Lecture Notes for Lecture 2 of CS 5200 (Database Management Systems) for the Summer 1, 2019 session at the Northeastern University Silicon Valley Campus.

Introduction to Database Concepts

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 1 Review

- This lecture introduced database management systems, and the two most widely used types of DBMS that present different models of information storage and organization.
- Relational database management systems (RDBMS) became dominant in the 1980s. They model data as rows and columns in a series of tables.
- Non-relational database management systems (NoSQL) became popular in the 2000s. They model data as hierarchical documents of attribute/value pairs.

Lecture 1 Review

- We learned that DBMSs provide functions that allow management of a database and its data to be classified into four main groups, which are also subsystems in most DBMSs:
  - **Data definition** – Creation, modification and removal of definitions that define the organization of the data.
  - **Update** – Insertion, modification, and deletion of the actual data.
  - **Retrieval** – Providing information in a form directly usable or for further processing by other applications.
  - **Administration** – Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information that has been corrupted by some event such as an unexpected system failure.

Lecture 2 Introduction

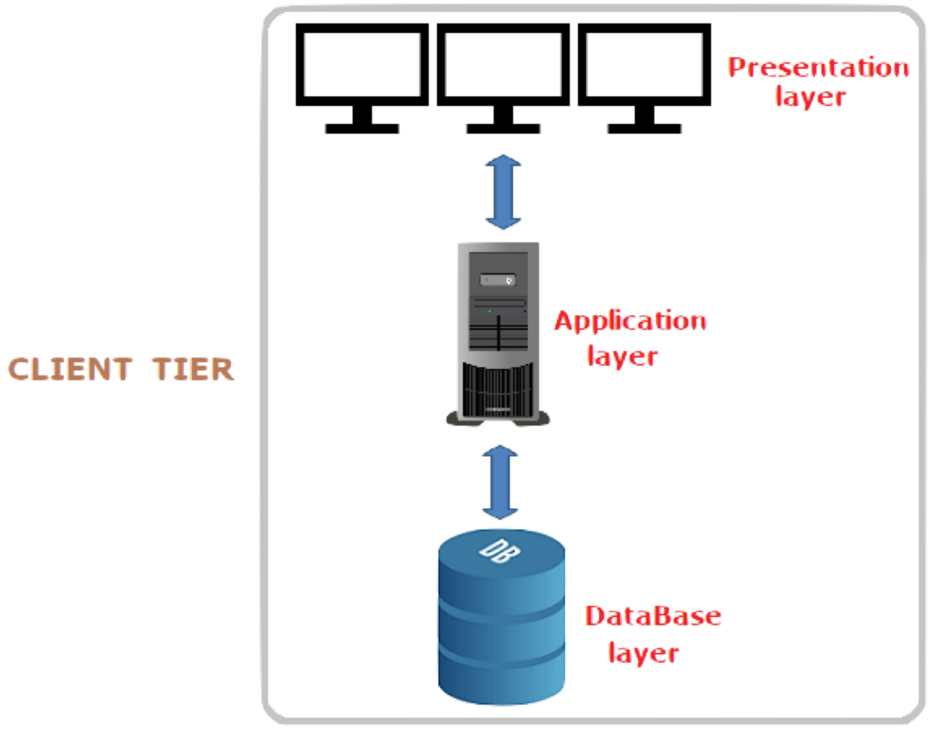
- In this lecture, we will consider some of the basic principals of data management that are foundational to databases in general and to relational databases in particular.
- We will also look at common software architectures that are used to create data-centric applications.
- Finally, we will begin to explore the relational database and the the underlying principals that Edgar Codd developed to characterize databases based on his relational model.

Database Architecture

- The design of a DBMS depends on its architecture. It can be centralized, decentralized or hierarchical.
- The architecture of a DBMS can be seen as either single tier or multi-tier.
- An *n-tier architecture* divides the whole system into related but **n** independent modules that can be independently modified, altered, changed, or replaced.

