**11/4/2019**

**Good and Bad Database Design**

**1. Introduction**

* Bad database designs:
	1. Unnecessary redundancy: inefficient storage (storing same information, 110 is the manager of dept D123, more than once.)
		+ State: storing same information, more than once.)
		+ Elaborate: inefficient storage, may contribute to anomaly; just D123 is not *unnecessary* redundancy.
			- There are necessary redundancy: accessibility, fault tolerance, …
		+ Exemplify: 110 is the manager of dept D123
	2. Anomaly: conflicts in data, difficulties in maintenance.

***Example:***

Consider the relation (poor designed): Not in third normal form (3NF)

EMPLOYEE(EMP\_NO, NAME, DEPT\_NO, MANAGER\_NO).

Assumptions made:

1. Every employee works for only one department.
2. Every department has only one manager.
3. Every manager manages only one department.
4. Every employee is represented as a tuple in the EMPLOYEE relation.
5. Every employee has an unique EMP\_NO.

Thus, EMP\_NO is a candidate key.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** | **MANAGER\_NO** |
| 101 | Lady Gaga | *D123* | *110* |
| 122 | Brad Pitts | *D123* | *110* |
| 140 | Lebron James | *D123* | *110* |
| 155 | Narendra Modi | D225 | 205 |
| 167 | Jennifer Lopez | D225 | 205 |
| 311 | John Smiths | D337 | 333 |

**Problem:**

* Unnecessary redundancy: the fact Manager 110 manages department D123 is stored three times.

**Update Anomaly:**

(a) 415 is the new manager of department D123

* Inefficiency in update.
* Potential inconsistency.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** | **MANAGER\_NO** |
| 101 | Lady Gaga | D123 | **415** |
| 122 | Brad Pitts | D123 | **415** |
| 140 | Lebron James | D123 | ***110 ??*** |
| 155 | Narendra Modi | D225 | 205 |
| 167 | Jennifer Lopez | D225 | 205 |
| 311 | John Smiths | D337 | 333 |

SQL: it works but not efficient.

UPDATE employee
SET MANAGER\_NO = 415
WHERE DEPT\_NO 'D123';

(b) Jennifer Lopez changes to working for department D337:

* Need to know the manager of D337.
* Potential inconsistency.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** | **MANAGER\_NO** |
| 101 | Lady Gaga | D123 | 110 |
| 122 | Brad Pitts | D123 | 110 |
| 140 | Lebron James | D123 | 110 |
| 155 | Narendra Modi | D225 | 205 |
| 167 | Jennifer Lopez | **D337** | *205 ??<-- must change to 333 too.* |
| 311 | John Smiths | D337 | 333 |

The SQL command:

UPDATE Employee
SET DEPT\_NO = 'D337'
WHERE NAME = 'Jennifer Lopez';

will produces inconsistent result.

You need to update both DEPT\_NO and MANAGER\_NO. However,

UPDATE Employee
SET DEPT\_NO = 'D337',
     MANAGER\_NO = (SELECT DISTINCT MANAGER\_NO FROM Employee WHERE DEPT\_NO = 'D337')
WHERE NAME = 'Jennifer Lopez';

will not work in MySQL as one cannot include a SELECT clause on the same table in the SET clause.

The correct one, may use session variables in MySQL:

SELECT MANAGER\_NO INTO @mid -- session variable
FROM Employee
WHERE NAME = 'Jennifer Lopez';

UPDATE Employee
SET DEPT\_NO = 'D337', MANAGER\_NO = @mid
WHERE NAME = 'Jennifer Lopez';

Example:

mysql> set @major = 'CSCI';

Query OK, 0 rows affected (0.04 sec)

mysql> select @major;

+--------+

| @major |

+--------+

| CSCI |

+--------+

1 row in set (0.03 sec)

mysql> SELECT \* FROM student where major = @major;

+--------+-------+-------+-------+-------+---------+---------+

| stuId | fname | lname | major | minor | credits | advisor |

+--------+-------+-------+-------+-------+---------+---------+

| 100000 | Tony | Hawk | CSCI | CINF | 40 | 1011 |

| 100001 | Mary | Hawk | CSCI | CINF | 35 | 1011 |

| 100002 | David | Hawk | CSCI | ITEC | 66 | 1011 |

+--------+-------+-------+-------+-------+---------+---------+

3 rows in set (0.03 sec)

**Insertion Anomaly:**

Creating a new department D777, with manager 520, currently with no employee is not possible.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** | **MANAGER\_NO** |
| 101 | Lady Gaga | D123 | 110 |
| 122 | Brad Pitts | D123 | 110 |
| 140 | Lebron James | D123 | 110 |
| 155 | Narendra Modi | D225 | 205 |
| 167 | Jennifer Lopez | D225 | 205 |
| 311 | John Smiths | D337 | 333 |
| **???? no null value acceptable** | **????** | **D777** | **520** |

EMP\_ID, as PK, cannot be null.

