by step with **examples and key points**.

---

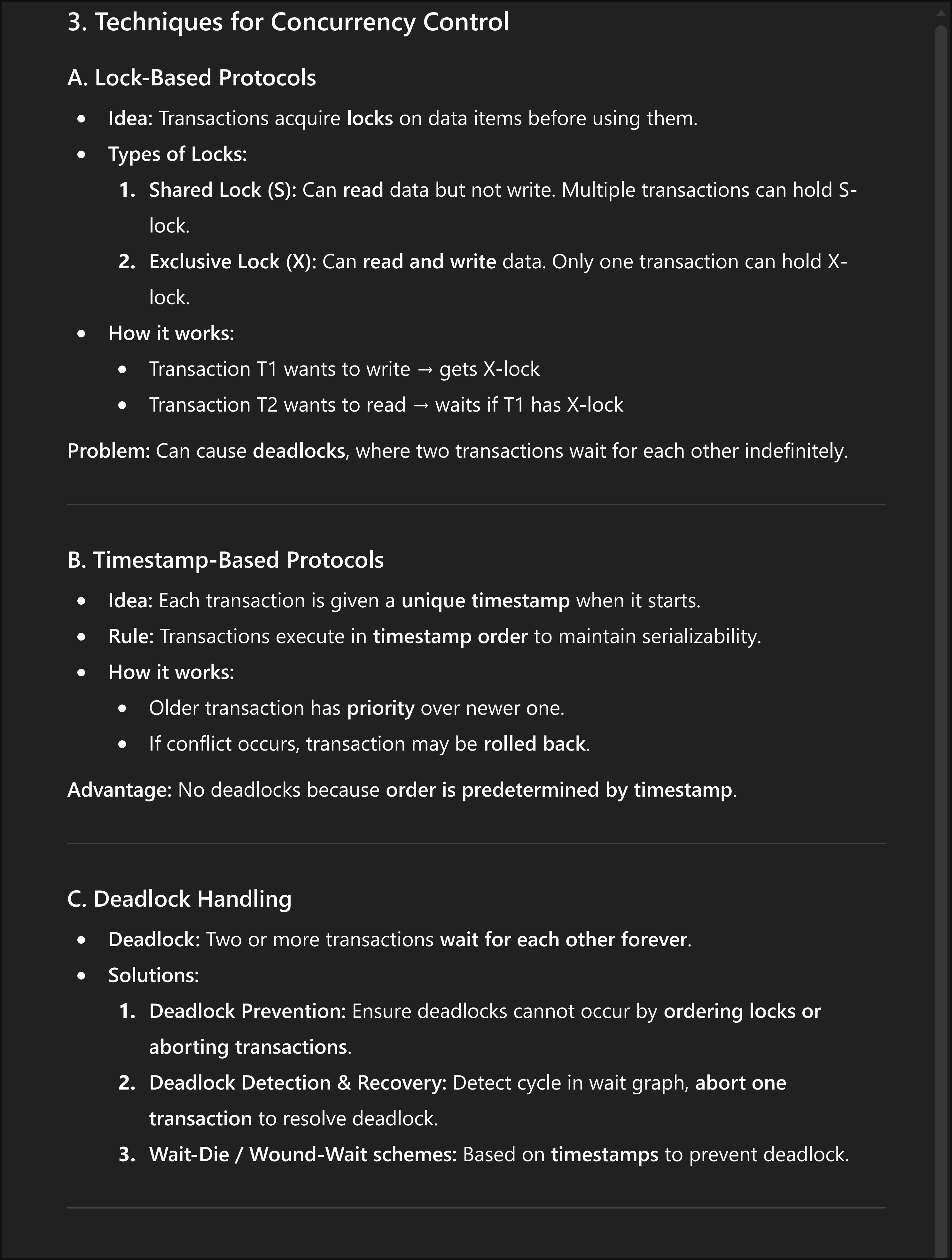
1. What is Concurrency Control?

