Touray - DSP567-HW1

Sheikh-Sedat Touray

September 2024

1 Examples of Integrity Constraints

Integrity constraints are rules enforced on databases to ensure data accuracy, consistency, and reliability. Here are some examples of integrity constraints that can apply:

- **a**. **Primary Key Constraint**: Ensures that each record in a table is unique and identifiable. For example, in a table of customers, the "Customer ID" can be a primary key.
- **b**. **Foreign Key Constraint**: Enforces a relationship between two tables by ensuring that the value in one table corresponds to a valid entry in another. For example, in an "Orders" table, the "Customer ID" field should refer to a valid "Customer ID" in the "Customers" table.
- **c**. **Unique Constraint**: Ensures that all values in a column or a set of columns are distinct. For example, an "Email" field in a "Users" table might have a unique constraint to prevent duplicate email addresses.
- **d**. **Not Null Constraint**: Ensures that a column cannot contain NULL values. For example, a "Date of Birth" field in an "Employees" table might have a not null constraint, meaning every employee must have a valid birthdate.
- **e**. **Check Constraint**: Ensures that all values in a column satisfy a specific condition. For example, a "Salary" column in an "Employees" table might have a check constraint to ensure the salary is greater than zero.
- **f**. **Domain Integrity**: Ensures that data values in a column fall within a specific range or set. For instance, a "Gender" field might be limited to the values 'Male', 'Female', or 'Other'.

2 DBMS Architecture for a Web-based system to make airline reservations and sell airline tickets

For designing a web-based system to make airline reservations and sell airline tickets, I would choose a Distributed Database Management System (DDBMS) with a client-server architecture because of the following reasons:

2.0.1 Scalability and Availability:

- **Airline reservations** involve millions of transactions from different locations worldwide. A distributed system can spread the load across multiple servers in different locations, improving scalability and system availability.
- The client-server architecture allows front-end applications (user interfaces) to communicate with multiple backend databases, ensuring real-time processing of reservations and payments. The distributed nature supports high availability, reducing downtime, which is critical for such a high-demand system.

2.0.2 Fault Tolerance and Disaster Recovery:

- A distributed system ensures data replication across multiple sites, so even if one server goes down, another can take over with minimal impact. This redundancy is crucial for the **24/7 uptime** requirements of an airline reservation system.
- In case of regional outages (e.g., due to natural disasters), the system can reroute traffic to other operational nodes, ensuring that customers can still make bookings and the business continues uninterrupted.

2.0.3 Performance and Speed:

- A **distributed architecture** allows data to be stored closer to where the transactions occur (e.g., booking servers in different countries), reducing latency for users. A centralized architecture could result in bottlenecks or delays as all users access the same server.
- Optimized query processing and caching mechanisms can be employed to ensure fast responses to booking queries, which is important for a system where users expect immediate feedback (e.g., ticket availability).

2.0.4 Concurrency Control:

Airline reservation systems must handle **high levels of concurrency**, with multiple users trying to book the same flights simultaneously. Distributed

databases can manage this efficiently using advanced concurrency control algorithms and transactional consistency across multiple nodes.

2.0.5 Data Localization and Compliance:

Airlines operate in various countries, each with specific data regulations (e.g., **GDPR** in the EU). A distributed architecture allows for local data storage to comply with these regulations while maintaining a global system.

2.0.6 Why Other Architectures Might Not Be Ideal:

1. **Centralized DBMS**:
   - A centralized database could lead to **performance bottlenecks**, as all users would need to access a single server. This would reduce the system's ability to scale, especially during peak periods like holidays.
   - In the event of a server failure, the entire system could be down, affecting ticket sales and reservations across the world.
2. **File-based Systems**: File-based systems lack transactional consistency and advanced querying capabilities. Airline reservations require atomicity (all or nothing) to ensure that double bookings or incomplete reservations don't occur, which file systems cannot efficiently manage.
3. **Peer-to-peer Architecture**: Peer-to-peer architectures, although decentralized, are not suitable for systems requiring **high levels of coordination, consistency, and transactional integrity**. Airline reservation systems need central oversight to prevent issues like **overbooking**, which is difficult to manage in a peer-to-peer setup.
4. **Hierarchical or Network DBMS**: These architectures, while useful in specific contexts, are less flexible and harder to scale for web-based applications. They also lack the ease of use and robustness needed to handle the complex relationships and transactions in a reservation system, such as ticket availability, customer details, payments, and more.

