# RDBMS: Functional dependency and Normalization

By

Vaibhav P. Vasani

Assistant Professor

Department of Computer Engineering

K. J. Somaiya College of Engineering

Somaiya Vidyavihar University

# Informal Design Guidelines for Relational Databases (1)

- What is relational database design?
  - The grouping of attributes to form "good" relation schemas
- Two levels of relation schemas
  - The logical "user view" level
  - The storage "base relation" level
- Design is concerned mainly with base relations
- What are the criteria for "good" base relations?

# Informal Design Guidelines for Relational Databases (2)

- Concepts of functional dependencies and normal forms
  - - 1NF (First Normal Form)
  - - 2NF (Second Normal Form)
  - - 3NF (Third Normal Form)
  - - BCNF (Boyce-Codd Normal Form)

# Semantics of the Relation Attributes

- GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.
- <u>Bottom Line:</u> *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*

# Figure 10.1 A simplified COMPANY relational database schema

**EMPLOYEE**

F.K.

| | | | | |
|---|---|---|---|---|
| Ename | Ssn | Bdate | Address | Dnumber |

P.K.

**DEPARTMENT**

F.K.

| | | |
|---|---|---|
| Dname | Dnumber | Dmgr_ssn |

P.K.

**DEPT_LOCATIONS**

F.K.

| | |
|---|---|
| Dnumber | Dlocation |

P.K.

**PROJECT**

F.K.

| | | | |
|---|---|---|---|
| Pname | Pnumber | Plocation | Dnum |

P.K.

**WORKS_ON**

F.K. F.K.

| | | |
|---|---|---|
| Ssn | Pnumber | Hours |

P.K.

**Figure 10.1**
A simplified COMPANY relational database schema.

From Sudarshan Korth

# 1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

# EXAMPLE OF AN UPDATE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from "Billing" to "Customer-Accounting" may cause this update to be made for all 100 employees working on project P1.

# EXAMPLE OF AN INSERT ANOMALY

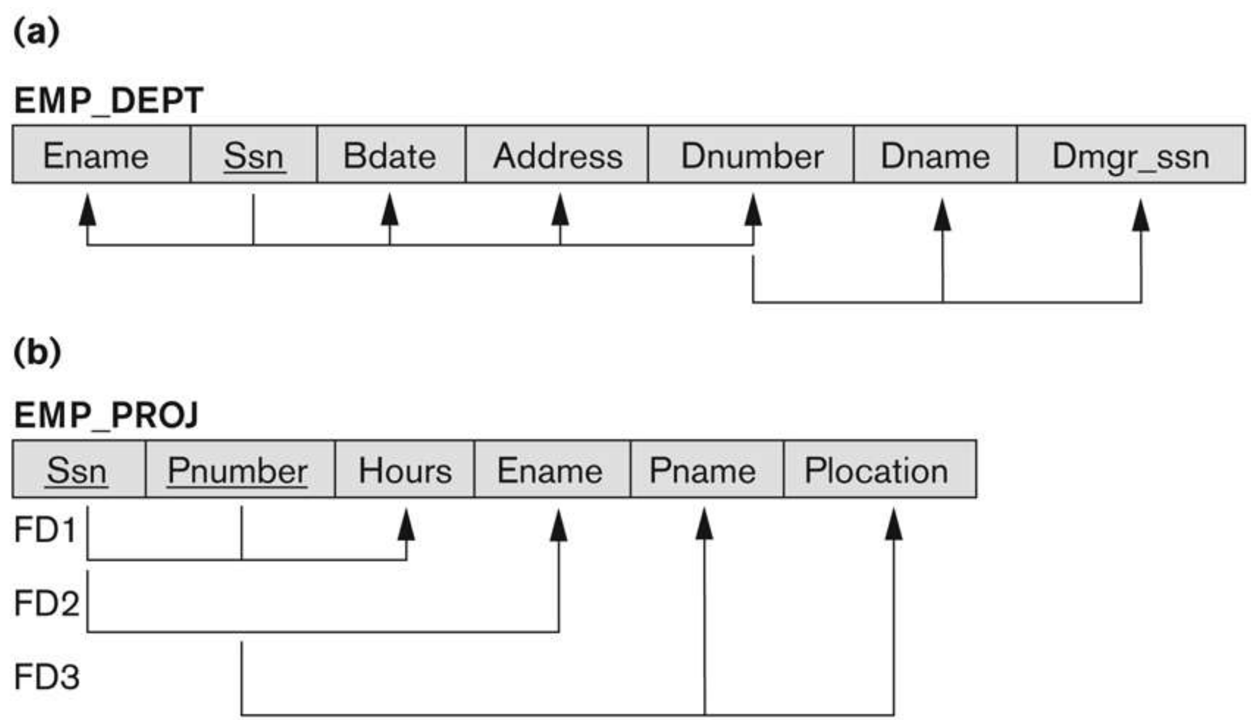
- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Insert  Anomaly:
  - Cannot insert a project unless an employee is assigned to it.
- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project.