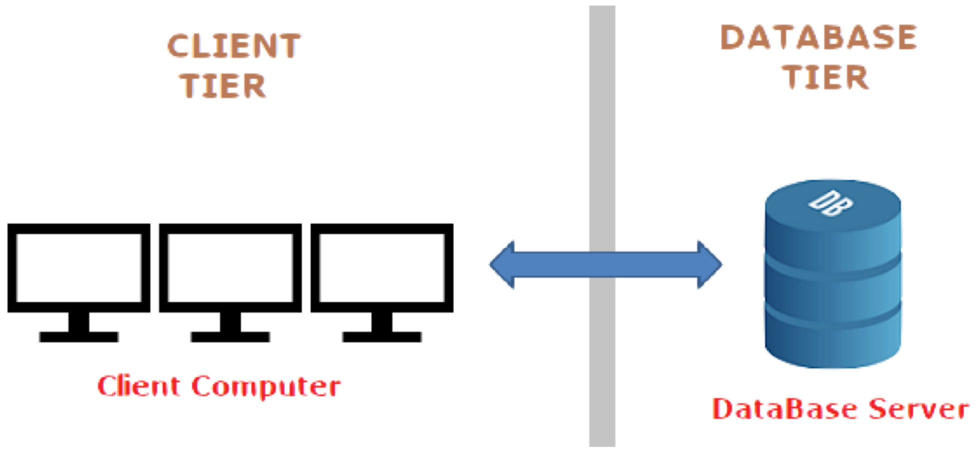
Database Architecture

- In *1-tier* architecture, the DBMS is the only entity where the application interacts directly with the DBMS.
- Any changes are done directly on the DBMS itself.



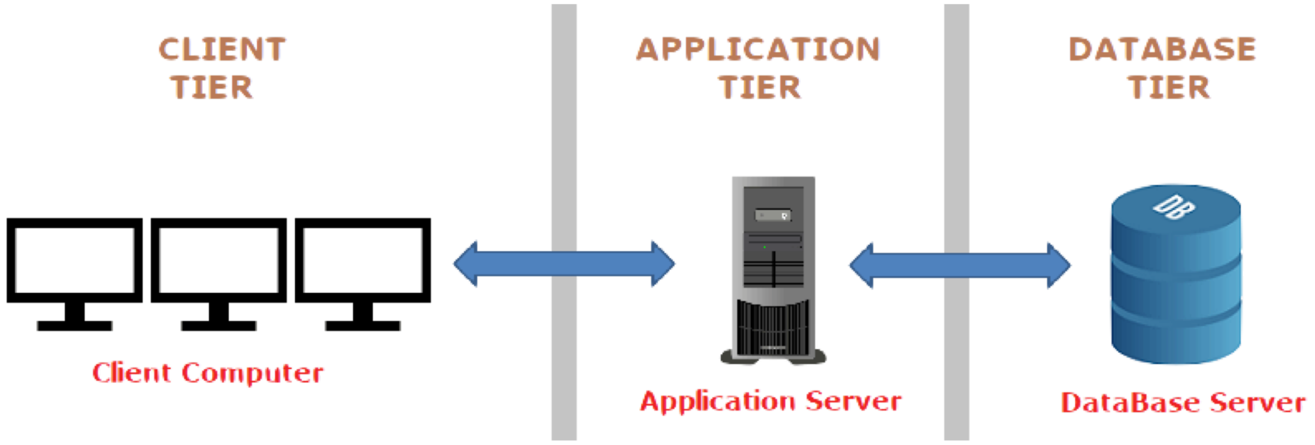
Database Architecture

- If the architecture of DBMS is *2-tier*, then it must have an application through which the DBMS can be accessed.
- Programmers use 2-tier architecture where they access the DBMS by means of an application.
- The application tier is entirely independent of the database in terms of operation, design, and programming.



Database Architecture

- A *3-tier* architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.
- Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.



Database Architecture

- **Database (Data) Tier** – At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

DataBase Server

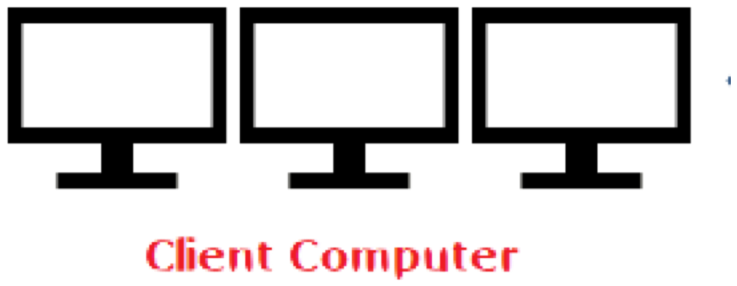
Database Architecture

- **Application (Middle) Tier** – The application server and the programs that access the database reside at this tier.
- The application tier presents an abstracted view of the database to the client tier.
- At the other end, the database tier is not aware of any user beyond the application tier. The application tier sits in the middle and acts as a mediator between the client and the database.