3 Why are tuples in a relation are not ordered?

In relational databases, tuples (which represent rows of data) in a relation (table) are not ordered due to the foundational principles of the relational model, which treats a relation as a set. Below are the reasons why:

- **Set Theory Basis**: The relational model is based on **set theory**, where a set is a collection of distinct elements with no inherent order. In a set, the position of an element has no meaning—only the membership matters. Similarly, tuples in a relation are considered elements of a set, where their ordering is not defined by the model.

- **Logical Independence**: The lack of ordering reflects the **logical data independence** of the relational model. This means that users interact with the data based on **what it represents**, not on how it is stored or arranged in memory. Since tuples represent data about entities (e.g., customer information), the order in which they are presented should not affect how queries or operations are performed.
- **Query Optimization**: By not enforcing an order, the database management system (DBMS) has more flexibility in optimizing query execution. The DBMS can store and retrieve data in any physical order that improves performance, such as indexing or partitioning schemes, without impacting the logical structure of the relation.
- **Ordering is Done at Query Time**: If order is needed (for example, when displaying results in a certain sequence, like sorting flights by departure time), it is explicitly specified in the query using `ORDER BY`. This allows for flexible ordering based on different criteria rather than relying on any intrinsic tuple order.
- **Consistency Across Implementations**: Since relations are unordered, the result of a query that does not specify an order (with `ORDER BY`) can be returned in any order. Different DBMS implementations might return tuples in a different sequence based on physical storage optimizations, but this does not affect the correctness of the query result.

4 What is the difference between a key and a superkey?

4.0.1 . Superkey:

- A **superkey** is any set of one or more attributes that can uniquely identify a tuple (row) in a relation. In other words, a superkey ensures that no two tuples in the relation will have the same values for the superkey attributes.
- A superkey can consist of a combination of attributes, even if some attributes are redundant or unnecessary for unique identification.
- For example, in a relation `Employee(EmpID, Name, Department)`, both `EmpID` and `(EmpID, Name)` are superkeys because they can uniquely identify an employee. However, the attribute `Name` might be unnecessary for unique identification since `EmpID` alone is sufficient.

4.0.2 . Key (Candidate Key):

- A **key** (also called a **candidate key**) is a **minimal superkey**. It is the smallest set of attributes that can uniquely identify a tuple in a relation. No attribute in a key can be removed without losing the ability to uniquely identify a tuple.

- A key is a superkey with no unnecessary or redundant attributes.
- In the same example `Employee(EmpID, Name, Department)`, `EmpID` is a key, while `(EmpID, Name)` is not a key because the inclusion of `Name` is redundant for unique identification.

4.0.3 Key Points:

- **All keys are superkeys**, but not all superkeys are keys.
- **Superkeys** may have extra, redundant attributes that are not needed to guarantee uniqueness.
- **Keys** are the minimal form of superkeys and have no unnecessary attributes.

4.0.4 Example:

Consider the relation `Student(StudentID, Name, Email)`:

- **Superkeys**:
  - `{StudentID}`
  - `{StudentID, Name}`
  - `{StudentID, Email}`
  - `{StudentID, Name, Email}`
- **Keys** (Minimal Superkeys): `{StudentID}`

5 Under what circumstances lead to occurrence of NULL values in a relation?

5.0.1 Missing Data:

- Data might not be available at the time of data entry. For example, in a customer database, if a customer has not yet provided their phone number, the "Phone Number" field could contain a NULL value.
- Example: `Customer(Name, Email, Phone)` where `Phone` might be NULL if not provided.

5.0.2 Unknown Data:

- The exact value for an attribute is unknown, even though it may exist. For instance, if an employee's start date is unknown, the "Start Date" field could be set to NULL until the correct data is obtained.

5.0.3 Inapplicable Data:

- Sometimes certain attributes are not applicable to certain tuples. For example, in a database of vehicles, a "Boat License Number" might be irrelevant for cars, so it would contain a NULL value for rows representing cars.

5.0.4 Optional Data:

- Some fields may be optional and not required to be filled out, leading to NULL values. For example, a database of students might have an optional field for "Middle Name," which could be NULL for students without one.

5.0.5 Default Initialization:

- When new records are inserted into a table, certain fields may not be given values explicitly, and they are initialized with NULL if no default value is specified.
- Example: Inserting a new product without specifying the "Discount" field might lead to it having a NULL value.

5.0.6 Outer Joins in SQL:

- In query results from **outer joins**, NULL values can occur when there is no matching data in one of the tables. For example, in a left join between two tables (e.g., customers and orders), if a customer has no orders, the fields corresponding to the order information will contain NULLs in the result set.

5.0.7 Aggregate Functions with No Results:

- When aggregate functions (like `SUM`, `COUNT`, `AVG`) are used on an empty result set, they might return NULL to represent the absence of data.
- Example: Calculating the `AVG(Salary)` in an empty department might return NULL because there are no salaries to average.

