CS 2451
Database Schema Design: Normalization

http://www.seas.gwu.edu/~bhagiweb/cs2541

Spring 2020

Instructor: Dr. Bhagi Narahari

Summary of Schema Design thus far….

- Desirable properties of relational schemas
  - Avoid anamolies (exhaustive searches), lossless join, NULLs,…
- Concept of functional dependencies
  - X → Y : X determines Y

    If two tuples have same value in columns/attributes X, they have same value in columns Y
  - Functional dependencies depend on meaning and domain of attributes and not based on an instance of the DB
- Closure of set of dependencies
  - Armstrong's Axioms – infer sets of dependencies starting with a small set
- Attribute set closure X^+
  - All attributes determined by set X

    If X^+ = all attributes, then X is key – math. definition of key

Recall: Functional Dependencies

- Functional dependencies represent constraints on the values of attributes in a relation and are used in normalization.
- A ***functional dependency*** (abbreviated ***FD***) is a statement about the relationship between attributes in a relation. We say a set of attributes *X* functionally determines an attribute *Y* if given the values of *X* we always know the only possible value of *Y*.
  - Notation: $X \rightarrow Y$
  - *X* functionally determines *Y*
  - *Y* is functionally dependent on *X*
- Example:
  - eno → ename
  - eno, pno → hours

Sets of Functional Dependencies

- Some obvious functional dependencies
  - {SSN} → {NAME, ADDRESS, DNUMBER}
  - {DNUMBER} → {DNAME, MGRSSN}
- From above dependencies, we can infer
  - {SSN} → {DNAME, MGRSSN}
- Concept of a set of dependencies that can be inferred from the given set
  - Inference rules ?
  - **Closure: F^+ is all dependencies that can be inferred from F**

Definition: Closure of a Set of FD's

Defn. Let F be a set of FD's.
Its *closure*, F^+, is the set of all FD's:
{$X \rightarrow Y$ | $X \rightarrow Y$ is derivable from ***F*** by Armstrong's Axioms}

- *Two sets of dependencies F and G are equivalent if $F^+=G^+$*
  - *i.e., their closures are equal*
  - *i.e., the same sets of FDs can be inferred from each*
- Algorithm to compute closure…read notes
  - Repeated application of the transitive rule

Attribute Set Closure

- Another interesting (important) question is: "Given a set X of attributes, what is the set of attributes X^+ that are functionally dependent on X ?"
  - X^+ is the attribute set closure of X
  - Values in X uniquely determine the values of all attributes in X^+
- Definition of key for relation R: X is a key for R if X^+ is the set of all attributes in R
- Algorithm to determine keys for a relation ?

Computing Attribute Set Closure

- For attribute set X, compute closure X^+ by:

```
Closure X+ := X;
repeat until no change in X+ {
        if there is an FD U → V in F
           such that U is in X+
              then add V to X+}
```

Attribute Set Closure and Keys

- If X is a key over relation scheme R, then what is X^+
  - Formal definition of a Key
- How to determine the keys for relation R ?
  - R is a set of attributes $\{A_1, A_2, \ldots, A_n\}$
  - For each subset S of R, compute S^+
    - If $S^+ = R$ then S is Key
  - What is the "catch" here ?
  - Can you improve this ?
- Some techniques to prune the search space:
  - If an attribute does NOT appear on RHS of any dependency then it must be part of a key
  - If a set S is a key, then all supersets of S are superkeys
  - If S is a key, and an attribute set Q determines S then Q is a key
    - Cycle in a graph

Now we are ready to define normal forms and normalization

Normalization

- ***Normalization*** is a technique for producing relations with desirable properties.
  - Using **concept of functional dependencies**
- Normalization decomposes relations into smaller relations that contain less redundancy. This decomposition requires that no information is lost and reconstruction of the original relations from the smaller relations must be possible.
- Normalization is a bottom-up design technique for producing relations. It pre-dates ER modeling and was developed by Codd in 1972 and extended by others over the years.
  - Normalization can be used after ER modeling or independently.
  - Normalization may be especially useful for databases that have already been designed without using formal techniques.

