07 Basic Concepts of DBMS

- **Database Schema**
- **Tables, Rows, and Columns**
- **DBMS Keys**
- **Normalization (1NF, 2NF, 3NF, BCNF)**
- **Indexes**

01 Database Schema

- A schema is the structure of a database, defining how data is organized into tables
- It includes the tables, fields, data types, and relationships between tables
- The schema acts as a blueprint for the database, guiding how data is stored, organized, and accessed
- For example, in an e-commerce database, you might have a schema with tables for `Customers`, `Orders`, and `Products`

02 Tables, Rows, and Columns

- **Tables**
  - The fundamental building blocks of a database
  - A table is a collection of related data entries organized in rows and columns
  - **Example:** A `Users` table might have columns like `UserID`, `Username`, `Email`, and `Password`
- **Rows**
  - Also known as records or tuples, rows represent a single entry in a table
  - **Example:** A single row in the `Users` table could represent one user's data, such as `1, "johndoe", "john@example.com", "password123"`
- **Columns**
  - Also known as fields or attributes, columns define the data type and characteristics of the data stored in each row
  - **Example:** The `Email` column in the `Users` table might be defined as a string that stores the user's email addresses

03 DBMS Keys

1. Primary Key

- **Definition:** A primary key is a unique identifier for each record in a table. It ensures that no two rows have the same value in this column or set of columns.
- **Properties:**
  - Uniqueness: Each value in the primary key column(s) must be unique across the table.
  - Not Null: A primary key cannot contain NULL values.
  - A table can have only one primary key.
- **Example:** In a `Students` table, the `StudentID` column could be the primary key, uniquely identifying each student.

2. Foreign Key

- **Definition:** A foreign key is a column or a set of columns in one table that refers to the primary key in another table. It is used to establish a relationship between the two tables.
- **Properties:**
  - The foreign key value must match an existing primary key value in the referenced table, ensuring referential integrity.
  - A table can have multiple foreign keys.
- **Example:** In an `Orders` table, the `CustomerID` column could be a foreign key that references the `CustomerID` primary key in a `Customers` table, linking each order to a customer.

3. Candidate Key

- **Definition:** A candidate key is a column, or a set of columns, that can uniquely identify any record in a table. Each table can have multiple candidate keys.
- **Properties:**
  - Uniqueness: Each candidate key must have unique values across the table.
  - Not Null: A candidate key cannot contain NULL values.
  - One of the candidate keys is selected as the primary key, while others can be considered as alternate keys.
- **Example:** In a `Students` table, both `StudentID` and `Email` could be candidate keys because they can uniquely identify each student.

4. Super Key

- **Definition:** A super key is any combination of columns that uniquely identifies a row in a table. It includes candidate keys as well as any superset of candidate keys.
- **Properties:**

        - Uniqueness: A super key can uniquely identify each row in a table.
        - Can be composed of one or more columns.
        - Super keys include all candidate keys, and any candidate key is a minimal super key.
- **Example:** In a `Students` table, both `StudentID` and the combination of `StudentID` + `Email` can be super keys.

5. Alternate Key

- **Definition:** An alternate key is any candidate key that is not selected as the primary key. It is an alternative option for unique identification of records in the table.
- **Properties:**
    - Like candidate keys, alternate keys must be unique and not null.
- **Example:** If `StudentID` is chosen as the primary key in the `Students` table, `Email` would be an alternate key.

6. Composite Key

- **Definition:** A composite key is a primary key that consists of two or more columns that together uniquely identify a row in a table.
- **Properties:**
    - The combination of the columns in the composite key must be unique.
    - None of the individual columns can uniquely identify the row by themselves.
- **Example:** In a `CourseRegistrations` table, a combination of `StudentID` and `CourseID` could be used as a composite key to uniquely identify each registration.

7. Unique Key

- **Definition:** A unique key is a constraint that ensures all values in a column or a set of columns are unique across the table, similar to a primary key but can allow one NULL value.
- **Properties:**
    - Uniqueness: Ensures all values are unique.
    - Allows NULL values, but only one NULL per column.
    - A table can have multiple unique keys.
- **Example:** In a `Users` table, the `Username` column might be set as a unique key to prevent duplicate usernames.

