Database Management System

Unit 4: Relational Database Design Process

Relational Database Design Process

4.1 E. F. Codd's Rule

4.2 Functional Dependency

4.3 Anomalies in Database Design:

- 4.3.1 Redundancy
- 4.3.2 Insertion
- 4.3.3 Updating
- 4.3.4 Deletion

4.4 Decomposition of Relation

4.5 Lossless Join and Dependency Preservation Property

4.6 Normalization:

- 4.6.1 First Normal Form
- 4.6.2 Second Normal Form
- 4.6.3. Third Normal Form

CE: 4.1
E. F. Codd's Rule

4.1. E. F. Codd's Rule

Dr. Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database.

These rules can be applied on any database system that manages stored data using only its relational capabilities.

This is a foundation rule, which acts as a base for all the other rules.

4.1. E. F. Codd's Rule

Rule 1: Information Rule

- The data stored in a database, may be user data or metadata, must be a value of some table cell.

- Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

- Every single data element value is guaranteed to be accessible logically with a combination of table-name, primary-key rowvalue, and attribute-name columnvalue.

- No other means, such as pointers, can be used to access data.

4.1. E. F. Codd's Rule (Conti….)

Rule 3: Systematic Treatment of NULL Values

- The NULL values in a database must be given a systematic and uniform treatment.
- This is a very important rule because a NULL can be interpreted as one the following − data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

- The structure description of the entire database must be stored in an online catalog, known as data dictionary, which can be accessed by authorized users.
- Users can use the same query language to access the catalog which they use to access the database itself.

4.1. E. F. Codd's Rule (Conti….)

Rule 5: Comprehensive Data Sub-Language Rule

- A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations.
- This language can be used directly or by means of some application.
- If the database allows access to data without any help of this language, then it is considered as a violation.

Rule 6: View Updating Rule

- All the views of a database, which can theoretically be updated, must also be updatable by the system.

4.1. E. F. Codd's Rule (Conti….)

Rule 7: High-Level Insert, Update, and Delete Rule

- A database must support high-level insertion, updation, and deletion.
- This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

- The data stored in a database must be independent of the applications that access the database.
- Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

4.1. E. F. Codd's Rule (Conti….)

Rule 9: Logical Data Independence

- The logical data in a database must be independent of its user's view application.
- Any change in logical data must not affect the applications using it.
- For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rule to apply.

Rule 10: Integrity Independence

- A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in the application.

4.1. E. F. Codd's Rule (Conti….)

This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

- The end-user must not be able to see that the data is distributed over various locations.
- Users should always get the impression that the data is located at one site only.
- This rule has been regarded as the foundation of distributed database systems.

4.1. E. F. Codd's Rule (Conti….)

Rule 12: Non-Subversion Rule

- If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.

CE: 4.2

Functional Dependency

4.2. Functional Dependency

- Functional dependency FD is a set of constraints between two attributes in a relation.

- Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

- Functional dependency is represented by an arrow sign → that is, X→Y, where X functionally determines Y.

- The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms

If F is a set of functional dependencies then the closure of F, denoted as F+, is the set of all functional dependencies logically implied by F.
Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

1. **Reflexivity rule:**
   If α is a set of attributes and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ holds.

2. **Augmentation rule:**
   If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.

Armstrong's Axioms (Conti….)

3. Transitivity rule:

If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

4. Union rule:

If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds.

5. Decomposition rule:

If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds.

6. Pseudotransitivity rule:

If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds.

CE: 4.3
Anomalies in Database Design

CE: 4.3 Anomalies in Database Design

There are different types of anomalies which can occur in referencing:

1. Redundancy Anomaly
2. Insertion Anomaly
3. Updation Anomaly
4. Deletion Anomaly

CE: 4.3 Anomalies in Database Design

For Example:

A college runs many classes. Each class may be taught by several teachers, and a teacher may teach several classes. A particular class always uses the same room. Because classes may meet at different times or on different evenings, it is possible for different classes to use the same room.

For that the database design can be

College(Name, Address, City)
Class(Class, ClassTeacher, Room)
Teacher(Name, Specilization, DOB, Gender, Address, City)
Room(RoomNo, Size)
Teach(Teacher, Class)
Allocation(Class, Room)

Let take simple as…

Teaching(Class, Teacher, RoomNo, Size)

CE: 4.3 Anomalies in Database Design

If records are being inserted in…

Teaching(Class, Teacher, RoomNo, Size)

Class	Teacher	RoomNo	Size
BE(IT Sem-1)	Prof. Sharma	C301	300
BE(IT Sem-1)	Dr. Patel	C301	300
BE(IT Sem-1)	Prof. Desai	C301	300
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300

CE: 4.3 Anomalies in Database Design

Now one more room with room no C304 and size 300 SQFT is available.