- **Definition:** Concurrency control is the **mechanism to manage simultaneous transactions** in a database so that they **don't interfere with each other** and maintain **data consistency**.
- **Why needed:** Multiple users may **read/write data at the same time**, which can cause problems like **lost updates, dirty reads, or inconsistency**.

---

2. Problems in Concurrent Transactions

1. **Lost Update:** Two transactions update the same data at the same time; one update is lost.
2. **Dirty Read:** A transaction reads **uncommitted data** from another transaction.
3. **Uncommitted Dependency:** Transaction depends on uncommitted data from another transaction.
4. **Inconsistent Retrieval:** Reading **partially updated data** from multiple transactions.



3. Techniques for Concurrency Control

A. Lock-Based Protocols

- **Idea:** Transactions acquire **locks** on data items before using them.
- **Types of Locks:**
  1. **Shared Lock (S):** Can **read** data but not write. Multiple transactions can hold S-lock.
  2. **Exclusive Lock (X):** Can **read and write** data. Only one transaction can hold X-lock.
- **How it works:**
  - Transaction T1 wants to write → gets X-lock
  - Transaction T2 wants to read → waits if T1 has X-lock

Problem: Can cause **deadlocks**, where two transactions wait for each other indefinitely.

B. Timestamp-Based Protocols

- **Idea:** Each transaction is given a **unique timestamp** when it starts.
- **Rule:** Transactions execute in **timestamp order** to maintain serializability.
- **How it works:**
  - Older transaction has **priority** over newer one.
  - If conflict occurs, transaction may be **rolled back**.

Advantage: No deadlocks because **order is predetermined by timestamp**.

C. Deadlock Handling

- **Deadlock:** Two or more transactions **wait for each other forever**.
- **Solutions:**
  1. **Deadlock Prevention:** Ensure deadlocks cannot occur by **ordering locks or aborting transactions**.
  2. **Deadlock Detection & Recovery:** Detect cycle in wait graph, **abort one transaction** to resolve deadlock.
  3. **Wait-Die / Wound-Wait schemes:** Based on **timestamps** to prevent deadlock.

---

4. Key Points for Viva

- Concurrency control ensures **correct execution of simultaneous transactions**.
- **Lock-based:** Uses **shared/exclusive locks**, may cause deadlocks.
- **Timestamp-based:** Uses **timestamps**, avoids deadlocks.
- **Deadlock handling:** Detect and resolve conflicts to keep DB consistent.
- Main goal: **Maintain ACID properties**, especially **Isolation and Consistency**.

Simple Example:

- Two transactions:
  - T1: Update balance of A by 100
  - T2: Read balance of A
- Without concurrency control: T2 may read old or partially updated value → **inconsistency**.
- With lock-based protocol: T2 waits until T1 commits → **correct result**.

If you want, I can make a **1-page super-short diagram + notes for concurrency control** with **lock-based, timestamp, deadlock handling** for **easy viva recall**.