8. Surrogate Key

- **Definition:** A surrogate key is an artificial key, usually an auto-incremented number, that is added to a table to act as a primary key.
- **Properties:**
  - Surrogate keys have no business meaning and are typically generated by the database system.
  - Often used when no natural primary key exists or when natural keys are too complex.
- **Example:** In a `Books` table, an auto-incremented `BookID` might be used as a surrogate key, even if each book has a unique ISBN.

9. Natural Key

- **Definition:** A natural key is a key that has a logical relationship to the data it represents, often derived from the data itself.
- **Properties:**
  - Has business meaning and is not artificially created.
  - Could be a candidate key or even a primary key if it uniquely identifies records.
- **Example:** In a `Employees` table, an `EmployeeID` derived from an employee's badge number might be a natural key

04 Normalization (1NF, 2NF, 3NF, BCNF)

- **Normalization**
  - The process of organizing data in a database to reduce redundancy and improve data integrity
- **1NF (First Normal Form)**
  - Ensures that each column contains atomic values (indivisible) and that each column in a table contains unique data
  - **Example:** A table with repeated columns for phone numbers would not be in 1NF; instead, each phone number should be in a separate row
- **2NF (Second Normal Form)**
  - Achieved when a table is in 1NF, and all non-primary key attributes are fully functionally dependent on the primary key
  - **Example:** In a table with a composite primary key, all non-key columns should depend on the entire key, not just part of it
- **3NF (Third Normal Form)**
  - Achieved when a table is in 2NF, and all the attributes are functionally dependent only on the primary key
  - **Example:** If a table contains a column `City` that depends on `ZipCode`, which in turn depends on the `CustomerID`, moving `City` and `ZipCode` to a new table

linked by `CustomerID` brings the table into 3NF

- **BCNF (Boyce-Codd Normal Form)**
  - A stricter version of 3NF where every determinant (a column or set of columns on which another column is fully dependent) must be a candidate key
  - **Example:** If a table has more than one candidate key, all of them must be a determinant for the table to be in BCNF

1. First Normal Form (1NF)

1NF is the simplest form of normalization and requires that:

- Each table has a primary key
- Each column contains atomic (indivisible) values
- Each column contains only one value per row (no repeating groups or arrays)

Example:

Consider a table storing student information and their enrolled courses:

StudentID	StudentName	Courses
1	John	Math, Science
2	Jane	English, History
3	Bob	Math, English, Art

Problem: The `Courses` column contains multiple values (a list of courses), violating the atomic value rule.

1NF Solution: Split the courses into individual rows:

StudentID	StudentName	Course
1	John	Math
1	John	Science
2	Jane	English
2	Jane	History
3	Bob	Math
3	Bob	English
3	Bob	Art

Now, each column contains atomic values, and the table is in 1NF.

2. Second Normal Form (2NF)

2NF is achieved when:

- The table is in 1NF
- All non-key attributes are fully functionally dependent on the primary key

Example:

Consider a `CourseRegistrations` table in 1NF with a composite primary key:

StudentID	CourseID	StudentName	CourseName	Instructor
1	101	John	Math	Dr. Smith
2	102	Jane	English	Dr. Brown
3	101	Bob	Math	Dr. Smith

Problem: The `StudentName` depends only on `StudentID`, and `CourseName` and `Instructor` depend only on `CourseID`, not on the entire composite primary key (`StudentID` + `CourseID`).

2NF Solution: Split the table into two tables:

Students Table:

StudentID	StudentName
1	John
2	Jane
3	Bob

Courses Table:

CourseID	CourseName	Instructor
101	Math	Dr. Smith
102	English	Dr. Brown

CourseRegistrations Table:

StudentID	CourseID
1	101
2	102
3	101

Now, the `CourseRegistrations` table is in 2NF because all non-key attributes are fully dependent on the entire primary key.

3. Third Normal Form (3NF)

3NF is achieved when:

- The table is in 2NF
- There are no transitive dependencies (i.e., non-key attributes do not depend on other non-key attributes)

Example:

Consider a `Students` table in 2NF:

StudentID	StudentName	AdvisorID	AdvisorName
1	John	201	Dr. Allen
2	Jane	202	Dr. Baker
3	Bob	201	Dr. Allen

Problem: `AdvisorName` depends on `AdvisorID`, which is not a primary key but is a non-key attribute. This creates a transitive dependency.

3NF Solution: Split the table into two tables:

Students Table:

StudentID	StudentName	AdvisorID
1	John	201
2	Jane	202
3	Bob	201

Advisors Table:

AdvisorID	AdvisorName
201	Dr. Allen
202	Dr. Baker

Now, the `Students` table is in 3NF because there are no transitive dependencies.