Normalization Goal

- The goal of normalization is to produce a set of relational schemas $R_1, R_2, \ldots, R_m$ from a set of attributes $A_1, A_2, \ldots, A_n$.
  - Imagine that the attributes are originally all in one big relation R= $\{A_1, A_2, .., A_n\}$ which we will call the ***Universal Relation***.
  - Normalization divides this relation into $R_1, R_2, \ldots, R_m$.

Desirable Relational Schema Properties

- 1) The most basic property is that relations consists of attributes that are logically related.
  - The attributes in a relation should belong to only one entity or relationship.
- 2) ***Lossless-join property*** ensures that the information decomposed across many relations can be reconstructed using natural joins.
- 3) ***Dependency preservation property*** ensures that constraints on the original relation can be maintained by enforcing constraints on the normalized relations.
- 4) **Avoid update anomalies**

Functional Dependencies & Normal Forms

- Normalization requires decomposing a relation into smaller tables
- Normal forms are properties of relations
- We say a relation is in xNF if its attributes satisfy certain properties
  - Properties formally defined using functional dependencies
  - For example, test the relation to see if it is in 3NF
  - If not in 3NF, then change design…how ?
    Decomposition

How to go about designing a good schema ?

- How to create a 3NF or BCNF database schema ? (i.e., a good design) ?
- Ad-hoc approach
  - Create relations intuitively and hope for the best!
- Formal method – procedure Start with single relation with all attributes
  - Systematically decompose relations that are not in the desired normal form
  - Repeat until all tables are in desired normal form
  - Can decomposition create problems if we are not careful ?
    Yes: (i) Spurious tuples and (ii) lost dependencies
- Can we automate the decomposition process…
  Input: Set of attributes and their functional dependencies
  Output: A 'good' schema design

General Thoughts on Good Schemas

We want all attributes in every tuple to be determined only by the tuple's key attributes, i.e. part of a *superkey* (for key X → Y, a superkey is a "non-minimal" X)

What does this say about redundancy?

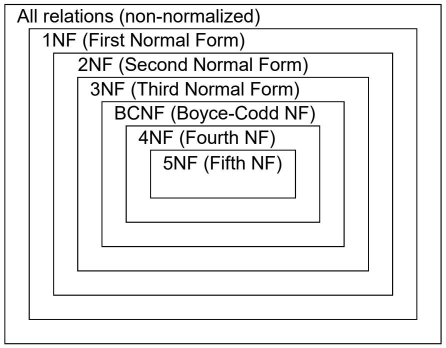
But:

- *What about tuples that don't have keys (other than the entire value)?*

Normal Forms

- A relation is in a particular ***normal form*** if it satisfies certain normalization properties.
- There are several normal forms defined:
  - 1NF - First Normal Form
  - 2NF - Second Normal Form
  - 3NF - Third Normal Form
  - BCNF - Boyce-Codd Normal Form
  - 4NF - Fourth Normal Form
  - 5NF - Fifth Normal Form
- Each of these normal forms are stricter than the next.
  For example, 3NF is better than 2NF because it removes more redundancy/anomalies from the schema than 2NF.
- 3NF and BCNF are relevant to 'real design'…
  - Others are of academic interest

Normal Forms



The two Important Normal Forms

Boyce-Codd Normal Form (BCNF). For every relation scheme R and for every $X \to A$ that holds over R,

either $A \in X$ (it is trivial) ,or

or X is a superkey for R

Third Normal Form (3NF). For every relation scheme R and for every $X \to A$ that holds over R,

either $A \in X$ (it is trivial), or

X is a superkey for R, or

A is a member of some key for R

Definitions

- Relation schema R
  - Superkey
  - Key
  - Candidate key – same as key
  - Primary key – a key designated for common use
  - **Prime attribute** – an attribute that belongs to some candidate key
  - **Non-prime attribute** – does not belong to any key
  - Set of attributes can be partitioned into Prime or Non-prime attributes