Class	Teacher	RoomNo	Size
BE(IT Sem-1)	Prof. Sharma	C301	300
BE(IT Sem-1)	Dr. Patel	C301	300
BE(IT Sem-1)	Prof. Desai	C301	300
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300

CE: 4.3 Anomalies in Database Design

OUTPUT

Class	Teacher	RoomNo	Size
BE(IT Sem-1)	Prof. Sharma	C301	300
BE(IT Sem-1)	Dr. Patel	C301	300
BE(IT Sem-1)	Prof. Desai	C301	300
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300
		C304	**300**

Insertion Anomaly

CE: 4.3 Anomalies in Database Design

Now suppose size of C301 is changed from 300 to 350.

Class	**Teacher**	**RoomNo**	**Size**
BE(IT Sem-1)	Prof. Sharma	C301	300
BE(IT Sem-1)	Dr. Patel	C301	300
BE(IT Sem-1)	Prof. Desai	C301	300
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300
		C304	**300**

Insertion Anomaly

CE: 4.3 Anomalies in Database Design

OUTPUT

Class	Teacher	RoomNo	Size
BE(IT Sem-1)	Prof. Sharma	C301	350
BE(IT Sem-1)	Dr. Patel	C301	350
BE(IT Sem-1)	Prof. Desai	C301	350
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300
		C304	**300**

Updation Anomaly

Insertion Anomaly

CE: 4.3 Anomalies in Database Design

Suppose Classes of BE(IT Sem-2) is over. So we have to delete it form the table.

Class	Teacher	RoomNo	Size
BE(IT Sem-1)	Prof. Sharma	C301	350
BE(IT Sem-1)	Dr. Patel	C301	350
BE(IT Sem-1)	Prof. Desai	C301	350
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300
		C304	**300**

Updation Anomaly

Insertion Anomaly

CE: 4.3 Anomalies in Database Design

OUTPUT

Class	Teacher	RoomNo	Size
BE(IT Sem-1)	Prof. Sharma	C301	350
BE(IT Sem-1)	Dr. Patel	C301	350
BE(IT Sem-1)	Prof. Desai	C301	350
BE(IT Sem-2)	Prof. Anjali	C302	250
BE(IT Sem-2)	Dr. Neha	C302	250
BE(IT Sem-2)	Prof. Desai	C302	250
BE(IT Sem-3)	Prof. Desai	C303	300
		C304	**300**

Updation Anomaly

Deletion Anomaly

Insertion Anomaly

CE: 4.4

Decomposition of Relation

Features of Good Relational Design:

1. Design Alternative: Larger Schemas
2. Design Alternative: Smaller Schemas
3. Lossy decompositions
4. Lossless decompositions

1. Design Alternative: Larger Schemas

Consider the following University Schema

classroom(building, room number, capacity)
department(dept name, building, budget)
course(course id, title, dept name, credits)
instructor(ID, name, dept name, salary)
section(course id, sec id, semester, year, building, room number, time slot id)
teaches(ID, course id, sec id, semester, year)
student(ID, name, dept name, tot cred)
takes(ID, course id, sec id, semester, year, grade)
advisor(s ID, i ID)
time slot(time slot id, day, start time, end time)
prereq(course id, prereq id)

1. Design Alternative: Larger Schemas

Suppose that instead of having the schemas of instructor and department as….

department(deptname, building, budget)
instructor(ID, name, deptname, salary)

we have the schema:

Int_dept (ID, name, salary, deptname, building, budget)

1. Design Alternative: Larger Schemas

Int_dept (ID, name, salary, deptname, building, budget)

ID	*name*	*salary*	*dept_name*	*building*	*budget*
22222	Einstein	95000	Physics	Watson	70000
12121	Wu	90000	Finance	Painter	120000
32343	El Said	60000	History	Painter	50000
45565	Katz	75000	Comp. Sci.	Taylor	100000
98345	Kim	80000	Elec. Eng.	Taylor	85000
76766	Crick	72000	Biology	Watson	90000
10101	Srinivasan	65000	Comp. Sci.	Taylor	100000
58583	Califieri	62000	History	Painter	50000
83821	Brandt	92000	Comp. Sci.	Taylor	100000
15151	Mozart	40000	Music	Packard	80000
33456	Gold	87000	Physics	Watson	70000
76543	Singh	80000	Finance	Painter	120000

This seems like a good idea because some queries can be expressed using fewer joins.