4. Boyce-Codd Normal Form (BCNF)

BCNF is a stricter version of 3NF. A table is in BCNF if:

- It is in 3NF
- For every functional dependency (A → B), A should be a super key

Example:

Consider a `ClassAssignments` table:

ClassID	TeacherID	TeacherName
1	101	Mr. Smith
2	102	Ms. Johnson
1	103	Ms. Brown

Problem: `ClassID` and `TeacherID` both determine `TeacherName`, but neither `ClassID` nor `TeacherID` is a super key by themselves.

BCNF Solution: Create separate tables:

Classes Table:

ClassID	TeacherID
1	101
2	102
1	103

Teachers Table:

TeacherID	TeacherName
101	Mr. Smith
102	Ms. Johnson
103	Ms. Brown

Now, each determinant is a super key, and the table is in BCNF.

05 Indexes

- Indexes are special database objects that improve the speed of data retrieval operations on a database table
- By creating an index on one or more columns, you can make queries that filter or sort data by those columns much faster
- **Types:**

    - **Primary Index:** Automatically created on the primary key of a table
    - **Secondary Index:** Created on columns other than the primary key to speed up query performance
- **Example:** An index on the `Email` column in a `Users` table would make searching for users by email faster

1. Creating Indexes

Basic Index

- A basic index is created on a single column of a table
- It speeds up queries that filter or sort by that column

Example

- Imagine a `Users` table with a `LastName` column. To speed up searches based on `LastName`, you can create an index:

```
CREATE INDEX idx_lastname
ON Users (LastName);
```

- The `idx_lastname` index is created on the `LastName` column of the `Users` table. Queries that search for users by `LastName` will be faster because the database can use this index to quickly locate the rows

Composite Index

- A composite index is created on multiple columns
- It improves performance on queries that filter or sort based on the combination of these columns

Example

- Suppose you frequently run queries that filter by both `LastName` and `FirstName`. You can create a composite index on both columns:

```
CREATE INDEX idx_name
ON Users (LastName, FirstName);
```

- The `idx_name` index is created on both `LastName` and `FirstName` columns. This index will be useful for queries that involve both columns, such as searching for users by both their last and first names

2. Using Indexes in Queries

- Indexes are automatically used by the DBMS when they are appropriate
- You don’t need to specify that you want to use an index; the DBMS handles this

Example

- Consider the `Users` table and the following query:

```
SELECT *
FROM Users
WHERE LastName = 'Smith';
```

- If an index on `LastName` exists, the DBMS will use it to quickly find all users with the last name 'Smith' rather than scanning the entire table

3. Unique Index

- A unique index ensures that the values in the indexed column(s) are unique across the table
- This is often used to enforce uniqueness on a column that is meant to have unique values, such as email addresses or usernames

Example

- To ensure that each email address in the `Users` table is unique, you can create a unique index:

```
CREATE UNIQUE INDEX idx_email
ON Users (Email);
```

- The `idx_email` index ensures that all values in the `Email` column are unique. If you try to insert a duplicate email address, the DBMS will reject it

4. Full-Text Index

- A full-text index is used for searching text within a column
- It is particularly useful for searching large amounts of text data

Example

- If the `Articles` table has a `Content` column with large text fields, you can create a full-text index to enable efficient searches:

```
CREATE FULLTEXT INDEX idx_content
ON Articles (Content);
```

- **Explanation:** The `idx_content` full-text index allows you to perform full-text searches within the `Content` column, making it faster to search for keywords or phrases

5. Dropping Indexes

- Indexes can be dropped if they are no longer needed, which can help improve performance for data modification operations

Example

- To remove the `idx_lastname` index from the `Users` table:

```
DROP INDEX idx_lastname
ON Users;
```

- The `idx_lastname` index is removed from the `Users` table, which might improve performance for inserts, updates, or deletes but may slow down queries that used this index.

6. Index Maintenance

- Indexes need to be maintained to ensure optimal performance
- This involves periodically rebuilding or reorganizing indexes to optimize their efficiency, especially after significant changes to the data

Example

- To rebuild an index in MySQL, you can use:

```
ALTER TABLE Users
DROP INDEX idx_lastname,
ADD INDEX idx_lastname (LastName);
```

- This command drops the existing `idx_lastname` index and then recreates it. Rebuilding indexes can help improve performance if they become fragmented