Formal Definitions

- We discussed lossless joins…how to define it formally ?
  - Recall: bad decompositions create spurious tuples, and/or we cannot reconstruct the original data

Lossless Join Decomposition

$R_1, \ldots R_k$ is a *lossless join decomposition* of R w.r.t. an FD set *F* if for every instance *r* of R that satisfies *F*,

$$\Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_k}(r) = r$$

sid	name	cid	subj	crnum	exp-grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

Is (sid, name) and (cid, subj, crnum, exp-grade) a lossless join decomposition ?

sid → name
cid → crnum, exp-grade
crnum → subj

sid	name	cid	subj	crnum	exp-grade
1	Sam	570103	SW	cs143	B
23	Dan	550103	DB	cs178	A

Testing for Lossless Join

R_1, R_2 is a lossless join decomposition of R with respect to *F* iff *at least one* of the following dependencies is in *F*+

$$(R_1 \cap R_2) \rightarrow R_1 - R_2$$
$$(R_1 \cap R_2) \rightarrow R_2 - R_1$$

- Set of attributes common to the two tables are key to one of the two table.

So for the FD set:

sid → name
cid → crnum, exp-grade
crnum → subj

Is (sid, name) and (crnum, subj, cid, exp-grade) a lossless decomposition?

Functional dependencies after decompositions

- Definition: Given a set of dependencies F on R, the **projection** of F on R_i, denoted by $p_{R_i}(F)$ where R_i is a subset of R, is the set of dependencies X → Y in F^+ such that the attributes in X υ Y are all contained in R_i.
- Informally: each decomposed relation will have a set of dependencies, such that LHS and RHS from the original set F are both in the relation
  - Let this set be Gi for each decomposed relation Ri
  - The set of relations after decomposition will have G= union of all the sets Gi

Properties of Relational Decompositions (5)

- **Dependency Preservation Property of a Decomposition (cont.):**
  - Dependency Preservation Property:

    A decomposition D = {R1, R2, ..., Rm} of R is **dependency-preserving** with respect to F if the union of the projections of F on each Ri in D is equivalent to F; that is

    $((\pi_{R1}(F)) \cup \ldots \cup (\pi_{Rm}(F)))^+ = F^+$

    (See examples in Fig 14.13a and Fig 14.12)
- Claim 1:
  - It is always possible to find a dependency-preserving decomposition D with respect to F such that each relation R_i in D is in 3nf.

Dependency Preservation

- Ensures we can "easily" check whether a FD $X \rightarrow Y$ is violated during an update to a database:
  - The *projection* of an FD set F onto a set of attributes Z, F_Z is

    $\{X \rightarrow Y \mid X \rightarrow Y \in F^+, X \cup Y \subseteq Z\}$

    i.e., it is those FDs local to Z's attributes
  - A decomposition $R_1, \ldots, R_k$ is *dependency preserving* if $F^+ = (F_{R_1} \cup \ldots \cup F_{R_k})^+$
- Why is this important/desirable ?
- The decomposition hasn't "lost" any essential FD's, so we can check without doing a join

Example of Lossless and Dependency-Preserving Decompositions

Given relation scheme

R(name, street, city, st, zip, item, price)

And FD set name → street, city
street, city → st
street, city → zip
name, item → price

Consider the decomposition

R_1(name, street, city, st, zip) and R_2(name, item, price)

- *Is it lossless?*
- *Is it dependency preserving?*

What if we added FD street, city → item?

Dependency Preservation

- Example:
  - FD set F= C → {everything}, JP → C, SD → P, J → S
  - Is decomposition of CPSJDQ into R1=(SDP), R2= (JS) and R3=(CJDQ)
  - (a) lossless join and (b) dependency preserving
- It is a lossless join decomposition.
  - CPSJDQ = (SDP) ⋈ ((JS) ⋈ (CJDQ))
- But not dependency preserving – since JPC is not in one table
- In this case, adding JPC to the collection of relations gives us a dependency preserving decomposition.

