The Relational Model

Mine eye hath play'd the painter and hath stell'd
Thy beauty's form in table of my heart.

Shakespeare, Sonnet XXIV

Why Study the Relational Model?

- **Most widely used model.**
  - Vendors: IBM, Informix, Microsoft, Oracle, Sybase, etc.
- **"Legacy systems" in older models**
  - e.g., IBM's IMS
- **Object-oriented concepts have recently merged in**
  - *object-relational model*
    - Informix, IBM DB2, Oracle 8i
    - Early work done in POSTGRES research project at Berkeley

Relational Database: Definitions

- *Relational database:* a set of *relations.*
- *Relation:* made up of 2 parts:
  - *Schema* : specifies name of relation, plus name and type of each column.
    - E.g. Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real)
  - *Instance* : a *table,* with rows and columns.
    - #rows = *cardinality*
    - #fields = *degree / arity*
- Can think of a relation as a *set* of rows or *tuples*.
  - i.e., all rows are distinct

Ex: Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

- Cardinality = 3, arity = 5 , all rows distinct
- Do all values in each column of a relation instance have to be distinct?

SQL - A language for Relational DBs

- **SQL* (a.k.a. "Sequel"), standard language**
- **Data Definition Language (DDL)**
  - create, modify, delete relations
  - specify constraints
  - administer users, security, etc.
- **Data Manipulation Language (DML)**
  - Specify *queries* to find tuples that satisfy criteria
  - add, modify, remove tuples

* Structured Query Language

SQL Overview

```
• CREATE TABLE <name> ( <field> <domain>, … )

• INSERT INTO <name> (<field names>)
       VALUES (<field values>)

• DELETE FROM <name>
        WHERE <condition>

• UPDATE <name>
      SET <field name> = <value>
    WHERE <condition>

• SELECT <fields>
     FROM <name>
    WHERE <condition>
```

Creating Relations in SQL

- **Creates the Students relation.**
  - Note: the type (domain) of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

```
CREATE TABLE Students
    (sid CHAR(20),
     name CHAR(20),
     login CHAR(10),
     age INTEGER,
     gpa FLOAT)
```


Table Creation (continued)

- **Another example: the Enrolled table holds information about courses students take.**

```
CREATE TABLE Enrolled
    (sid CHAR(20),
     cid CHAR(20),
     grade CHAR(2))
```


Adding and Deleting Tuples

- **Can insert a single tuple using:**

```
INSERT INTO Students (sid, name, login, age, gpa)
 VALUES  ('53688', 'Smith', 'smith@cs', 18, 3.2)
```

- **Can delete all tuples satisfying some condition (e.g., name = Smith):**

```
DELETE
  FROM Students S
 WHERE S.name = 'Smith'
```

Powerful variants of these commands are available; more later!

Keys

- Keys are a way to associate tuples in different relations
- Keys are one form of integrity constraint (IC)

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

FORIEGN Key

PRIMARY Key

Primary Keys

- **A set of fields is a *superkey* if:**
  - No two distinct tuples can have same values in all key fields
- **A set of fields is a *key* for a relation if :**
  - It is a superkey
  - No subset of the fields is a superkey
- **what if >1 key for a relation?**
  - one of the keys is chosen (by DBA) to be the ***primary key***. Other keys are called ***candidate*** keys.
- **E.g.**
  - *sid* is a key for Students.
  - What about *name*?
  - The set {*sid, gpa*} is a superkey.

Primary and Candidate Keys in SQL

- **Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.**

- Keys must be used carefully!
- "For a given student and course, there is a single grade."

```
CREATE TABLE Enrolled
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY (sid,cid))
```

vs.

```
CREATE TABLE Enrolled
   (sid CHAR(20)
    cid  CHAR(20),
    grade CHAR(2),
    PRIMARY KEY  (sid),
    UNIQUE (cid, grade))
```

"Students can take only one course, and no two students in a course receive the same grade."

Foreign Keys, Referential Integrity

- ***Foreign key*** **: Set of fields in one relation that is used to `refer' to a tuple in another relation.**
  - Must correspond to the primary key of the other relation.
  - Like a `logical pointer'.
- **If all foreign key constraints are enforced, *referential integrity* is achieved (i.e., no dangling references.)**

Foreign Keys in SQL

Example: Only students listed in the Students relation should be allowed to enroll for courses.