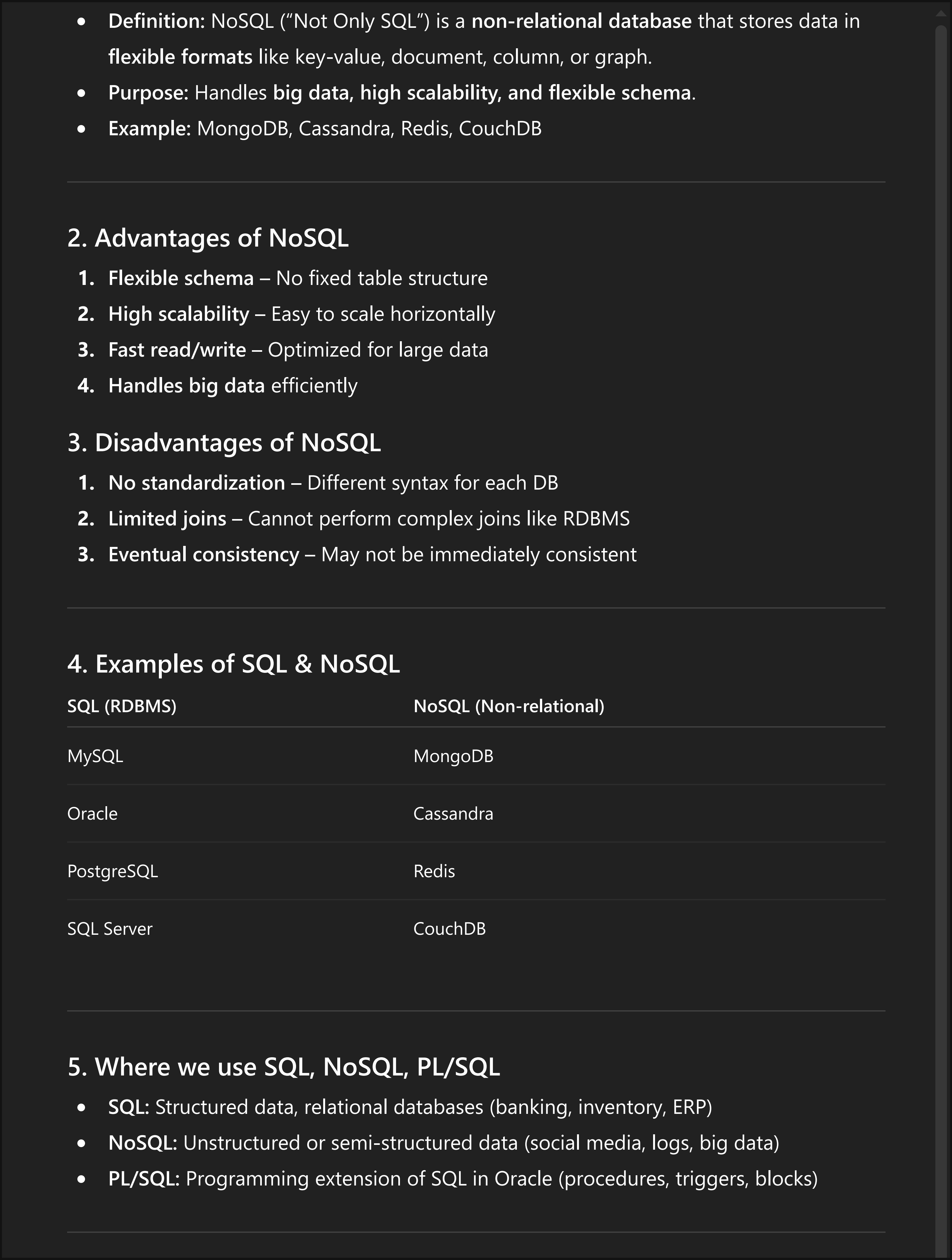
Do you want me to do that?

explain nosql , example , advantage ,distage , name some sql and nosql, where we perform sql, nosql, pl/sql. types of data stucterded ,semi and unstuct. need of nosql, base prperty . differnce bet acid and base. and crud operation of mongodb. what is sharding.

Sure! Let’s explain **NoSQL, SQL, PL/SQL, types of data, BASE vs ACID, MongoDB CRUD, sharding** in a **simple, viva-friendly way**, with **examples and key points**.

---

1. What is NoSQL?



- **Definition:** NoSQL (“Not Only SQL”) is a **non-relational database** that stores data in **flexible formats** like key-value, document, column, or graph.
- **Purpose:** Handles **big data, high scalability, and flexible schema**.
- **Example:** MongoDB, Cassandra, Redis, CouchDB

---

2. Advantages of NoSQL

1. **Flexible schema** – No fixed table structure
2. **High scalability** – Easy to scale horizontally
3. **Fast read/write** – Optimized for large data
4. **Handles big data** efficiently

3. Disadvantages of NoSQL

1. **No standardization** – Different syntax for each DB
2. **Limited joins** – Cannot perform complex joins like RDBMS
3. **Eventual consistency** – May not be immediately consistent