Application Server

Database Architecture

- **Client (Presentation) Tier** – The client or presentation tier interacts with the user and mediates user interactions with the application tier.
- For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application.



Data Models

- *Data models* define how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in a DBMS.
- Data models define how data is connected to each other and how they are processed and stored inside the system.
- The very first data model could be flat data-models, where all the data used are to be kept in the same plane.
- Earlier data models were not so scientific, hence they were prone to introduce lots of duplication and update anomalies.

Data Models

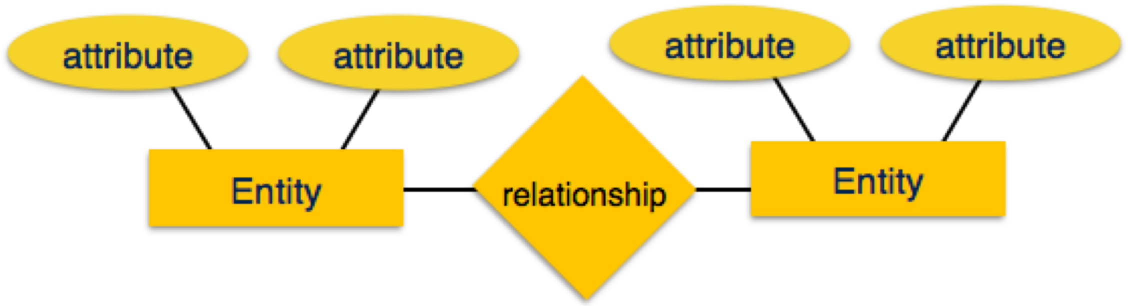
Entity-Relationship Model

- Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them.
- While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.
- ER Model is best used for the conceptual design of a database.

Data Models

Entity-Relationship Model

- ER Model is based on –
  - **Entities** and their *attributes.*
  - **Relationships** among entities.



Data Models

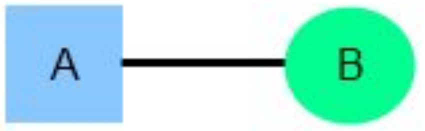
Entity-Relationship Model

- **Entity** − An entity in an ER Model is a real-world entity having properties called **attributes**.
- **Attribute** - Every attribute is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
- **Relationship** − The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways.

Data Models

One-to-One Relationship

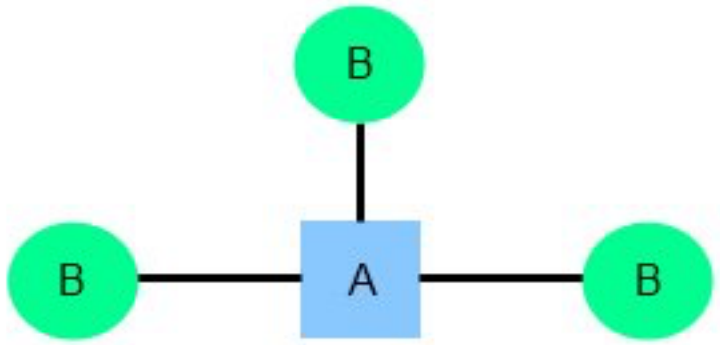
- In a one-to-one relationship, one instance of an entity A is associated with one other instance of another entity B.
- For example, in a database of employees, each employee name (A) is associated with only one social security number (B).



Data Models

One-to-Many Relationship

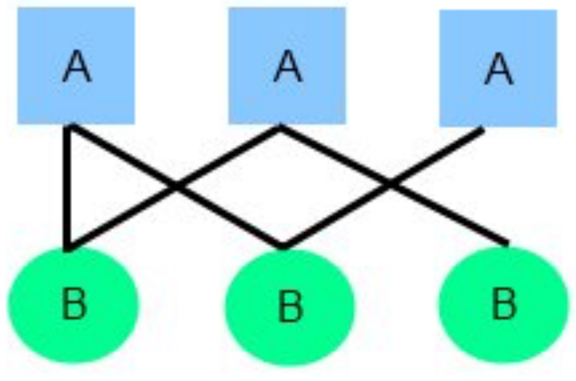
- In a one-to-many relationship, one instance of an entity A is associated with zero, one or many instances of entity B, but entity B is associated with only one instance of entity A.
- For example, if all employees work in one building, the building name (A) is associated with many different employees (B), but those employees all share a singular association with entity A.



