Chapter – 5: Relational Database Design

Relational Database Design

Objective:

The goal of relational database design is to generate a set of schemes that allow us to

- Store information without unnecessary redundancy.
- Retrieve information easily (and accurately).

Introduction

- In a relational database, data is stored in tables.
- The designer of a relational database is concerned with the selection of a set of tables that best serves the need of the users.
- These tables, or more correctly the associated set of relation schemas, constitute a schema or conceptual model of the database.
- While selecting the conceptual model we should bear in mind that the information content of most databases is not static but is subjected to regular updating operations.

Relational Database Design (contd…)

- However, arbitrary modification of tuples will not be, in general, permissible. Even when the modified tuples are correct.
- Since the tuples of a relational database represent the current information about some real life object, the external world will impose some constraints on the admissible values of the tuples. These constraints essentially determine the semantics of data and are called ***integrity constraints***. While designing a relational data model, the database designer must be aware of all the ***integrity constraints*** that need to be satisfied.

Goal — Devise a Theory for the Following

- Decide whether a particular relation *R* is in "good" form.
- In the case that a relation *R* is not in "good" form, decompose it into a set of relations $\{R_1, R_2, \ldots, R_n\}$ such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition
- Our theory is based on:
  - Integrity constraints
  - functional dependencies
  - multivalued dependencies

Benefits

The benefits of a database that has been designed according to the relational model are numerous. Some of them are:

- Data entry, updates and deletions will be efficient.
- Data retrieval, summarization and reporting will also be efficient.
- Since the database follows a well-formulated model, it behaves predictably.
- Since much of the information is stored in the database rather than in the application, the database is somewhat self-documenting.
- Changes to the database schema are easy to make.

Integrity Constraints

- Integrity ensures that the data in a database is both accurate and complete, in other words, that the data makes sense. An integrity constraint can be any arbitrary predicate applied to the database. The data analysis stage will identify the requirements of these.
- Different types of integrity that needs to be considered:
- Domain constraints
  - Entity integrity
  - Column constraints
- Referential integrity
- User-defined integrity constraints
  - Trigger
  - Assertion

Domain Constraints

- A domain is defined as the set of all unique values permitted for an attribute. For example, a domain of date is the set of all possible valid dates, a domain of integer is all possible whole numbers, a domain of day-of-week is Sunday, Monday, Tuesday ...., Saturday
- This in effect is defining rules for a particular attribute. If it is determined that an attribute is a date then it should be implemented in the database to prevent invalid dates being entered.
- A classic example of this is where the data from a legacy system is loaded into a newly designed database. The new system is well designed. Columns that hold dates are defined as such whereas, in the old system, they were held as character strings. Much data is rejected because of invalid dates, eg 30 February 2000.
- If the system supports domain constraints then this invalid data would not have stored in the first place. That is, the integrity of the database is being preserved.

Syntax:

Create table employee (eid number(3) primary key, ename char(20) NOT NULL, age number(2) check age between 30 and 40, sex char(1) check sex in ('M','F','m','f'));

Domain Constraints (Contd….)

Entity Integrity

- A requirement of E F Codd in his seminal paper is that a primary key of an entity, or any part of it, can never take a null value.
- Oracle, and most other relational database management systems, will enforce this.

Column Constraints

- During the data analysis phase, business rules will identify any column constraints. For example, a salary cannot be negative; an employee number must be in the range 1000 - 2000, etc.
- Oracle, and some other RDBMSs, will allow storage of these rules within the database itself. That is, in a central data dictionary

Foreign keys

- A foreign key is a reference to a key in another relation, meaning that the referencing table has, as one of its attributes, the values of a key in the referenced table. Foreign keys need not have unique values in the referencing relation. Foreign keys effectively use the values of attributes in the referenced relation to restrict the domain of one or more attributes in the referencing relation.
- A foreign key could be described formally as: "For all tables in the referencing relation projected over the referencing attributes, there must exist a table in the referenced relation projected over those same attributes such that the values in each of the referencing attributes match the corresponding values in the referenced attributes."

Referential Integrity

- This is the most common type of integrity constraint. This is used to manage the relationships between primary and foreign keys.

Referential integrity is best illustrated by an example.

- Let's assume the department and employee entities have been implemented as tables in a relational database system.
- When entering a new employee, the department in which they work needs to be specified. Department number is the foreign key in the employee table and the primary key in the department table.
- In order to preserve the integrity of the data in the database there are a set of rules that need to be observed:
- If inserting an employee in the employee table, the insertion should only be allowed if their department number exists in the department table
- If deleting a department in the department table, the deletion should only be allowed if there are no employees working in that department
- If changing the value of a department number in the department table, the update should only be allowed if there are no employees working in the department whose number is being changed
- If changing the value of a department number in the employee table, the update should only be allowed if the new value exists in the department table
- If any of the above is allowed to happen then we have data in an inconsistent state. If the integrity of the data is compromised - the data does not make sense.