  JPC tuples stored only for checking FD! *(Redundancy!)*

FD's and Keys

- Ideally, we want a design s.t. for each nontrivial dependency $X \rightarrow Y$, X is a superkey for some relation schema in R and all dependencies are preserved
  - We just saw that this isn't always possible
- What if a dependency is lost during decomposition, but we want to enforce the condition ??
  - Is there anything in SQL that can help us enforce this dependency condition ?

    Triggers and Assertions

Now ready to define Normal Forms…

First Normal Form (1NF)

- A relation is in ***first normal form*** (***1NF***) if all its attribute values are atomic.
- That is, a 1NF relation cannot have an attribute value that is:
  - a set of values (multi-valued attribute)
  - a set of tuples (nested relation)
- 1NF is a standard assumption in relational DBMSs.
  - However, object-oriented DBMSs and nested relational DBMSs relax this constraint.
  - NoSQL DBs do not have this assumption…in fact, it is a feature!
- A relation that is not in 1NF is an ***unnormalized*** relation.

A non-1NF Relation

eno	ename	pno	resp	hours
E1	J. Doe	P1	Manager	12
E2	M. Smith	P1	Analyst	24
		P2	Analyst	6
E3	A. Lee	P3	Consultant	10
		P4	Engineer	48
E4	J. Miller	P2	Programmer	18
E5	B. Casey	P2	Manager	24
E6	L. Chu	P4	Manager	48
E7	J. Jones	P3	Engineer	36

Two equivalent representations

eno	ename	pno	resp	hours
E1	J. Doe	P1	Manager	12
E2	M. Smith	{P1,P2}	{Analyst,Analyst}	{24,6}
E3	A. Lee	{P3,P4}	{Consultant,Engineer}	{10,48}
E4	J. Miller	P2	Programmer	18
E5	B. Casey	P2	Manager	24
E6	L. Chu	P4	Manager	48
E7	J. Jones	P3	Engineer	36

Converting non-1NF to 1NF

- Two ways to convert a non-1NF relation to a 1NF relation:
  - 1) ***Splitting Method*** - Divide the existing relation into two relations: non-repeating attributes and repeating attributes.

    Make a relation consisting of the primary key of the original relation and the repeating attributes. Determine a primary key for this new relation.

    Remove the repeating attributes from the original relation.
  - 2) ***Flattening Method*** - Create new tuples for the repeating data combined with the data that does not repeat.

    Introduces redundancy that will be later removed by normalization.

    Determine primary key for this flattened relation.

Converting a non-1NF Relation to 1NF Using Splitting

eno	ename	pno	resp	hours
E1	J. Doe	P1	Manager	12
E2	M. Smith	P1	Analyst	24
		P2	Analyst	6
E3	A. Lee	P3	Consultant	10
		P4	Engineer	48
E4	J. Miller	P2	Programmer	18
E5	B. Casey	P2	Manager	24
E6	L. Chu	P4	Manager	48
E7	J. Jones	P3	Engineer	36

Also need original primary key: eno

Repeating group: (pno, resp, hours)

eno	ename
E1	J. Doe
E2	M. Smith
E3	A. Lee
E4	J. Miller
E5	B. Casey
E6	L. Chu
E7	J. Jones

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

Converting a non-1NF Relation to 1NF Using Flattening

eno	ename	pno	resp	hours
E1	J. Doe	P1	Manager	12
E2	M. Smith	P1	Analyst	24
		P2	Analyst	6
E3	A. Lee	P3	Consultant	10
		P4	Engineer	48
E4	J. Miller	P2	Programmer	18
E5	B. Casey	P2	Manager	24
E6	L. Chu	P4	Manager	48
E7	J. Jones	P3	Engineer	36

eno	ename	pno	resp	hours
E1	J. Doe	P1	Manager	12
E2	M. Smith	P1	Analyst	24
E2	M. Smith	P2	Analyst	6
E3	A. Lee	P3	Consultant	10
E3	A. Lee	P4	Engineer	48
E4	J. Miller	P2	Programmer	18
E5	B. Casey	P2	Manager	24
E6	L. Chu	P4	Manager	48
E7	J. Jones	P3	Engineer	36

