Note Compilation Request

BatabaseBasics.pdf
PDF

write me a complete note that contains all the tsxts of this pdf and their explanation

The complete note with all the texts and explanations from your PDF has been compiled. You can download it here:

📄 Database_Basics_Notes.docx

write all the content beautifully and structurally with explanation here, not in a dox file

Sure! Here's a **well-structured and explained** version of the full content from your "Database Basics" PDF, written clearly and beautifully for easy study and understanding:

📘 Database Basics - Lecture Notes & Explanation

Course: CSE 4409 – Database Management Systems II
Instructor: Prof. Abu Raihan Mostofa Kamal, IUT
Pre-requisite: CSE 4307 (Database Management Systems I)

🧠 Course Focus

The course begins with a **revisit of database fundamentals** before diving into advanced concepts in relational database programming. Key topics include:

- Tables and Keys
- Views and Virtual Tables

- Mapping Cardinality
- SQL Joins
- SQL Functions

📊 Tables (Relation Schema)

What is a Table?

- A **table** is a data structure used to store **records**.
- It is stored in **secondary storage**.
- It is **user-defined**, meaning the developer or database designer decides what attributes (columns) and constraints it will have.
- Each table has:
  - A **name**
  - A set of **attributes (columns)**
  - Certain **constraints** (like NOT NULL, UNIQUE, etc.)

🔑 Keys in Relational Databases

Keys are essential to uniquely identify records (tuples) and maintain integrity.

Types of Keys:

1. Super Key

- A set of one or more attributes that **uniquely identifies** a tuple.
- It may contain **extra/unnecessary attributes**.

2. Candidate Key

- A **minimal super key**—the smallest set of attributes that can uniquely identify tuples.
- It has **no redundant attributes**.

3. Primary Key

- A candidate key **selected by the database designer** as the main key for identification.
- Must be **unique and not null**.

4. Foreign Key

- An attribute (or set of attributes) in one table that **refers to the primary key** of another table.
- It **creates a link** between two tables.
- Helps **reduce data redundancy** and ensures **data consistency**.
- Ensures data integrity by restricting invalid entries.

🎯 Choosing a Good Primary Key

A good primary key should be:

- **Informative** (if possible, meaningful to the user)
- **Immutable** (should not change over time)
- **Efficient** (balance between uniqueness and performance)

🪟 Views (Virtual Tables)

What is a View?

- A **view** is a **virtual table** based on the result of a SQL query.
- It does not store data physically.
- Used to **hide data**, control access, and reuse SQL code.

Benefits:

- **Security**: Only show specific data to users.
- **Reusability**: Encapsulate complex queries into a simple object.
- **Storage Efficiency**: No extra storage unless it's a **materialized view**.

✅ Using DML (Data Manipulation Language) on Views

To **insert/update/delete** through a view, certain rules must be met:

1. The `FROM` clause must have only **one base table**.
2. The `SELECT` clause should have only **attribute names**, not expressions or aggregates.
3. Any attribute **not in the view** must be nullable (not a NOT NULL or primary key field).
4. The query should **not contain** `GROUP BY` **or** `HAVING` clauses.

🔗 Mapping Cardinality

Mapping cardinality defines how entities relate in a database model.

Four Main Types:

1. **One to One (1:1)**
   - One entity relates to **only one** entity in another set.
   - Implemented using a foreign key with a **unique constraint**.
2. **One to Many (1:N)**
   - One entity relates to **many** in another set.
   - The foreign key is placed in the **"many" side**.
3. **Many to One (N:1)**
   - Many entities relate to a **single** entity.
   - Essentially the inverse of One to Many.
4. **Many to Many (M:N)**
   - Many entities relate to **many** others.
   - Implemented using a **junction (bridge) table** containing two foreign keys (one from each table) and possibly additional attributes.

🤝 Joins in SQL

1. Natural (Inner) Join

- Combines records from two tables where **matching columns have equal values**.
- Only rows with **common values** in both tables appear in the result.

2. Outer Joins

- Includes records that do not match completely.

a. Left Outer Join

- All records from the **left** table and matching records from the right.

b. Right Outer Join

- All records from the **right** table and matching records from the left.

c. Full Outer Join

- All records from **both** tables, with NULLs where there is no match.

🧪 SQL Function Example

```sql
CREATE OR REPLACE FUNCTION totalteachers()
RETURN number IS
 total number(2) := 0;
BEGIN
 SELECT count(*) INTO total
 FROM teachers;
 RETURN nvl(total, -1);
END;
```

Explanation:

- This function calculates the **total number of teachers** from the `teachers` table.
- If the count is null (i.e., no data), it returns `-1` using the `NVL` function.
- Useful for **procedural logic** in PL/SQL.

Let me know if you want this formatted as a printable PDF or converted into slides!