# EXAMPLE OF AN DELETE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

# Figure 10.3 Two relation schemas suffering from update anomalies

**Figure 10.3**
Two relation schemas suffering from update anomalies.
(a) EMP_DEPT and
(b) EMP_PROJ.



# Figure 10.4 Example States for EMP_DEPT and EMP_PROJ

**Figure 10.4**
Example states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 10.2. These may be stored as base relations for performance reasons.

**EMP_DEPT**

| Ename | Ssn | Bdate | Address | Dnumber | Dname | Dmgr_ssn |
|---|---|---|---|---|---|---|
| Smith, John B. | 123456789 | 1965-01-09 | 731 Fondren, Houston, TX | 5 | Research | 333445555 |
| Wong, Franklin T. | 333445555 | 1955-12-08 | 638 Voss, Houston, TX | 5 | Research | 333445555 |
| Zelaya, Alicia J. | 999887777 | 1968-07-19 | 3321 Castle, Spring, TX | 4 | Administration | 987654321 |
| Wallace, Jennifer S. | 987654321 | 1941-06-20 | 291 Berry, Bellaire, TX | 4 | Administration | 987654321 |
| Narayan, Ramesh K. | 666884444 | 1962-09-15 | 975 FireOak, Humble, TX | 5 | Research | 333445555 |
| English, Joyce A. | 453453453 | 1972-07-31 | 5631 Rice, Houston, TX | 5 | Research | 333445555 |
| Jabbar, Ahmad V. | 987987987 | 1969-03-29 | 980 Dallas, Houston, TX | 4 | Administration | 987654321 |
| Borg, James E. | 888665555 | 1937-11-10 | 450 Stone, Houston, TX | 1 | Headquarters | 888665555 |

Redundancy (Dname, Dmgr_ssn)

**EMP_PROJ**

| Ssn | Pnumber | Hours | Ename | Pname | Plocation |
|---|---|---|---|---|---|
| 123456789 | 1 | 32.5 | Smith, John B. | ProductX | Bellaire |
| 123456789 | 2 | 7.5 | Smith, John B. | ProductY | Sugarland |
| 666884444 | 3 | 40.0 | Narayan, Ramesh K. | ProductZ | Houston |
| 453453453 | 1 | 20.0 | English, Joyce A. | ProductX | Bellaire |
| 453453453 | 2 | 20.0 | English, Joyce A. | ProductY | Sugarland |
| 333445555 | 2 | 10.0 | Wong, Franklin T. | ProductY | Sugarland |
| 333445555 | 3 | 10.0 | Wong, Franklin T. | ProductZ | Houston |
| 333445555 | 10 | 10.0 | Wong, Franklin T. | Computerization | Stafford |
| 333445555 | 20 | 10.0 | Wong, Franklin T. | Reorganization | Houston |
| 999887777 | 30 | 30.0 | Zelaya, Alicia J. | Newbenefits | Stafford |
| 999887777 | 10 | 10.0 | Zelaya, Alicia J. | Computerization | Stafford |
| 987987987 | 10 | 35.0 | Jabbar, Ahmad V. | Computerization | Stafford |
| 987987987 | 30 | 5.0 | Jabbar, Ahmad V. | Newbenefits | Stafford |
| 987654321 | 30 | 20.0 | Wallace, Jennifer S. | Newbenefits | Stafford |
| 987654321 | 20 | 15.0 | Wallace, Jennifer S. | Reorganization | Houston |
| 888665555 | 20 | Null | Borg, James E. | Reorganization | Houston |

Redundancy (Ename) Redundancy (Pname, Plocation)

# Guideline to Redundant Information in Tuples and Update Anomalies

- GUIDELINE 2:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

# 1.3 Null Values in Tuples

- GUIDELINE 3:
  - Relations should be designed such that their tuples will have as ***few NULL*** values as possible
  - Attributes that are NULL frequently could be placed in separate relations (with the primary key)
- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

# 1.4 Spurious Tuples

- **Bad designs** for a relational database may result in erroneous **results for certain JOIN operations**
- The "lossless join" property is used to guarantee meaningful results for join operations

- GUIDELINE 4:
  - The relations should be designed to satisfy the lossless join condition.
  - No spurious tuples should be generated by doing a natural-join of any relations.

- 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.