Second Normal Form (2NF)

- A relation is in ***second normal form*** (***2NF***) if it is in 1NF and every non-prime attribute is fully functionally dependent on a candidate key.
  - A ***prime attribute*** is an attribute in any candidate key.
- *Alternate definition: there is no partial dependency on the key*
- If there is a FD X → Y that violates 2NF:
  - Compute X^+.
  - Replace R by relations: $R_1 = X^+$ and $R_2 = (R - X^+) \cup X$
- Note:
  - By definition, any relation with a single key attribute is in 2NF.

Partial Dependency

- A FD X → Y is a partial dependency if there exists an attribute A ∈ X such that X – A → Y
  - Y is partially dependent on X
- Second Normal Form: Relation is in 2NF if no non-prime attribute is partially dependent on the primary key.

What problems (if any) caused by partial dependencies

EMP_PROJ(<u>SSN, PNUMBER</u>, HOURS,ENAME, PNAME, PLOCATION)

{SSN,PNUMBER} → HOURS

{SSN, PNUMBER} → ENAME

SSN → ENAME
PNUMBER → PNAME, PLOCATION

Key for above relation is (SSN,PNUMBER)

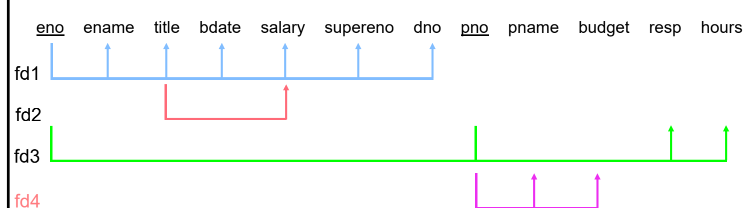
The last two dependencies are partial dependencies since the LHS is part of the key

Problems with partial dependencies

- Insert tuple
  - <987654321, 3, 12,Jones,Sprite,Atlanta>
  - Need to check that 987654321 is Jones
- We have insertion anomaly
  - Check if 987654321 is Jones, project 3 is Sprite…
- We have deletion problem
  - If last tuple with Project #1 is deleted
- Similarly, we have modification anamoly
  - Smith changes name to Brown

Second Normal Form (2NF) Example

- EmpProj relation:

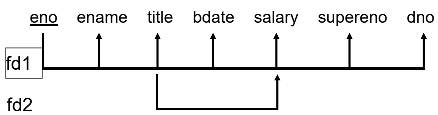


fd1 and fd4 are partial functional dependencies. Normalize to:

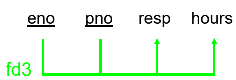
- Emp (<u>eno</u>, ename, title, bdate, salary, supereno, dno)
- WorksOn (<u>eno</u>, <u>pno</u>, resp, hours)
- Proj (<u>pno</u>, pname, budget)

Second Normal Form (2NF) Example

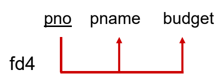
Emp relation:



- WorksOn relation:



Proj relation:



Transitive Dependencies

- FD X $\rightarrow$ Y is a transitive dependency in relation R if there exists set of attributes Z $\in$ R such that
  - X $\rightarrow$ Z and Z $\rightarrow$ Y
  - Z is not a subset of any key of R

Problem with Transitive Dependencies

- EMP_DEPT(ENAME, SSN, BDATE,ADDRESS,
  DNO, DNAME, MGRSSN)

FDs in relation:

{SSN} $\rightarrow$ {DNO}

{DNO} $\rightarrow$ {MGRSSN}

{DNO} $\rightarrow$ {DNAME}

- Insertion, Deletion, Modification anamolies in above schema

- Recall earlier discussion on the same example…..