- *sid* is a foreign key referring to Students:

```
CREATE TABLE Enrolled
(sid CHAR(20),cid CHAR(20),grade CHAR(2),
 PRIMARY KEY (sid,cid),
 FOREIGN KEY (sid) REFERENCES Students )
```

Enrolled

sid	cid	grade
53666	15-101	C
53666	18-203	B
53650	15-112	A
53666	15-105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@cs	18	3.2
53650	Smith	smith@math	19	3.8

Enforcing Referential Integrity

- **Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.**
- **What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)**
- **What should be done if a Students tuple is deleted?**
  - Also delete all Enrolled tuples that refer to it?
  - Disallow deletion of a Students tuple that is referred to?
  - Set sid in Enrolled tuples that refer to it to a *default sid*?
  - (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null,* denoting `*unknown*' or `*inapplicable*'.)
- **Similar issues arise if primary key of Students tuple is updated.**

Integrity Constraints (ICs)

- **IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.***
  - ICs are specified when schema is defined.
  - ICs are checked when relations are modified.
- **A *legal* instance of a relation is one that satisfies all specified ICs.**
  - DBMS should not allow illegal instances.
- **If the DBMS checks ICs, stored data is more faithful to real-world meaning.**
  - Avoids data entry errors, too!

Where do ICs Come From?

- ICs are based upon the semantics of the real-world that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we can NEVER infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances!
  - From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.
- Key and foreign key ICs are the most common; more general ICs supported too.

Relational Model: Summary

- A tabular representation of data.
- Simple and intuitive, currently the most widely used
  - Object-relational variant gaining ground
- Integrity constraints can be specified by the DBA, based on application semantics. DBMS checks for violations.
  - Two important ICs: primary and foreign keys
  - In addition, we *always* have domain constraints.
- Mapping from ER to Relational is (fairly) straightforward.
- NEXT: FILES< STORAGE, BUFFERS, DISKS…
- READ CHAPTER 9!

Schema Refinement and Normalization

> Nobody realizes that some people expend tremendous energy merely to be normal.
>
> Albert Camus

Functional Dependencies (FDs)

- **A functional dependency X → Y holds over relation schema R if, for every allowable instance *r* of R:**

  $t1 \in r,\ t2 \in r,\ \pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$

  (where t1 and t2 are tuples;X and Y are sets of attributes)
- **In other words: X → Y means**

  Given any two tuples in *r*, if the X values are the same, then the Y values must also be the same. (but not vice versa)
- **Can read "→" as "determines"**

FD's Continued

- **An FD is a statement about *all* allowable relations.**
  - Must be identified based on semantics of application.
  - Given some instance *r1* of R, we can check if *r1* violates some FD *f*, but we cannot determine if *f* holds over R.
- **Question: How related to keys?**
- **if "K → all attributes of R" then K is a *superkey* for R**

  (does not require K to be *minimal*.)
- **FDs are a generalization of keys.**

Normal Forms

- **Back to schema refinement…**
- **Q1: is any refinement needed??!**
- **If a relation is in a *normal form* (BCNF, 3NF etc.):**
  - we know that certain problems are avoided/minimized.
  - helps decide whether decomposing a relation is useful.
- **Role of FDs in detecting redundancy:**
  - Consider a relation R with 3 attributes, ABC.
    - No (non-trivial) FDs hold: There is no redundancy here.
    - Given A → B: If A is not a key, then several tuples could have the same A value, and if so, they'll all have the same B value!
- **1st Normal Form – all attributes are atomic**
- **1st ⊃2nd (of historical interest) ⊃ 3rd ⊃ Boyce-Codd ⊃ …**

Boyce-Codd Normal Form (BCNF)

- **Reln R with FDs *F* is in BCNF if, for all X → A in F+**
  - $A \in X$ (called a *trivial* FD), or
  - X is a superkey for R.
- **In other words: "R is in BCNF if the only non-trivial FDs over R are *key constraints.*"**
- **If R in BCNF, then every field of every tuple records information that cannot be inferred using FDs alone.**
  - Say we know FD X → A holds this example relation:

X	Y	A
x	y1	a
x	y2	?

- Can you guess the value of the missing attribute?

- Yes, so relation is not in BCNF

Decomposition of a Relation Scheme

- If a relation is not in a desired normal form, it can be *decomposed* into multiple relations that each are in that normal form.
- Suppose that relation R contains attributes *A1 ... An.* A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R, and
  - Every attribute of R appears as an attribute of at least one of the new relations.

Example (same as before)

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Hourly_Emps

- SNLRWH has FDs S $\rightarrow$ SNLRWH and R $\rightarrow$ W
- Q: Is this relation in BCNF?

No, The second FD causes a violation;
W values repeatedly associated with R values.