Therefore, NULL values in a relation typically arise from missing, unknown, inapplicable, or optional data, as well as from query results involving certain types of joins or aggregate functions. They signify that the information is currently unavailable or not relevant for that specific tuple.

6 Relational and Non-Relational Tables

6.0.1 Table that is a relation

A table that follows the rules of the **relational model** (e.g., no duplicate rows, each attribute contains atomic values, attributes have unique names, and order

Table 1: Relational Table

EmployeeID	Name	Department	Salary
101	Alice	HR	50000
102	Bob	Engineering	70000
103	Carol	Marketing	60000

Table 2: Non-Relational Table

OrderID	Customer	Items	Quantity	OrderID
201	John	Phone, Headphones	2, 1	201
202	Maria	Laptop	1	202
201	John	Phone, Headphones	2, 1	201

of tuples/columns does not matter).

Employee Table This table is a relation because:

1. **Unique Rows**: No two rows are identical; each employee has a unique `EmployeeID`.
2. **Atomic Values**: Each attribute contains atomic (indivisible) values (e.g., no multiple values in a single cell).
3. **No Ordering Requirement**: The order of rows or columns does not matter.
4. **Unique Attribute Names**: Each column has a distinct name.

6.0.2 Table that is not a relation

A table that **does not follow** the rules of the relational model (e.g., it contains duplicate rows, non-atomic values, or columns with the same name).

Orders Table This table is **not a relation** because:

1. **Duplicate Rows**: The first and last rows are identical, violating the uniqueness of tuples.
2. **Non-Atomic Values**: The `Items` and `Quantity` columns contain multiple values in a single cell (e.g., `Phone, Headphones`), which violates the atomicity requirement.
3. **Duplicate Column Names**: The table contains two columns with the name `OrderID`, which is not allowed in the relational model.

These violations make this table non-relational.

7 Set Operations

Given: $A = \{1, 3, 4, 5, 7\}$ $B = \{1, 2, 3, 5, 8, 9\}$

A intersection B The intersection of two sets is the set of elements that are common to both A and B.

$$A \cap B = \{1, 3, 5\}$$

A union B The union of two sets is the set of all distinct elements that are in either A, B, or both.

$$A \cup B = \{1, 2, 3, 4, 5, 7, 8, 9\}$$

A - B (A difference B) The difference of two sets (A - B) is the set of elements that are in A but not in B.

$$A - B = \{4, 7\}$$

A complement (A')

For simplicity, assume $U = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (the union of elements in both A and B).

$$A' = U - A = \{2, 6, 8, 9\}$$

A x B (Cartesian product of A and B) The Cartesian product $A \times B$ is the set of all ordered pairs (a, b), where $a \in A$ and $b \in B$.

$$A \times B = \begin{Bmatrix} (1,1), (1,2), (1,3), (1,5), (1,8), (1,9), \\ (3,1), (3,2), (3,3), (3,5), (3,8), (3,9), \\ (4,1), (4,2), (4,3), (4,5), (4,8), (4,9), \\ (5,1), (5,2), (5,3), (5,5), (5,8), (5,9), \\ (7,1), (7,2), (7,3), (7,5), (7,8), (7,9) \end{Bmatrix}$$

8 Dimension and Cardinality

Given the **Course** relation with the following data below:

- **Dimension**: This is the number of attributes (columns) in the relation.
  - The relation has 4 attributes: `Title`, `Room`, `Time`, and `Instructor`.
  - **Dimension**: 4
- **Cardinality**: This is the number of tuples (rows) in the relation.
  - The relation has 4 tuples (rows).
  - **Cardinality**: 4

Title	Room	Time	Instructor
Calculus 1	Jones Hall 110	1:00	Dr. Smith
Calculus 1	Ramsey Hall 236	2:00	Dr. Adams
Calculus 2	Jones Hall 120	3:00	Dr. Williams
History 111	Lambert Hall 325	1:00	Dr. Roberts

A1	A2	A3	A4
1	1	1	1
2	1	3	1
3	2	3	4
4	1	3	1

9 Domain of the Time Attribute

Based on the provided data, the `Time` attribute includes the following values:

- 1:00
- 2:00
- 3:00

Therefore, the domain of the `Time` attribute is:
Domain of Time={1:00,2:00,3:00}

10 Evaluating the Section Predicate where the Value of A1 is greater than 2

Given the relation R:
The tuples where A1 is greater than 2 are:
The result of the selection predicate $\sigma_{A1>2}$ is:

$$\{(3, 2, 3, 4), (4, 1, 3, 1)\}$$

A1	A2	A3	A4
3	2	3	4
4	1	3	1