Data Models

Many-to-Many Relationship

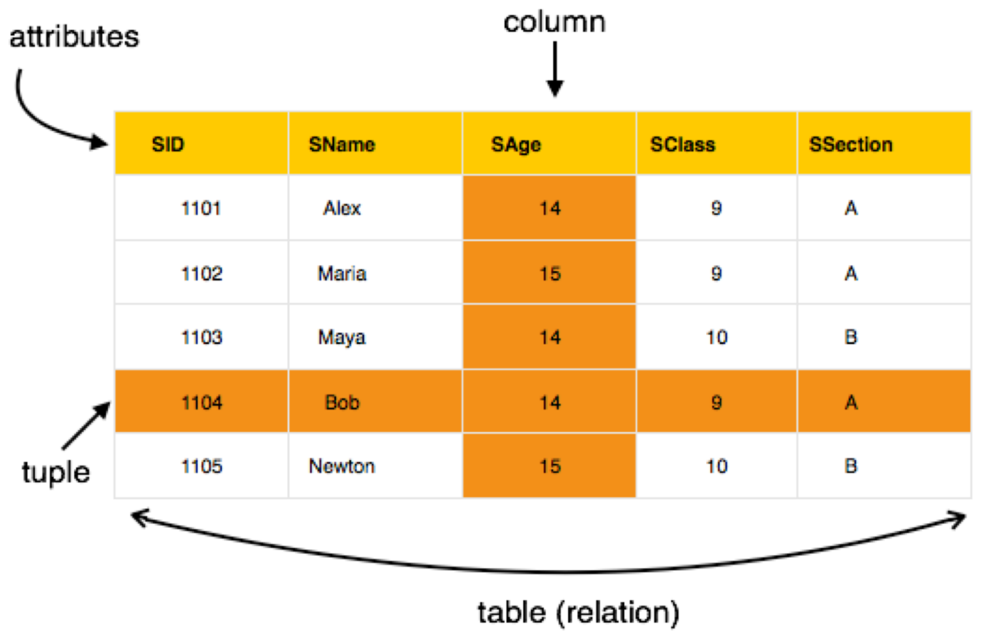
- In a one-to-many relationship, one instance of entity A is associated with one, zero or many instances of entity B, and one instance of entity B is associated with one, zero or many instances of entity A.
- For example, if all of its employees work on multiple projects, an employee (A) is associated with many instances of a project (B), and a project (B) has multiple employees (A).



Data Models

Relational Model

- Relational databases like MySQL, Oracle, and Microsoft SQL use a relational model that is based on first-order predicate logic and defines a table as an *n-ary relation*.



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

Data Models

Relational Model

- The main highlights of the relational model are −
  - Data is stored in tables called **relations**.
  - Each row in a relation contains a unique value.
  - Each column in a relation contains values from a same domain
- With a relational model, all data has an inherent structure that is defined by the schema -- a formal definition of how data inserted into the database will be composed.
- For instance, a given column in a table may be restricted to integers only.
- Entities, attributes, and relations must be mapped to the schema defined by the relational models.

Data Models

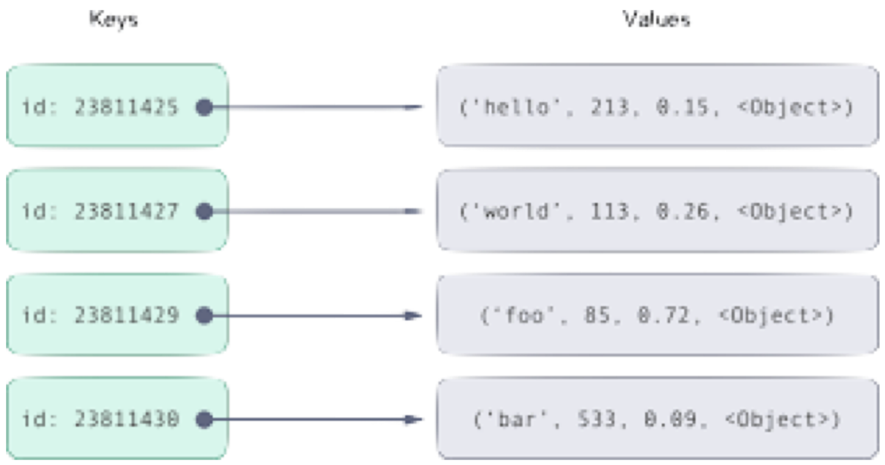
NoSQL Models

- With NoSQL, data can be stored in a schema-less or free-form fashion. Any data can be stored in any record.
- Among the NoSQL databases, there are four common models for storing data, which lead to four common types of NoSQL systems:
    - Document databases
    - Key-value stores
    - Wide column stores
    - Graph databases