Decomposing a Relation

- Easiest fix is to create a relation RW to store these associations, and to remove W from the main schema:

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

Hourly_Emps2

R	W
8	10
5	7

Wages

- Q: Are both of these relations are now in BCNF?
- **Decompositions should be used only when needed.**
  - Q: potential problems of decomposition?

Problems with Decompositions

- There are three potential problems to consider:
  1) May be impossible to reconstruct the original relation! (Lossiness)
     - Fortunately, not in the SNLRWH example.
  2) Dependency checking may require joins.
     - Fortunately, not in the SNLRWH example.
  3) Some queries become more expensive.
     - e.g., How much does Guldu earn?

***Tradeoff*: Must consider these issues vs. redundancy.**

Review – Natural Join

- **Natural Join is a fundamental operator of relational algebra** $\bowtie$
- **Semantics of R $\bowtie$ S are:**
  - Compute Cartesian Product of R and S
  - Select out those tuples where the common attributes of R and S have the same values
  - Keep all unique attributes of these tuples plus one copy of each of the common attributes.

Lossless Decomposition (example)

S	N	L	R	H
123-22-3666	Attishoo	48	8	40
231-31-5368	Smiley	22	8	30
131-24-3650	Smethurst	35	5	30
434-26-3751	Guldu	35	5	32
612-67-4134	Madayan	35	8	40

$\bowtie$

R	W
8	10
5	7

=

S	N	L	R	W	H
123-22-3666	Attishoo	48	8	10	40
231-31-5368	Smiley	22	8	10	30
131-24-3650	Smethurst	35	5	7	30
434-26-3751	Guldu	35	5	7	32
612-67-4134	Madayan	35	8	10	40

Lossy Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

A	B
1	2
4	5
7	2

B	C
2	3
5	6
2	8

$A \rightarrow B$; $C \rightarrow B$

A	B
1	2
4	5
7	2

$\bowtie$

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8
1	2	8
7	2	3

Lossless Join Decompositions

- **Decomposition of R into X and Y is *lossless-join* w.r.t. a set of FDs F if, for every instance r that satisfies F:**
  $\pi_X(r) \bowtie \pi_Y(r) = r$
- **It is always true that** $r \subseteq \pi_X(r) \bowtie \pi_Y(r)$
  - In general, the other direction does not hold! If it does, the decomposition is lossless-join.
- **Definition extended to decomposition into 3 or more relations in a straightforward way.**
- ***It is essential that all decompositions used to deal with redundancy be lossless! (Avoids Problem #1)***

More on Lossless Decomposition

- The decomposition of R into X and Y is **lossless with respect to F** *if and only if* the closure of F contains:
  $X \cap Y \to X$, **or**
  $X \cap Y \to Y$

  in example: decomposing ABC into AB and BC is lossy, because intersection (i.e., "B") is not a key of either resulting relation.
- Useful result: If $W \to Z$ holds over R and $W \cap Z$ is empty, then decomposition of R into R-Z and WZ is loss-less.

Lossless Decomposition (example)

A	B	C
1	2	3
4	5	6
7	2	8

⇒

A	C
1	3
4	6
7	8

B	C
2	3
5	6
2	8

$A \to B$; $C \to B$

A	C
1	3
4	6
7	8

$\bowtie$

B	C
2	3
5	6
2	8

=

A	B	C
1	2	3
4	5	6
7	2	8

But, now we can't check $A \to B$ without doing a join!

Dependency Preserving Decomposition

- **Dependency preserving decomposition (Intuitive):**
  - If R is decomposed into X, Y and Z, and we enforce the FDs that hold individually on X, on Y and on Z, then all FDs that were given to hold on R should also hold. *(Avoids Problem #2 on our list.)*
- ***Projection of set of FDs F*: If R is decomposed into X and Y the projection of F on X (denoted F_X) is the set of FDs $U \to V$ in F^+ (*closure of F, not just F*) such that all of the attributes U, V are in X. *(same holds for Y of course)***

Dependency Preserving Decompositions (Cont.)

