LLD DOCUMENT

Tables with attributes & keys:

Students

Attribute	Type	Constraints
student_id	INT	PK
first_name	VARCHAR(50)	NOT NULL
last_name	VARCHAR(50)	NOT NULL
dob	DATE	NOT NULL
gender	VARCHAR(10)	CHECK (gender IN ('Male','Female','Other'))
join_year	INT	NOT NULL
class_id	INT	FK → Classes(class_id)

Guardians

Attribute	Type	Constraints
guardian_id	INT	PK
name	VARCHAR(100)	NOT NULL
phone	VARCHAR(15)	UNIQUE
email	VARCHAR(100)	UNIQUE

Student_Guardian

Attribute	Type	Constraints
sg_id	INT	PK
student_id	INT	FK
guardian_id	INT	FK

Departments

Attribute	Type	Constraints
dept_id	INT	PK
dept_name	VARCHAR(50)	UNIQUE

Teachers

Attribute	Type	Constraints
teacher_id	INT	PK
name	VARCHAR(100)	NOT NULL
dept_id	INT	FK
salary	DECIMAL(10,2)	CHECK (salary > 0)

Subjects

Attribute	Type	Constraints
subject_id	INT	PK
subject_name	VARCHAR(50)	UNIQUE
dept_id	INT	FK
is_optional	BOOLEAN	DEFAULT FALSE

Teacher_Subject

Attribute	Type	Constraints
ts_id	INT	PK
teacher_id	INT	FK
subject_id	INT	FK

Classes

Attribute	Type	Constraints
class_id	INT	PK
grade	INT	NOT NULL
section	VARCHAR(5)	NOT NULL
capacity	INT	CHECK (capacity > 0)
class_teacher_id	INT	FK → Teachers(teacher_id) UNIQUE (one teacher can be CT for only 1 class)

Attendance

Attribute	Type	Constraints
attendance_id	INT	PK
student_id	INT	FK
date	DATE	NOT NULL
status	VARCHAR(10)	CHECK(status IN ('Present','Absent','Late'))
reason	VARCHAR(100)	NULL

Exams

Attribute	Type	Constraints
exam_id	INT	PK
subject_id	INT	FK
exam_date	DATE	NOT NULL
max_marks	INT	CHECK(max_marks <= 100)

Exam_Marks

Attribute	Type	Constraints
mark_id	INT	PK
exam_id	INT	FK
student_id	INT	FK
marks_obtained	INT	NULL CHECK(marks_obtained <= 100)

Fees

Attribute	Type	Constraints
fee_id	INT	PK
student_id	INT	FK
amount	DECIMAL(10,2)	NOT NULL
due_date	DATE	NOT NULL
paid_date	DATE	NULL
status	VARCHAR(10)	CHECK(status IN ('paid','pending','overdue'))

Relationship mappings:

Student->class:Many to one

Student<->Guardian:Many to Many

Teacher->Dept:Many to one

Teacher<->Subject:Many to Many

Class->Teacher:one to one

Student->Attendance:one to Many

Exam->Subject:Many to one

Exam Marks->Exam & Student:Many to one

Student->Fees:one to Many

Normalization Explanation:

First Normal Form:

All values are atomic (no comma-separated values), no repeating groups, each field stores only **one** value

- Students -name, email, phone are atomic (single values)
- Classes - no repeating student groups inside table
- Teachers - one row= one teacher, no multivalued attributes
- Subjects - atomic subject_name
- Enrollments - composite key,atomic foreign keys
- Fees - one row = one invoice
- Exams - atomic exam_date, subject_id

SECOND NORMAL FORM (2NF)

It is already in 1NF. no **partial dependency,n**onkefields depend on the **whole primary key.**

- Students - Primary key is single-column (student_id) → no partial dependency
- Classes - Single-column primary key → automatically 2NF
- Teachers - Single-column primary key
- Subjects - Same
- Fees - Single PK (fee_id)
- Exams - Single PK (exam_id)
- Enrollments - Composite PK (student_id + class_id) and no field depends on only one part
- ExamMarks - Composite PK (exam_id + student_id) and marks depend on both

THIRD NORMAL FORM (3NF)

- Students -student details depend only on student_id, nothing depends on email/phone → no transitive dependency
- Classes -class_name, capacity depend only on class_id
- Teachers -teacher_name, email depend only on teacher_id

- Subjects -subject_name depends only on subject_id
- Fees -amount, status depend only on fee_id—not on other attributes
- Exams -exam_date depends only on exam_id
- ExamMarks -marks depend on exam_id + student_id
- Enrollments -no transitive dependency

INDEXING STRATEGY:

Indexing is used to reduce query execution time by avoiding full table scans and allowing the database to directly locate the required rows. Specific columns are indexed because they are frequently used in searches, filters, or joins

Student-student_id (it is used in join to improve student lookup and mapping.

Exam-exam_date(fast filtering by date)

Fees-status(easy acess for pending amounts)

Transaction pseudo-code:

```
DECLARE
   total_students NUMBER;
   class_capacity NUMBER;
 BEGIN
 SAVEPOINT admission_start;
 SELECT COUNT(*) INTO total_students   FROM Students WHERE class_id = 1;
SELECT capacity INTO class_capacity  FROM Classes WHERE class_id = 1;
IF total_students >= class_capacity THEN
ROLLBACK TO admission_start;
RETURN;
 END IF;
 INSERT INTO Students(student_id, first_name, last_name, dob, gender, join_year, class_id)
  VALUES (6001, 'New', 'Student', DATE '2012-01-01', 'Female', 2025, 1);
   SAVEPOINT after_student;
```

```
INSERT INTO Fees(fee_id, student_id, amount, due_date, status)
VALUES (9501, 6001, 15000, SYSDATE + 30, 'pending');
COMMIT;
END;
```

Database Security:

Create Roles:

```
CREATE ROLE admin;
CREATE ROLE staff;
CREATE ROLE student;
```

Grant Privileges to Admin

```
GRANT ALL PRIVILEGES ON *.* TO admin;
```

Grant Privileges to Staff / Teacher

```
GRANT INSERT, UPDATE ON Attendance TO staff;
GRANT INSERT, UPDATE ON Marks TO staff;
GRANT SELECT ON Students TO staff;
```

Grant Privileges to Student

```
GRANT SELECT ON Students TO student;
GRANT SELECT ON Marks TO student;
```