---

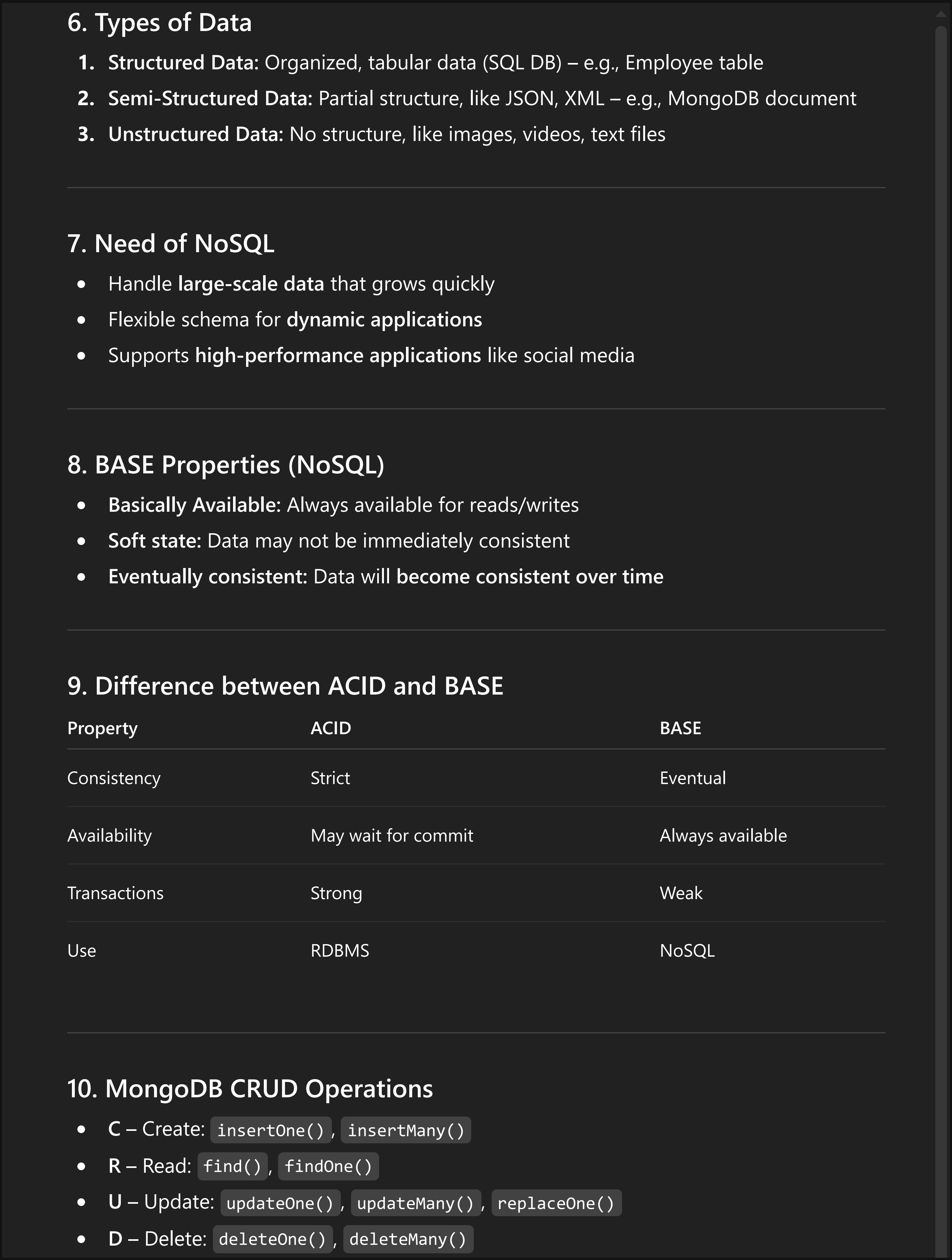
4. Examples of SQL & NoSQL

SQL (RDBMS)	NoSQL (Non-relational)
MySQL	MongoDB
Oracle	Cassandra
PostgreSQL	Redis
SQL Server	CouchDB

---

5. Where we use SQL, NoSQL, PL/SQL

- **SQL:** Structured data, relational databases (banking, inventory, ERP)
- **NoSQL:** Unstructured or semi-structured data (social media, logs, big data)
- **PL/SQL:** Programming extension of SQL in Oracle (procedures, triggers, blocks)