- ***Definition:* Decomposition of R into X and Y is *dependency preserving* if $(F_X \cup F_Y)^+ = F^+$**
  - i.e., if we consider only dependencies in the closure F^+ that can be checked in X without considering Y, and in Y without considering X, these imply all dependencies in F^+.
- **Important to consider F^+ in this definition:**
  - ABC, $A \to B$, $B \to C$, $C \to A$, decomposed into AB and BC.
  - Is this dependency preserving? Is $C \to A$ preserved?????
    - note: F^+ contains $F \cup \{A \to C, B \to A, C \to B\}$, so…
- F_{AB} contains $A \to B$ and $B \to A$; F_{BC} contains $B \to C$ and $C \to B$
- So, $(F_{AB} \cup F_{BC})^+$ contains $C \to A$

Decomposition into BCNF

- **Consider relation R with FDs F. If $X \to Y$ violates BCNF, decompose R into R - Y and XY (guaranteed to be loss-less).**
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
  - e.g., CSJDPQV, key C, $JP \to C$, $SD \to P$, $J \to S$
  - *{contractid, supplierid, projectid,deptid,partid, qty, value}*
  - To deal with $SD \to P$, decompose into SDP, CSJDQV.
  - To deal with $J \to S$, decompose CSJDQV into JS and CJDQV
  - So we end up with: SDP, JS, and CJDQV
- **Note: several dependencies may cause violation of BCNF. The order in which we ``deal with'' them could lead to very different sets of relations!**

BCNF and Dependency Preservation

- **In general, there may not be a dependency preserving decomposition into BCNF.**
  - e.g., CSZ, CS → Z, Z → C
  - Can't decompose while preserving 1st FD; not in BCNF.
- **Similarly, decomposition of CSJDPQV into SDP, JS and CJDQV is not dependency preserving (w.r.t. the FDs JP → C, SD → P and J → S).**
- ***{contractid, supplierid, projectid,deptid,partid, qty, value}***
  - However, it is a lossless join decomposition.
  - In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.
    - but JPC tuples are stored only for checking the f.d. (*Redundancy!*)

Third Normal Form (3NF)

- **Reln R with FDs *F* is in 3NF if, for all X → A in F⁺**

  A ∈ X (called a *trivial* FD), or

  X is a superkey of R, or

  A is part of some candidate key (not superkey!) for R. (sometimes stated as "A is *prime*")
- ***Minimality* of a key is crucial in third condition above!**
- **If R is in BCNF, obviously in 3NF.**
- **If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good'' decomp, or performance considerations).**
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

What Does 3NF Achieve?

- **If 3NF violated by X → A, one of the following holds:**
  - X is a subset of some key K ("partial dependency")
    - We store (X, A) pairs redundantly.
    - e.g. Reserves SBDC (C is for credit card) with key SBD and S → C
  - X is not a proper subset of any key. ("transitive dep.")
    - There is a chain of FDs K → X → A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value (different K's, same X implies same A!) – problem with initial SNLRWH example.
- **But: even if R is in 3NF, these problems could arise.**
  - e.g., Reserves SBDC (note: "C" is for credit card here), S → C, C → S is in 3NF (why?), but for each reservation of sailor S, same (S, C) pair is stored.
- **Thus, 3NF is indeed a compromise relative to BCNF.**

Decomposition into 3NF

- **Obviously, the algorithm for lossless join decomp into BCNF can be used to obtain a lossless join decomp into 3NF (typically, can stop earlier) but does not ensure dependency preservation.**
- **To ensure dependency preservation, one idea:**
  - If X → Y is not preserved, add relation XY.

  Problem is that XY may violate 3NF! e.g., consider the addition of CJP to `preserve' JP → C. What if we also have J → C ?
- **Refinement: Instead of the given set of FDs F, use a *minimal cover for F*.**

Minimal Cover for a Set of FDs

- ***Minimal cover* G for a set of FDs F:**
  - Closure of F = closure of G.
  - Right hand side of each FD in G is a single attribute.
  - If we modify G by deleting an FD or by deleting attributes from an FD in G, the closure changes.
- **Intuitively, every FD in G is needed, and ``*as small as possible*'' in order to get the same closure as F.**
- **e.g., A → B, ABCD → E, EF → GH, ACDF → EG has the following minimal cover:**
  - A → B, ACD → E, EF → G and EF → H
- **M.C. implies Lossless-Join, Dep. Pres. Decomp!!!**
  - (in book)

Summary of Schema Refinement

- **BCNF: each field contains information that cannot be inferred using only FDs.**
  - ensuring BCNF is a good heuristic.
- **Not in BCNF? Try decomposing into BCNF relations.**
  - Must consider whether all FDs are preserved!
- **Lossless-join, dependency preserving decomposition into BCNF impossible? Consider 3NF.**
  - Same if BCNF decomp is unsuitable for typical queries
  - Decompositions should be carried out and/or re-examined while keeping *performance requirements* in mind.
- **Note: even more restrictive Normal Forms exist (we don't cover them in this course, but some are in the book.)**