Detailed Revision Notes on Database Table Inheritance and Identifiers

Topics Covered

1. **Approaches to Representing Inheritance in Database Tables:**
   - Table per Class
   - Joined Tables
   - Single Table Strategy
   - Mapped Superclass
2. **Identifying Primary Keys:**
   - Integer Ranges
   - UUIDs and their Implications

---

1. Approaches to Representing Inheritance in Database Tables

Different strategies can be employed to represent class inheritance in databases. Each approach has its pros and cons, and the choice largely depends on the requirements such as space efficiency and query speed.

A. Table per Class

- **Concept:** Each class in the inheritance hierarchy gets its own table. This includes creating separate tables for base classes and derived classes.
- **Structure:** For example, four entities (User, Instructor, TA, Mentor) would require four tables, each storing distinct attributes related to its class 【4:4†source】.
- **Pros:**
  - No redundant data storage due to single responsibility of each table .

- Increased table maintenance .

B. Joined Tables

- **Concept:** Information is stored in multiple related tables. Foreign key relationships maintain links between entities.
- **Structure:** A central user table with additional tables for the other entity types maintaining linkage through foreign keys .
- **Pros:**
  - Offers normalized data storage.
- **Cons:**
  - Performance overhead due to the need for joins in queries .

C. Single Table Strategy

- **Concept:** All fields for all classes are merged into a single table, with a discriminator column to identify the class.
- **Structure:** Stores all data in one table with a type column like `UserType` to distinguish between different classes .
- **Pros:**
  - Simplified schema with fewer joins necessary.
- **Cons:**
  - Can result in sparse table with many null values, leading to inefficient space usage .

D. Mapped Superclass

- **Concept:** A base class provides common fields to inherited classes without a corresponding table in the database.
- **Pros:**
  - Useful when the base class is purely for structure and there's no need to query directly on it .
- **Cons:**
  - Limited as it cannot be queried, primarily intended for inheritance in object design .

Selecting a data type for primary keys is crucial due to the range and uniqueness requirements.

Integer as a Primary Key

- **Limitations:**
  - Integer is typically 4 bytes long with a limited range (2^31-1).
  - Not feasible for large-scale unique key requirements such as URLs, IDs for social media posts .

Universally Unique Identifiers (UUIDs)

- **Concept:** A UUID offers a 128-bit unique identifier useful for large datasets where collision is a concern["UUID - Universally Unique Identifier"].
- **Advantages:**
  - Almost nil probability of collisions which enhances data integrity.
  - Suitable for systems needing decentralized key creation.
- **Applications:** Particularly beneficial for distributed systems or systems where node synchronization is complex and costly .

This summary provides an in-depth understanding of the strategies used for representing inheritance in databases and the considerations involved in primary key selection. Careful analysis of requirements can guide the implementation of the most appropriate approach for inheritance and key management in your database architecture.