Data Models

NoSQL Models

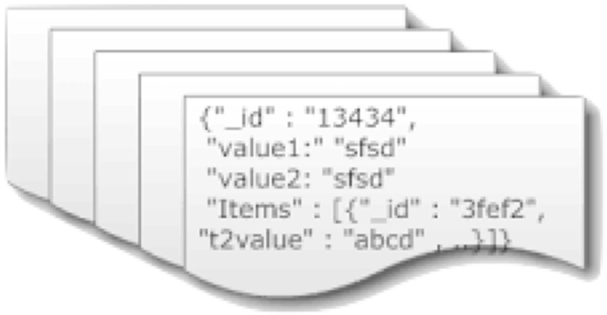
- ***Key-value store*** (e.g. Redis, Riak)
  - Free-form values—from simple integers or strings to complex JSON documents—are accessed in the database by way of keys.


Keys
Values
id: 23811425
('hello', 213, 0.15, <Object>)
id: 23811427
('world', 113, 0.26, <Object>)
id: 23811429
('foo', 85, 0.72, <Object>)
id: 23811430
('bar', 533, 0.09, <Object>)

Data Models

NoSQL Models

- ***Document database*** (e.g. CouchDB, MongoDB)
  - inserted data is stored in the form of free-form JSON structures or “documents.”
  - The data could be anything from integers to strings to freeform text. There is no inherent need to specify what fields, if any, a document will contain.
  - A document database is a specialized key-value store that exposes the internal structure of the document.

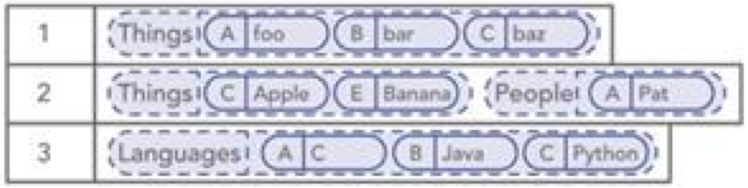


```
{"_id" : "13434",
 "value1:" "sfsd"
 "value2: "sfsd"
 "Items" : [{"_id" : "3fef2",
"t2value" : "abcd" , ...}]}
```

Data Models

NoSQL Models

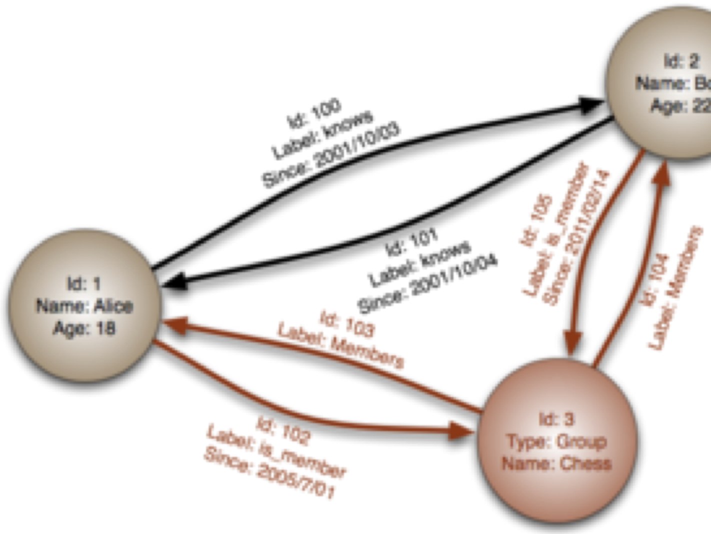
- ***Wide column stores*** (e.g. HBase with Hadoop, Cassandra).
  - Data is stored in columns instead of rows as in a conventional SQL system.
  - Any number of columns (and therefore many different types of data) can be grouped or aggregated as needed for queries or data views.