6. Types of Data

1. **Structured Data:** Organized, tabular data (SQL DB) – e.g., Employee table
2. **Semi-Structured Data:** Partial structure, like JSON, XML – e.g., MongoDB document
3. **Unstructured Data:** No structure, like images, videos, text files

---

7. Need of NoSQL

- Handle **large-scale data** that grows quickly
- Flexible schema for **dynamic applications**
- Supports **high-performance applications** like social media

---

8. BASE Properties (NoSQL)

- **Basically Available:** Always available for reads/writes
- **Soft state:** Data may not be immediately consistent
- **Eventually consistent:** Data will **become consistent over time**

---

9. Difference between ACID and BASE

Property	ACID	BASE
Consistency	Strict	Eventual
Availability	May wait for commit	Always available
Transactions	Strong	Weak
Use	RDBMS	NoSQL

---

10. MongoDB CRUD Operations

- **C** – Create: `insertOne()`, `insertMany()`
- **R** – Read: `find()`, `findOne()`
- **U** – Update: `updateOne()`, `updateMany()`, `replaceOne()`
- **D** – Delete: `deleteOne()`, `deleteMany()`

Example:

```javascript
db.students.insertOne({name:"Ram", age:20}); // Create
db.students.find({age:20});                  // Read
db.students.updateOne({name:"Ram"}, {$set:{age:21}}); // Update
db.students.deleteOne({name:"Ram"});         // Delete
```

---

11. What is Sharding?

- **Definition:** Sharding is **horizontal partitioning** of a database to **distribute data across multiple servers**.
- **Purpose:** Improves **performance and scalability** in NoSQL.
- **Example:** MongoDB splits large collections into shards across servers.

Key Points for Viva

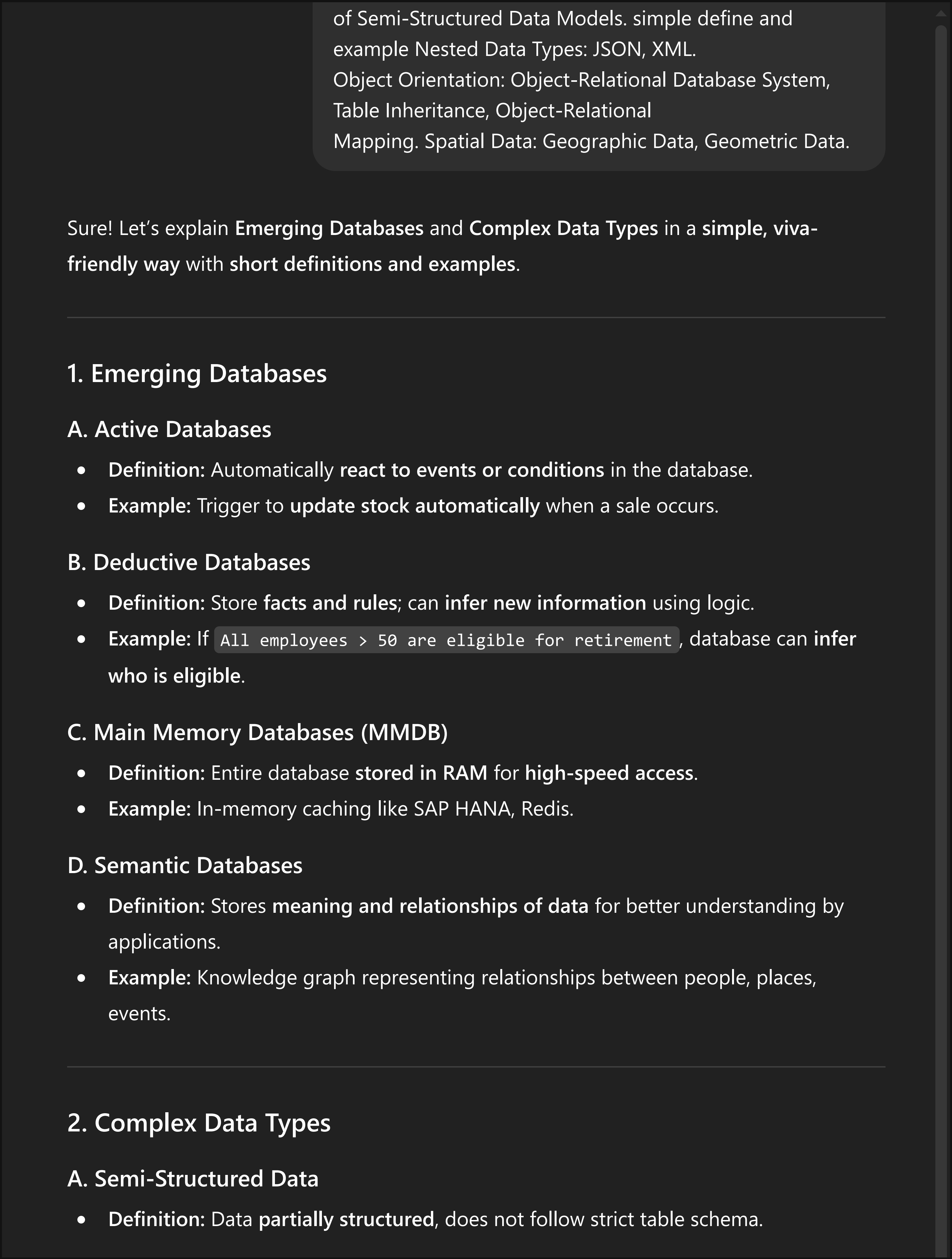
- SQL → structured, relational, ACID, tables
- NoSQL → unstructured/semi, flexible, BASE, documents/key-value
- PL/SQL → procedural SQL (Oracle), used for **blocks, procedures, triggers**
- MongoDB → document DB, supports CRUD operations and sharding for big data