**Deletion Anomaly:**

John Smiths no longer works here. Result: the information that 333 is the manager of department D337 is lost.

John Smiths was the last employee in D337.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** | **MANAGER\_NO** |
| 101 | Lady Gaga | D123 | 110 |
| 122 | Brad Pitts | D123 | 110 |
| 140 | Lebron James | D123 | 110 |
| 155 | Narendra Modi | D225 | 205 |
| 167 | Jennifer Lopez | D225 | 205 |

Two collections of information:

1. Employee: EMP\_NO, NAME, DEPT\_NO
2. Department: DEPT\_NO, MANAGER\_NO

A standard way of resolving these issues are by proper **decomposition**: breaking down a relation into two or more relations.

Example: decomposition into two relations:

EMP(EMP\_NO, NAME, DEPT\_NO): employee information only.

|  |  |  |
| --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D225 |
| 167 | Jennifer Lopez | ~~D225~~ D337 |
| ~~311~~ | ~~John Smiths~~ | ~~D337~~ |

DEPARTMENT(DEPT\_NO, MANAGER\_NO): department information only

|  |  |
| --- | --- |
| **DEPT\_NO** | **MANAGER\_NO** |
| D123 | 110 |
| D225 | 205 |
| D337 | 333 |
| D777 | 520 |

To obtain the original relation EMPLOYEE(EMP\_NO, NAME, DEPT\_NO, MANAGER\_NO) from

EMP(EMP\_NO, NAME, DEPT\_NO)
DEPARTMENT(DEPT\_NO, MANAGER\_NO)

Relational algebra:

EMPLOYEE = EMP |x| DEPARTMENT

SQL:

SELECT EMP.\*, DEPARTMENT.MANAGER\_NO INTO EMPLOYEE
FROM EMP, DEPARTMENT
WHERE EMP.DEPT\_NO = DEPARTMENT.DEPT\_N

SELECT EMP.\*, DEPARTMENT.MANAGER\_NO INTO EMPLOYEE
FROM EMP INNER JOIN DEPARTMENT
 ON(EMP.DEPT\_NO = DEPARTMENT.DEPT\_NO);

1. There is no loss of information.
2. No previously mentioned redundancy and anomaly.

**Methods for good designs**

* Integrity Rules:  Constraints for avoiding impossible configurations.
* Normal Forms:  A set of rules for designing good relation schemes.

**Integrity Rules:**

* Most of the integrity rules are application-dependent.
* Need to analyze the semantic of the applications to find out the integrity rules.
* These are known as *Database-Specific Integrity Rules*. (Domain specific or application specific)

***Examples***:

1. Student Id should be a seven digit number
2. Date of Birth should be greater than 1900.
3. The room number of Delta Building should start with a 'D'.
4. A student cannot take more than 24 credits in any semester.
5. A student must show proof of meningitis shot before registration for the first semester.

**General Integrity Rules:**

* Should be satisfied by *every* database, but not necessarily enforced by every DBMS package.
* *Entity Integrity Rule* and *Referential Integrity Rule*: based on the concepts of primary keys, superkeys, and foreign keys

**Entity Integrity Rule**

* Entity Integrity: no*component* of a *candidate key* of a relation can have a null value.
* Meaning: In a relational database, a tuple that cannot be identified will not be stored.

***Example:***

EMPLOYEE(EMP\_NO, NAME, DEPT\_NO, SALARY): IF SSN is a CK, then it cannot have a null value.

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_NO (PK is a CK)** | **NAME** | **DEPT\_NO** | **SALARY** |
| 101 | Lady Gaga | D123 | 550000000 |
| 122 | Brad Pitts | D123 | 101000 |
| 140 | Lebron James | D123 | 50000000 |
| 155 | Narendra Modi | @: OK | @: OK, not a CK |
| *@: Not OK because it is a CK* | Jennifer Lopez | D225 | 2000000 |
| *@* | John Smiths | D337 | 70000 |

* If EMP\_NO is a candidate key, this EMPLOYEE instance does not satisfy the entity integrity rule.
* If we accept the relation instance above as valid, then EMP\_NO cannot be a candidate key.