# Examples of FD constraints (1)

- Social security number determines employee name
  - SSN -> ENAME
- Project number determines project name and location
  - PNUMBER -> {PNAME, PLOCATION}
- Employee ssn and project number determines the hours per week that the employee works on the project
  - {SSN, PNUMBER} -> HOURS

# Examples of FD constraints (2)

- An FD is a property of the attributes in the schema R
- The constraint must hold on *every* relation instance r(R)
- If K is a key of R, then K functionally determines all attributes in R
  - (since we never have two distinct tuples with t1[K]=t2[K])

# Armstrong's Axioms
# or
# Inference Rules for FDs

- **Reflexive rule** − If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

# Inference Rules for FDs

- Some additional inference rules that are useful:
  - **Decomposition:** If X -> YZ, then X -> Y and X -> Z
  - **Union:** If X -> Y and X -> Z, then X -> YZ
  - **Psuedotransitivity:** If X -> Y and WY -> Z, then WX -> Z

# Inference Rules for FDs

- **Closure** of a set F of FDs is the set $F^+$ of all FDs that can be inferred from F
- **Closure** of a set of attributes X with respect to F is the set $X^+$ of all attributes that are functionally determined by X

# Trivial Functional Dependency

- **Trivial** − If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** − If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial** − If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

# Minimal Sets of FDs (1)

- A set of FDs is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency X -> A in F with a dependency Y -> A, where Y proper-subset-of X ( Y subset-of X) and still have a set of dependencies that is equivalent to F.

# Minimal Sets of FDs (2)

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set

# Normal Forms Based on Primary Keys

- First Normal Form
- Second Normal Form
- Third Normal Form

# Normalization of Relations

- **Normalization:**
  - The process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- **Normal form:**
  - Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

- **Update anomalies**
- **Deletion anomalies**
- **Insert anomalies**

# Normalization of Relations (2)

- 2NF, 3NF, BCNF
  - based on keys and FDs of a relation schema
- 4NF
  - based on keys, multi-valued dependencies : MVDs; 5NF based on keys, join dependencies : JDs (Chapter 11)
- Additional properties may be needed to ensure a good relational design (lossless join, dependency preservation; Chapter 11)

# Practical Use of Normal Forms

- **Normalization** is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- The practical utility of these normal forms becomes questionable when the constraints on which they are based are *hard to understand* or to *detect*
- The database designers *need not* normalize to the highest possible normal form
  - (usually up to 3NF, BCNF or 4NF)
- **Denormalization**:
  - The process of storing the join of higher normal form relations as a base relation—which is in a lower normal form

# Definitions of Keys and Attributes Participating in Keys

- A **superkey** of a relation schema R = {A1, A2, ...., An} is a set of attributes S *subset-of* R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]

- A **key** K is a **superkey** with the *additional property* that removal of any attribute from K will cause K not to be a superkey any more.

# Definitions of Keys and Attributes Participating in Keys

- If a relation schema has more than one key, each is called a **candidate** key.
  - One of the candidate keys is *arbitrarily* designated to be the **primary key**, and the others are called **secondary keys**.
- A **Prime attribute** must be a member of *some* candidate key
- A **Nonprime attribute** is not a prime attribute—that is, it is not a member of any candidate key.

# First Normal Form

- defines that all the attributes in a relation must have atomic domains.

| Course | Content |
|---|---|
| Programming | Java, c++ |
| Web | HTML, PHP, ASP |

| Course | Content |
|---|---|
| Programming | Java |
| Programming | c++ |
| Web | HTML |
| Web | PHP |
| Web | ASP |

# First Normal Form

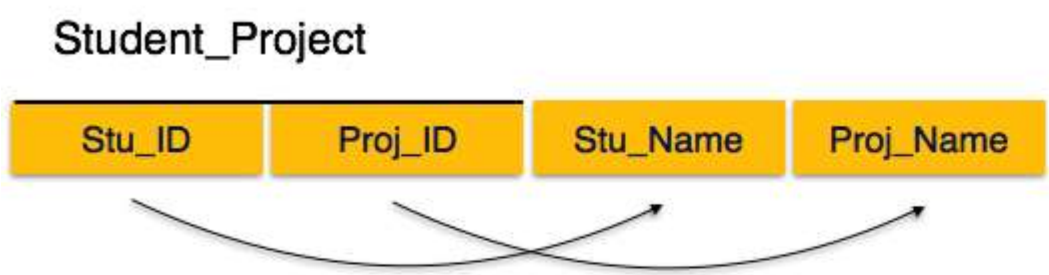
- Disallows
  - composite attributes
  - multivalued attributes
  - **nested relations**; attributes whose values for an *individual tuple* are non-atomic

- Considered to be part of the definition of relation

# Second Normal Form

- **Prime attribute** − An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute** − An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.

