Tutorial 5
CS4.301: Data and Applications

12th November, 2025

Agenda

- FDs
- Normalization
- Joins

Definition. A **functional dependency**, denoted by $X \rightarrow Y$, between two sets of attributes X and Y that are subsets of R specifies a *constraint* on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, they must also have $t_1[Y] = t_2[Y]$.

This means that the values of the Y component of a tuple in r depend on, or are *determined by*, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or **functionally**) *determine* the values of the Y component. We also say that there is a functional dependency from X to Y, or that Y is **functionally dependent** on X. The abbreviation for functional dependency is **FD** or **f.d.** The set of attributes X is called the **left-hand side** of the FD, and Y is called the **right-hand side**.

Some Functional Dependency rules

- $A \rightarrow A$
- $\{A,B\} \rightarrow C$ does not imply $A \rightarrow C$; $\{A,B\} \rightarrow C$ does not imply $B \rightarrow C$
- $A \rightarrow \{B,C\}$ implies $A \rightarrow B$ and $A \rightarrow C$
- $\{A,B\} \rightarrow A$

A SINGLE RELATION WITH ALL DATA

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

What are the possible anomalies that can happen when we perform the basic operations ?

UPDATE, INSERT, DELETE

INSERT ANOMALY

- Consider the situation that a new employee joins the company, and is in the training phase. Since they don't belong to a department yet, we can't add their details if the employee department column doesn't allow NULL value.

UPDATE ANOMALY

- In the table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent.

DELETE ANOMALY

- If the company decides to remove a certain department, for example – D890. Since Maggie is the only employee currently in the department, removing that record would cause the loss of her details entirely.

NORMALIZATION

- It is the process of organizing data into different tables that are well connected to reduce data redundancy, insert anomalies, update anomalies and delete anomalies.
- The different normal forms (levels of normalization) are:

  - First normal form – 1NF
  - Second normal form – 2NF
  - Third normal form – 3NF
  - Boyce-Codd normal form - BCNF

First normal form (1NF)

- As per the rule of 1NF, any column in the relation should hold only atomic values, i.e., No multivalued attributes are allowed.

Consider the following table where an employee can have more than one phone number. This makes the column emp_mobile have multivalued attributes.

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
			9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
			8123450987

How do you decompose this to the First normal form ?

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
			9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
			8123450987

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

No more column entries with multiple values

Second normal form (2NF)

- A relation is in second normal form if the following properties hold:
  - The relation is in first normal form.
  - There exists no Partial Dependency, i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

In the above table, COURSE_FEE doesn't belong to any candidate key.
{COURSE_FEE} can't uniquely identify a record.
{COURSE_FEE, COURSE_NO} can't uniquely identify a record.
{COURSE_FEE, STUD_NO} can't uniquely identify a record.

We see that the only candidate key is {STUD_NO, COURSE_NO}.
But, COURSE_FEE depends on the COURSE_NO. Hence, a partial dependency exists.

DECOMPOSITION

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

Table 2

COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

Third normal form (3NF)

- A relation is in third normal form if the following properties hold:

  - The relation is in second normal form.
  - Transitive dependency of non-prime attributes on any super key, should be removed.
    i.e. superkey →non-prime attribute through transitive dependency is not allowed.

Note – If A→B and B→C are two FDs then A→C is called transitive dependency.

- Consider the following table

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

We see that EMP_STATE & EMP_CITY depend on EMP_ZIP.
EMP_ZIP is dependent on EMP_ID (which is the candidate key).

EMP_ZIP ⟶ {EMP_STATE, EMP_CITY}
EMP_ID ⟶ EMP_ZIP

Hence, EMP_ID ⟶ {EMP_STATE, EMP_CITY} (Transitive dependency)

DECOMPOSITION

- We remove the transitively dependent attribute(s) from the relation by placing the attribute(s) in a new relation along with a copy of the determinant.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Alternate Defn of 3NF

A relation is in 3NF if:

- It is in 2NF
- For every X→Y:
  - X is a Super Key, **or**
  - Y is a prime attribute

Boyce-Codd normal form (BCNF)

- A relation is in BCNF if the following properties hold:
  - The relation is in third normal form.
  - For every functional dependency $X \rightarrow Y$, X should be the super key of the table.

Course	Instructor	Room
Math101	Dr. Smith	Room A
Math101	Dr. Jones	Room B
CS101	Dr. Smith	Room A
CS101	Dr. Brown	Room C

The functional dependencies are:

{Course,Instructor} →▫Room
Room →Instructor

Candidate key: {Course,Instructor}

We can see that Room is not a super key by itself. Hence, the BCNF criteria is being violated.