Example:

CREATE TABLE IF NOT EXISTS Enroll(
   stuId      INT NOT NULL,
   classId      INT NOT NULL,
   grade      VARCHAR(2),
   CONSTRAINT Enroll\_classId\_stuId\_pk PRIMARY KEY (classId, stuId),
 -- composite primary key
   CONSTRAINT Enroll\_classNumber\_fk FOREIGN KEY (classId)
      REFERENCES Class(classId) ON DELETE CASCADE,
   CONSTRAINT Enroll\_stuId\_fk FOREIGN KEY (stuId)
      REFERENCES Student (stuId) ON DELETE CASCADE
);

Replaced by:

CK: 1. classId, stuId: CK => Non-null
 2. EnrollId: surrogate primary key 🡺 not null.

CREATE TABLE IF NOT EXISTS Enroll(
 EnrollId INT NOT NULL, -- …
   stuId      INT NOT NULL, -- entity integrity rule
   classId    INT NOT NULL, -- entity integrity rule
   grade      VARCHAR(2),
 CONSTRAINT Enroll\_pk PRIMARY KEY (ENrollId)
   CONSTRAINT Enroll\_classId\_stuId\_ck UNIQUE (classId, stuId),
 -- CK should be unique
   CONSTRAINT Enroll\_classNumber\_fk FOREIGN KEY (classId)
      REFERENCES Class(classId) ON DELETE CASCADE,
   CONSTRAINT Enroll\_stuId\_fk FOREIGN KEY (stuId)
      REFERENCES Student (stuId) ON DELETE CASCADE
);

**Referential Integrity Rule**

* Referential integrity rule: the database does not contain any *unmatched* *non-null foreign key* values.
* Any non-null value of a foreign key K must appear in the base (host) relation where K is a candidate key.

***Example:***

EMP(EMP\_NO, NAME, DEPT\_NO): DEPT\_NO is FK.

|  |  |  |
| --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO: FK** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | *@: OK* |
| 155 | Narendra Modi | D225 |
| 167 | Jennifer Lopez | D225 |
| 311 | John Smiths | D337 |

DEPARTMENT(DEPT\_NO, MANAGER\_NO): DEPT\_NO is the PK of Department

|  |  |
| --- | --- |
| **DEPT\_NO: PK** | **MANAGER\_NO** |
| D123 | 110 |
| D225 | 205 |
| D337 | 333 |

* DEPT\_NO is a foreign key in EMP, referencing DEPT\_NO in DEPARTMENT.
* The referential integrity rule is satisfied.
* Note that DEPT\_NO may be null in EMP.

***Example:***

EMP(EMP\_NO, NAME, DEPT\_NO)

|  |  |  |
| --- | --- | --- |
| **EMP\_NO** | **NAME** | **DEPT\_NO** |
| 101 | Lady Gaga | D123 |
| 122 | Brad Pitts | D123 |
| 140 | Lebron James | D123 |
| 155 | Narendra Modi | D225 |
| 167 | Jennifer Lopez | D225 |
| 311 | John Smiths | D337 |
| **350** | **Bun Yue** | **D119** |

Violates the referential integrity

DEPARTMENT(DEPT\_NO, MANAGER\_NO)

|  |  |
| --- | --- |
| **DEPT\_NO** | **MANAGER\_NO** |
| D123 | 110 |
| D225 | 205 |
| D337 | 333 |

* The referential integrity rule is not satisfied.

mysql> INSERT INTO Student VALUES

 -> (200000,'Tom','Yue','CSCI','MAGIC',70,2011);

ERROR 1406 (22001): Data too long for column 'minor' at row 1

Minor is VARCHAR(4).

Note:

1. In practical DBMS, pay attention when the referential integrity rule is enforced.
2. For example, in MySQL, only the INNODB data/storage engine may enforce the referential integrity rule.
3. If the DBMS does not enforce the referential integrity rule (e,g. CVS, memory storage engines, etc), it will be the task of the DB developer to do so.

1NF, 2NF, 3NF, BCNF, 4NF, 5NF, DKNF, 6NF.

BCNF = Boyce Codd (3.5NF)