- every non-prime attribute should be fully functionally dependent on prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



Student

| Stu_ID | Stu_Name | Proj_ID |
| --- | --- | --- |

Project

| Proj_ID | Proj_Name |
| --- | --- |

# Third Normal Form

- For a relation to be in Third Normal Form, it must be in Second Normal form and the following must satisfy –
- No non-prime attribute is transitively dependent on prime key attribute.
- For any non-trivial functional dependency, X → A, then either –
  - X is a superkey or,
  - A is prime attribute.

Stu_ID → Zip → City, so there exists **transitive dependency**.

Student_Detail

| Stu_ID | Stu_Name | City | Zip |
|---|---|---|---|

Student_Detail

| Stu_ID | Stu_Name | Zip |
|---|---|---|

ZipCodes

| Zip | City |
|---|---|

# Boyce-Codd Normal Form

- an extension of Third Normal Form on strict terms. BCNF states that −
- For any non-trivial functional dependency, $X \rightarrow A$, X must be a super-key.

# Example

- Suppose there is a company wherein employees work in **more than one department**. They store the data like this

| emp_id | emp_nationality | emp_dept | dept_type | dept_no_of_emp |
|---|---|---|---|---|
| 1001 | Austrian | Production and planning | D001 | 200 |
| 1001 | Austrian | stores | D001 | 250 |
| 1002 | American | design and technical support | D134 | 100 |
| 1002 | American | Purchasing department | D134 | 600 |

- **Functional dependencies in the table above**:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}
- **Candidate key**: {emp_id, emp_dept}
- The table is not in BCNF as neither emp_id nor emp_dept alone are keys.
- To make the table comply with BCNF we can break the table in three tables like this

**emp_nationality table:**

| emp_id | emp_nationality |
|---|---|
| 1001 | Austrian |
| 1002 | American |

**emp_dept table:**

| emp_dept | dept_type | dept_no_of_emp |
|---|---|---|
| Production and planning | D001 | 200 |
| stores | D001 | 250 |
| design and technical support | D134 | 100 |
| Purchasing department | D134 | 600 |

| emp_dept_mapping table: | emp_dept |
|---|---|
| 1001 | Production and planning |
| 1001 | stores |
| 1002 | design and technical support |
| 1002 | Purchasing department |

**Functional dependencies**:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}
**Candidate keys**:
For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}
This is now in BCNF as in both the functional dependencies left side part is a key.

# Summery

- **First Normal Form (1NF)**
  - It should only have single(atomic) valued attributes/columns.
  - Values stored in a column should be of the same domain
  - All the columns in a table should have unique names.
  - And the order in which data is stored, does not matter.

# Summery continue..

- **Second Normal Form (2NF)**
  - It should be in the First Normal form.
  - And, it should not have Partial Dependency.
- **Third Normal Form (3NF)**
  - It is in the Second Normal form.
  - And, it doesn't have Transitive Dependency.
- **Boyce and Codd Normal Form (BCNF)**
  - R must be in 3rd Normal Form
  - and, for each functional dependency ( $X \rightarrow Y$ ), X should be a super Key.
- **Fourth Normal Form (4NF)**
  - It is in the Boyce-Codd Normal Form.
  - And, it doesn't have Multi-Valued Dependency.

- **Fifth Normal Form**
  - Relation should be already in 4NF.
  - It cannot be further non loss decomposed (join dependency)
- Sixth Normal Form or Domain Key Normal form
  - Every constraint on the relationship is dependent on key constraints and domain constraints where domain is set of allowable values for a column. This form prevents the insertion of any unacceptable data by enforcing constraints at the level of relationship, rather than at the table or column level. DKNF is less design model than an abstract "ultimate" Normal form.

| NF | Description |
| --- | --- |
| 1NF | Scaler values, No repeating values |
| 2NF | Non-key column depend on entire on PK |
| 3NF | Non-key column depend only on entire on PK |
| BCNF | Non key column can not depend on another non-key column<br>Prevent transitive dependency<br>A-> C, B-C, A and B depends on C, A and B moved to another table C will be key |
| 4NF | Table must not have more than one multivalued dependency, where PK has one to many relationship to non key column, this form get rid of misleading many to many relationship |
| 5NF | The Data structure is spilt into smaller and smaller tables until all redundancy has been eliminated. Id further splitting would result in tables until that could not be joined to recreate the original table, structure is in fifth normal form |
| DKNF or 6NF | Every constraint on the relationship is dependent on key constraints and domain constraints where domain is set of allowable values for a column. This form prevents the insertion of any unacceptable data by enforcing constraints at the level of relationship, rather than at the table or column level. DKNF is less design model than an abstract “ultimate” Normal form |

# Questions

# ?



vaibhav.vasani@gmail.com

# Thank You



vaibhav.vasani@gmail.com