1. Design Alternative: Larger Schemas

❖ **Problems with large schema:**

1. The department information (“building” and “budget”) for each instructor in the department is repeating.
2. In the original design, we stored the amount of each budget exactly once.
3. This suggests that using **instdept** is a bad idea, since it stores the budget amounts redundantly. **[REDUNDANCY]**
4. There is also a risk that, if some user update the budget amount in one tuple, but not in all, and it create inconsistency. **[UPDATION ANOMALY]**
5. Suppose we are creating a new department in the university.
   In the alternative design above, we cannot INSERT the information concerning a **department (deptname, building, budget)** unless that department has at least one instructor at the university. **[INSERTION ANOMALY]**

2. Design Alternative: Smaller Schemas

- It does not mean that smaller schema is always good.

- Suppose we decompose

  employee(ID, name, street, city, salary)

  Into

  employee1(ID, name) & employee2 (name, street, city, salary)

  It is a Lossy Decomposition.

3. Lossy decompositions

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

employee

ID	name
⋮	
57766	Kim
98776	Kim
⋮	

name	street	city	salary
⋮			
Kim	Main	Perryridge	75000
Kim	North	Hampton	67000
⋮			

natural join

ID	name	street	city	salary
⋮				
57766	Kim	Main	Perryridge	75000
57766	Kim	North	Hampton	67000
98776	Kim	Main	Perryridge	75000
98776	Kim	North	Hampton	67000
⋮				

3. Lossy decompositions (Conti…)

As we have seen in the above figure…

- The two original tuples appear in the result along with two new tuples, that incorrectly mix data values affecting to the two employees named **“Kim”**.
- We actually have less information, but we have more tuples after joins.
- Such decompositions are called **lossy decompositions**.

CE: 4.5

Lossless Join

4. Lossless decompositions

- A decomposition {R1, R2,…, Rn} of a relation R is called a lossless decomposition for R, if the natural join of R1, R2,…, Rn produces exactly the relation R.

- <u>For Example:</u>
  Decomposition of R = (A, B, C) into…
  R1 = (A, B)   R2 = (B, C)

A	B	C
α	1	A
β	2	B

R

A	B
α	1
β	2

R1

B	C
1	A
2	B

R2

Natural Join of R1 & R2 is

A	B	C
α	1	A
β	2	B

which is same as the original relation. Therefore it's a Lossless decomposition.

4. Lossless decompositions (Conti…)

- **Decompositions should always be lossless.**
- **Lossless decomposition** ensure that the information in the original relation can be exactly reconstructed based on the information represented in the decomposed relations.

Class Work

- Is the following is a lossless decomposition? Justify your answer.

Employee	Project	Branch
Brown	Mars	L.A.
Green	Jupiter	San Jose
Green	Venus	San Jose
Hoskins	Saturn	San Jose
Hoskins	Venus	San Jose

Employee	Branch
Brown	L.A.
Green	San Jose
Hoskins	San Jose

Project	Branch
Mars	L.A.
Jupiter	San Jose
Saturn	San Jose
Venus	San Jose

Answer

- No it’s a lossly decomposition.
- After Natural Join, we get two extra tuples. Thus, we are loosing some information.

After Natural Join

Employee	Project	Branch
Brown	Mars	L.A.
Green	Jupiter	San Jose
Green	Venus	San Jose
Hoskins	Saturn	San Jose
Hoskins	Venus	San Jose
Green	Saturn	San Jose
Hoskins	Jupiter	San Jose

Original Relation

Employee	Project	Branch
Brown	Mars	L.A.
Green	Jupiter	San Jose
Green	Venus	San Jose
Hoskins	Saturn	San Jose
Hoskins	Venus	San Jose

CE: 4.6
Normalization

CE: 4.6 Normalization

And the solution is…..

Normalization

- **Normalization** is a process of decomposition of a relation (table) into more relations to avoid database anomalies (Insertion, updation, deletion)

- Normalization is based on **Functional Dependency**.

CE: 4.6 Normalization

And the solution is…..

Normalization

- **Normalization** is a process of decomposition of a relation (table) into more relations to avoid database anomalies (Insertion, updation, deletion)

- Normalization is based on **Functional Dependency**.


Thank you