---

If you want, I can make a **1-page MongoDB + NoSQL + SQL + PL/SQL quick revision sheet** for **all these points with examples** so you can memorize in **5–10 minutes before viva**.

Do you want me to do that?

explain Emerging Databases: what are Active and Deductive Databases, Main Memory Databases, Semantic Databases. what are Complex Data Types:
define withexample simple Semi-Structured Data, Features



of Semi-Structured Data Models. simple define and example Nested Data Types: JSON, XML.
Object Orientation: Object-Relational Database System, Table Inheritance, Object-Relational
Mapping. Spatial Data: Geographic Data, Geometric Data.

Sure! Let's explain **Emerging Databases** and **Complex Data Types** in a **simple, viva-friendly way** with **short definitions and examples**.

---

1. Emerging Databases

A. Active Databases

- **Definition:** Automatically **react to events or conditions** in the database.
- **Example:** Trigger to **update stock automatically** when a sale occurs.

B. Deductive Databases

- **Definition:** Store **facts and rules**; can **infer new information** using logic.
- **Example:** If `All employees > 50 are eligible for retirement`, database can **infer who is eligible**.

C. Main Memory Databases (MMDB)

- **Definition:** Entire database **stored in RAM** for **high-speed access**.
- **Example:** In-memory caching like SAP HANA, Redis.

D. Semantic Databases

- **Definition:** Stores **meaning and relationships of data** for better understanding by applications.
- **Example:** Knowledge graph representing relationships between people, places, events.

---

2. Complex Data Types

A. Semi-Structured Data

- **Definition:** Data **partially structured**, does not follow strict table schema.

- **Example:** JSON, XML documents.
- **Features:**
  1. Flexible schema
  2. Self-describing tags/keys
  3. Hierarchical/nested structure

B. Nested Data Types

- **Definition:** Data types that can **contain other data types or collections**.
- **Examples:**
  - **JSON:**

    ```json
    { "name": "Ram", "courses": ["Math", "Physics"] }
    ```

  - **XML:**

    ```xml
    <student>
      <name>Ram</name>
      <age>20</age>
    </student>
    ```