Referential Integrity

- Oracle, and other relational database management systems, will allow enforcement of referential integrity rules. If implemented, and a user tries to break any of the rules, an error message will be given and the change will not take place.
  - create table customer (cname char(20) not null, street char(30), city char(30), primary key (cname));
  - create table branch (bname char(15) not null, bcity char(30), assets integer, primary key (bname) check (assets >= 0));
  - create table account (account# char(10) not null, bname char(15), balance integer, primary key (account#) foreign key (bname) references branch, check (balance >= 0));
  - create table depositor (cname char(20) not null, account# char(10) not null, primary key (cname, account#), foreign key (cname) references customer(cname), foreign key (account#) references account (account#));

User-Defined Integrity Constraints

- Business rules may dictate that when a specific action occurs further actions should be triggered. For example, deletion of a record automatically writes that record to an audit table.

Triggers

- A trigger is a statement that is automatically executed by the system as a side effect of a modification to the database.
- We need to
  - Specify the conditions under which the trigger is executed.
  - Specify the actions to be taken by the trigger.
- There are four types of database triggers:
  i. Table-level triggers can initiate activity before or after an INSERT, UPDATE, or DELETE event.
  ii. View-level triggers define what can be done to the view.
  iii. Database-level triggers can be activated at startup and shutdown of a database.
  iv. Session-level triggers can be used to store specific information.

User-Defined Integrity Constraints (contd…)

Assertions

- An assertion is a predicate expressing a condition we wish the database to always satisfy.
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
  - This testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Domain constraints, functional dependency and referential integrity are special forms of assertion.
- Where a constraint cannot be expressed in these forms, we use an assertion, e.g.
  - Ensuring the sum of loan amounts for each branch is less than the sum of all account balances at the branch.
  - Ensuring every loan customer keeps a minimum of $1000 in an account.

Anomalies

Anomalies present a slightly more complex concept. Anomalies are problems that arise in the data due to a flaw in the database design.

In any database insertion, deletion or modification that leaves the database in an inconsistent state is said to have caused an anomaly.

There are three types of anomalies that may arise, and we will consider how they occur with the flawed schema shown in above table.

Database Anomolies

- The Problems resulting from data redundancy in an un-normalized database table are collectively known as **anomalies**. So any database insertion, deletion or modification that leaves the database in an inconsistent state is said to have caused an anomaly.
- Three types of Anomaly to guard against:
  - insert
  - delete
  - update

- Consider the Schema

EmployeeDepartment(Eid, Name,JoiningDate,DeptID,DepartmentName)

Eid	Name	Joining Date	DeptID	Department Name
1	Ram Shrestha	2010/01/10	42	Account
2	Shayam Mishra	2011/05/18	41	Administrator
3	Mohan Awasthi	2011/06/20	43	Development
4	Mohan Karmacharya	2010/10/20	41	Administrator

Insertion Anomalies

- Insertion anomalies occur when we try to insert data into a flawed table. Imagine that we have a new employee starting at the company.
  - When we insert the employee's details into the employeeDepartment table, we must insert both his department id and his department name.
  - What happens if we insert data that does not match what is already in the table, for example, by entering an employee as working for Department 42, Development? It will not be obvious which of the rows in the database is correct. This is an insertion anomaly.
  - Let us assume that a new department has been started by the organization but initially there is no employee appointed for the department, then the tuple for this department cannot be inserted into this table as the Eid will have NULL, Which is not allowed as Eid is primary key

Deletion Anomalies

- Deletion anomalies occur when we delete data from a flawed schema. Imagine that all the employees of Department 41 leave on the same day (walking out in disgust, perhaps). When we delete these employee records, we no longer have any record that Department 41 exists or what its name is. This is a deletion anomaly.

Update Anomalies

- Update anomalies occur when we change data in a flawed schema. Imagine that Department 41 decides to change its name to Emerging Technologies. We must change this data for every employee who works for this department. We might easily miss one. If we do miss one (or more), this is an update anomaly.

Functional Dependencies

- Constraints on the set of legal relations.
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- A functional dependency is a generalization of the notion of a *key.*
- If there is a functional dependency between column A and column B in a given table, which may be written A —> B, then the value of column A determines the value of column B. For example, in the employee table, the employeeID functionally determines the name

Definition

A functional dependency is defined as a constraint between two sets of attributes in a relation from a database.

Given a relation *R*, a set of attributes *X* in *R* is said to **functionally determine** another attribute *Y*, also in *R*, (written $X \rightarrow Y$) if and only if each *X* value is associated with at most one *Y* value.

In other words….

X is the *determinant set* and *Y* is the *dependent attribute*. Thus, given a tuple and the values of the attributes in *X*, one can determine the corresponding value of the *Y* attribute.

Example

Employee

SSN	Name	JobType	DeptName
557-78-6587	Rohan	Accountant	Salary
214-45-2398	Mohan	Engineer	Product

Note:

Name is functionally dependent on SSN because an employee's name can be uniquely determined from their SSN.

Name does not determine SSN, because more than one employee can have the same name..