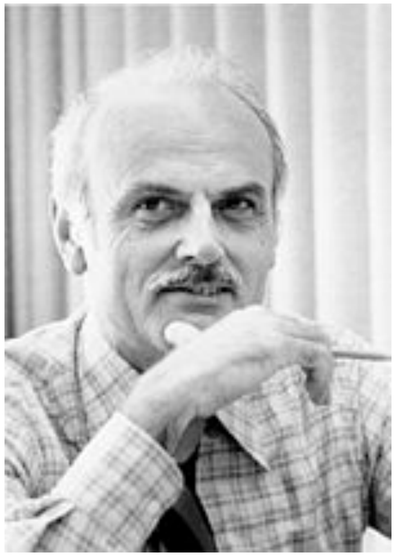
Data Models

NoSQL Models

- ***Graph databases*** (e.g. Neo4j).
  - Data is represented as a graph or network of entities and their relationships, with each node in the graph a free-form chunk of data.



Relational Database



- The term "relational database" was invented by E. F. Codd at IBM in 1970. Codd introduced the term in his research paper "A Relational Model of Data for Large Shared Data Banks". (CACM v. 13, issue. 6, June 1970, pp 377-387)
- In the paper and later papers, he defined what he meant by "relational". One well-known definition of what constitutes a relational database system is composed of Codd's 12 rules.
- Many early implementations of the relational model did not conform to all of Codd's rules, so the term gradually came to describe a broader class of database systems,

Relational Database

- The most common definition of a relational database presents a view of data as a collection of rows and columns, even if it is not based strictly upon relational theory.
- By this definition, RDBMS products typically implement some but not all of Codd's 12 rules.
- A second school of thought argues that if a database does not implement all of Codd's rules (or the current understanding on the relational model, it is not relational.
- This view, shared by many theorists and other strict adherents to Codd's principles, would disqualify most DBMSs as not *truly relational*.

Relational Data Model

Concepts

- **Table**: relational model saves relations among entities as tables with rows as records and columns as attributes.
- **Tuple**: row of table with single record for relationship.
- **Relation instance**: finite set of tuples in database, with no duplicate tuples.
- **Relation schema**: structural description of relation (table) name, attributes, and attributes and their names.
- **Relation key**: attributes uniquely identify tuple in relation.
- **Attribute domain**: pre-defined value scope for attribute.

Codd's Database Rules

Developed by Edgar Codd in the late 1960s, he posited twelve rules databases must obey to be regarded as truly relational.

1. **Information Rule**: everything stored in tables.
2. **Guaranteed Access**: everything accessible through tables
3. **Systematic Treatment of NULL**: consistent interpretation of NULL across system: missing? not known? not applicable?
4. **Active Online Catalog**: complete structure of database accessible using same query language.
5. **Comprehensive Data Sub-Language**: access exclusively through language for definition, manipulation, transactions.
6. **View Updating Rule**: All available views must also be updatable by the systems.

Codd’s Database Rules

7. **High-level Insert, Update, Delete**: operations support operations across rows and tables to yield data records
8. **Physical Data Independence**: data independent of any given application, or physical structure.
9. **Logical Data Independence**: changes to logical data must not affect application using it (most difficult rule to apply)
10. **Integrity Independence**: database, integrity constraints can be modified independent of application
11. **Distribution Independence**: end-user unaware that database is distributed over various locations.
12. **Non-Subversion Rule**: if system has low-level access, cannot subvert system or bypass security or integrity constraints.

Relational Data Model

Relational Integrity Constraints

- Every relation has conditions that must hold to be valid.
- Three main types of integrity constraints
  - Key constraints
  - Domain constraints
  - Referential integrity constraints

Relational Data Model

Key Constraints

- A minimal subset of attributes in a relation can uniquely identify a tuple
- Minimal subset of attributes is known as the *key*.
- If more than one such subset, they are *candidate keys*.
- Key constraints force that:
  - No two tuples have identical values for key attributes
  - Key attribute cannot have NULL value.
- Key constraints also known as *entity constraints*.

Relational Data Model

Domain Constraints

- Attributes of relation limited by real-world scenarios
- Same attributes that were employed in ER model
- Examples:
    - Age cannot be less than 0
    - Telephone number cannot contain digits outside 0-9
    - Identifier must begin with at least one letter and may have any number of additional letters or digits.

Relational Data Model

Referential Integrity Constraint

- Referential integrity constraint states that if relation refers to key attribute of another or same relation, then key elements must exist.
- Works on concept of a foreign key: a key attribute of one relation that can be referred to in another relation.

Next Lecture

- In the next lecture, we will look at the underlying mathematical model on which Edgar Codd based the design his relational database.
- We will learn about the history and motivation of Codd's relational algebra, and then look in more detail at the individual operations.
- Finally, we will use an online tool that can execute relational algebra queries against pre-defined data bases to help prepare us for studying relational database systems.