Exercise: Verify that the given relation is in 3NF

DECOMPOSITION

Course_Rooms

Course	Room
Math101	Room A
Math101	Room B
CS101	Room A
CS101	Room C

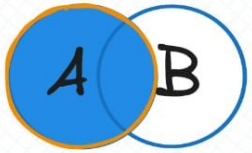
Room_Instructors

Room	Instructor
Room A	Dr. Smith
Room B	Dr. Jones
Room C	Dr. Brown

databites.tech @rfeers Josep Ferrer

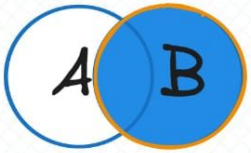
SQL JOINS

LEFT JOIN




```
SELECT *
FROM A
LEFT JOIN  B
ON A.KEY = B.KEY
```

RIGHT JOIN

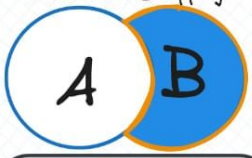



```
SELECT *
FROM A
RIGHT JOIN  B
ON A.KEY = B.KEY
```

LEFT JOIN (without overlapping)

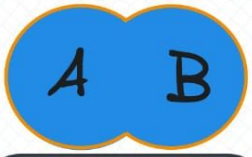

```
SELECT *
FROM A
LEFT JOIN B
ON A.KEY = B.KEY
WHERE B.KEY IS NULL
```

RIGHT JOIN (without overlapping)



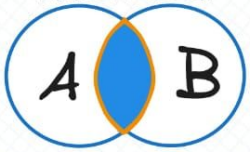

```
SELECT *
FROM A
RIGHT JOINB
ON A.KEY = B.KEY
WHERE A.KEY IS NULL
```

FULL JOIN




```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.KEY = B.KEY
```

INNER JOIN




```
SELECT *
FROM A
INNER JOIN   B
ON A.KEY = B.KEY
```

FULL JOIN (without overlapping)


```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.KEY = B.KEY
WHERE A.KEY IS NULL
   OR B.KEY IS NULL
```

The INNER JOIN keyword selects records that have matching values in both tables.

The INNER JOIN keyword returns only rows with a match in both tables.

Which means that if you have a Department with no Mgr_ssn, or with a Mgr_ssn that is not present in the EMPLOYEE table, that record would not be returned in the result.

Eg : SELECT e.Fname, e.Ssn , DEPARTMENT.Dname FROM EMPLOYEE AS e INNER JOIN DEPARTMENT ON DEPARTMENT.Mgr_ssn = e.Ssn;

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2).

The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

Eg : SELECT e.Fname, e.Ssn , DEPARTMENT.Dname FROM EMPLOYEE AS e LEFT JOIN DEPARTMENT ON DEPARTMENT.Mgr_ssn = e.Ssn;

```
mysql> SELECT e.Fname, e.Ssn , DEPARTMENT.Dname FROM EMPLOYEE AS e LEFT JOIN DEP
ARTMENT ON DEPARTMENT.Mgr_ssn = e.Ssn;
+----------+-----------+----------------+
| Fname    | Ssn       | Dname          |
+----------+-----------+----------------+
| John     | 123456789 | NULL           |
| Franklin | 333445555 | Research       |
| Joyce    | 453453453 | NULL           |
| Ramesh   | 666884444 | NULL           |
| James    | 888665555 | Headquarters   |
| Jennifer | 987654321 | Administration |
| Ahmad    | 987987987 | NULL           |
| Alicia   | 999887777 | NULL           |
+----------+-----------+----------------+
8 rows in set (0.00 sec)
```

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1).

The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

Eg : SELECT e.Fname, e.Ssn , DEPARTMENT.Dname FROM DEPARTMENT RIGHT JOIN EMPLOYEE AS e ON DEPARTMENT.Mgr_ssn = e.Ssn;

```
mysql> SELECT e.Fname, e.Ssn , DEPARTMENT.Dname FROM DEPARTMENT RIGHT JOIN EMPLO
YEE AS e ON DEPARTMENT.Mgr_ssn = e.Ssn;
+----------+-----------+----------------+
| Fname    | Ssn       | Dname          |
+----------+-----------+----------------+
| John     | 123456789 | NULL           |
| Franklin | 333445555 | Research       |
| Joyce    | 453453453 | NULL           |
| Ramesh   | 666884444 | NULL           |
| James    | 888665555 | Headquarters   |
| Jennifer | 987654321 | Administration |
| Ahmad    | 987987987 | NULL           |
| Alicia   | 999887777 | NULL           |
+----------+-----------+----------------+
8 rows in set (0.00 sec)
```

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

FULL OUTER JOIN can potentially return very large result-sets!

MySQL does not directly support FULL JOIN. However, you can achieve the same result by combining LEFT JOIN and RIGHT JOIN using UNION.

```
mysql> SELECT e.Fname, e.Ssn, d.Dname
    -> FROM EMPLOYEE AS e
    -> LEFT JOIN DEPARTMENT AS d ON d.Mgr_ssn = e.Ssn
    ->
    -> UNION
    ->
    -> SELECT e.Fname, e.Ssn, d.Dname
    -> FROM EMPLOYEE AS e
    -> RIGHT JOIN DEPARTMENT AS d ON d.Mgr_ssn = e.Ssn;
+----------+-----------+----------------+
| Fname    | Ssn       | Dname          |
+----------+-----------+----------------+
| John     | 123456789 | NULL           |
| Franklin | 333445555 | Research       |
| Joyce    | 453453453 | NULL           |
| Ramesh   | 666884444 | NULL           |
| James    | 888665555 | Headquarters   |
| Jennifer | 987654321 | Administration |
| Ahmad    | 987987987 | NULL           |
| Alicia   | 999887777 | NULL           |
+----------+-----------+----------------+
```