Keys

Whereas a key is a set of attributes that uniquely identifies an entire tuple, a functional dependency allows us to express constraints that uniquely identify the values of certain attributes.

However, a candidate key is always a determinant, but a determinant doesn't need to be a key.

Closure

Let a relation *R* have some functional dependencies *F* specified. The *closure of F* (usually written as ***F+***) is the set of all functional dependencies that may be logically derived from *F*. Often *F* is the set of most obvious and important functional dependencies and ***F+***, the closure, is the set of all the functional dependencies including *F* and those that can be deduced from *F*. The closure is important and may, for example, be needed in finding one or more candidate keys of the relation.

Example

Student

SNo	SName	CNo	CName	Addr	Instr.	Office
5425	Rohan	102	Calc I	New Road	Mira	B42 Room 112
7845	Mohan	541	Bio 10	Thamel	Rojesh	B24 Room 210

SNo -> SName
SNo -> Addr

CNo -> CName
CNo -> Instr

Instr -> Office

Axioms

- Before we can determine the closure of the relation, Student, we need a set of rules.
- Developed by Armstrong in 1974, there are six rules (axioms) that all possible functional dependencies may be derived from them.

Axioms Cont.

1. *Reflexivity Rule* --- If *X* is a set of attributes and *Y* is a subset of *X*, then $X \rightarrow Y$ holds.

   each subset of *X* is functionally dependent on *X*.

1. *Augmentation Rule* --- If $X \rightarrow Y$ holds and *W* is a set of attributes, then $WX \rightarrow WY$ holds.

2. *Transitivity Rule* --- If $X \rightarrow Y$ and $Y \rightarrow Z$ holds, then $X \rightarrow Z$ holds.

Derived Theorems from Axioms

4. *Union Rule* --- If $X \rightarrow Y$ and $X \rightarrow Z$ holds, then $X \rightarrow YZ$ holds.
5. *Decomposition Rule* --- If $X \rightarrow YZ$ holds, then so do $X \rightarrow Y$ and $X \rightarrow Z$.
6. *Pseudotransitivity Rule* --- If $X \rightarrow Y$ and $WY \rightarrow Z$ hold then so does $WX \rightarrow Z$.

Back to the Example

SNo	SName	CNo	CName	Addr	Instr.	Office

Based on the rules provided, the following dependencies can be derived.

(SNo, CNo) → ***SNo*** (Rule 1) -- subset
(SNo, CNo) → ***CNo*** (Rule 1)
(SNo, CNo) → ***(SName, CName)*** (Rule 2) -- augmentation
CNo → ***office*** (Rule 3) -- transitivity
SNo → ***(SName, address)*** (Union Rule)
etc.

Closure of a Set of Functional Dependencies

- Given a set *F* set of functional dependencies, there are certain other functional dependencies that are logically implied by *F*.
  - E.g. If $A \rightarrow B$ and $B \rightarrow C$, then we can infer that $A \rightarrow C$
- The set of all functional dependencies logically implied by *F* is the *closure* of *F*.
- We denote the *closure* of *F* by F^+.
- We can find all of F^+ by applying Armstrong's Axioms:
  - if $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$ **(reflexivity)**
  - if $\alpha \rightarrow \beta$, then $\gamma\ \alpha \rightarrow \gamma\ \beta$ **(augmentation)**
  - if $\alpha \rightarrow \beta$, and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$ **(transitivity)**
- These rules are
  - sound (generate only functional dependencies that actually hold) and
  - complete (generate all functional dependencies that hold).

Example

- $R = (A, B, C, G, H, I)$
  $F = \{ A \rightarrow B$
  $A \rightarrow C$
  $CG \rightarrow H$
  $CG \rightarrow I$
  $B \rightarrow H\}$
- some members of F^+
  - $A \rightarrow H$ (by transitivity from $A \rightarrow B$ *and* $B \rightarrow H$)
  - $AG \rightarrow I$ (by augmenting $A \rightarrow C$ with G, to get $AG \rightarrow CG$ and then transitivity with $CG \rightarrow I$)
  - $CG \rightarrow HI$
    - from $CG \rightarrow H$ *and* $CG \rightarrow I$: "union rule" can be inferred from
      - definition of functional dependencies, or
      - Augmentation of $CG \rightarrow I$ to infer $CG \rightarrow CGI$, augmentation of $CG \rightarrow H$ to infer $CGI \rightarrow HI$, and then transitivity

Closure of Functional Dependencies (Cont.)

- We can further simplify manual computation of F^+ by using the following additional rules.
  - If $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds, then $\alpha \rightarrow \beta\gamma$ holds **(union)**
  - If $\alpha \rightarrow \beta\gamma$ holds, then $\alpha \rightarrow \beta$ holds and $\alpha \rightarrow \gamma$ holds **(decomposition)**
  - If $\alpha \rightarrow \beta$ holds and $\gamma\beta \rightarrow \delta$ holds, then $\alpha\gamma \rightarrow \delta$ holds **(pseudotransitivity)**

  The above rules can be inferred from Armstrong's axioms.