Third Normal Form (3NF)

- A relation is in ***third normal form*** (***3NF***) if it is in 2NF and there is no non-prime attribute that is transitively dependent on the primary key.

- That is, for all functional dependencies $X \rightarrow Y$ of R, one of the following holds:
  - Y is a prime attribute of R
  - X is a superkey of R

Problem with 3NF?

- ADDR_INFO(CITY, ADDRESS, ZIP)

{CITY, ADDRESS} $\rightarrow$ ZIP

{ZIP} $\rightarrow$ {CITY}

Possible keys: {CITY, ADDRESS} or {ADDRESS,ZIP}

Is it in 3NF?

every attribute is prime, so in 3NF

Problems with the 3NF schema

- Delete <Washington, 800 22nd St, 20052>
- What if this is the last 20052 tuple ?
  - We lose the info that 20052 is in Washington
  - We also have insert, modify anomalies
- Why the problem ?
  - Dependencies from an attribute *to* part of a key
- Solution ?
  - Make all LHS of dependencies be key or superkey!
- **BCNF – Boyce Codd Normal Form**: if all FDs are of the form X $\rightarrow$ Y where X is superkey.

General Definition of 3NF, BCNF

- Can simplify the 3NF definition to remove the reference to partial dependencies/2NF
- R is in 3NF if for every FD X $\rightarrow$ Y, either
  - X is a superkey or
  - Y is a prime attribute
- R is in BCNF if for every FD X $\rightarrow$ Y , X is a superkey
  - R in BCNF $\Rightarrow$ R is in 3NF
- Important: If R is in BCNF then it is in 3NF and 2NF
  - If R is not in 3NF it is not in BCNF

Testing for 3NF, BCNF

- Is the schema in BCNF ?
  - Check if there are non-BCNF dependencies
- Is the schema in 3NF ?
  - Check if there are non-3NF dependencies

    Is there a dependency to non-prime attribute from something that is not a key ?

Decomposition Algorithms

- Algorithms for lossless join dependency preserving 3NF Decomposition
  - Polynomial time algorithm
- Algorithm for lossless join BCNF decomposition
  - No known polynomial time algorithm (NPC)
  - If no of attributes and functional dependencies is a small set, then okay to run an exponential time algorithm

BCNF Decomposition Algorithm

- Input: Relation R (consisting of all attributes), set of functional dependencies F
- Output: BCNF schema *result*
- Algorithm: Initialize *result* := {R}
- while there is a schema R_i in *result* that is not in BCNF

{

let A → B be a FD that violates BCNF in relation R_i

$result := (result - R_i) \cup \{(R_i - B), (A,B)\}$

}

- Note that decomposition $\{(R_i - B), (A,B)\}$ is lossless join
  - Since A is key for (A,B) and A is intersection of these two relations

Example Decomposition into BCNF

- Consider relation R with FDs F. If X →Y violates BCNF, decompose R into R - Y and XY.
  - Repeated application of this idea will give us a collection of relations that are in BCNF; lossless join decomposition, and guaranteed to terminate.
- Example: R=(CSJDPQ)
  - key C (C → everything), JP → C, SD → P, J → S
- Algorithm iteration 1: SD → P is non-BCNF
  - Decompose into R1=(SDP) and R2=(CSJDQV)
- Iteration 2: J → S is not in BCNF in R2=(CSJDQ)
  - Decompose R2 into R3=(JS) and R4=(CJDQ)
- Iteration 3: R1,R3,R4 are in BCNF therefore terminates
- In general, 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.
- Example: decomposition of CSJDQ into SDP, JS and CJDQ is not dependency preserving (we lose the FD JP→C,

  non BCNF FDs SD →P and J → S).
  - However, it is a lossless join decomposition.
  - In this case, adding JP →C to the collection of relations gives us a dependency preserving decomposition.

  JPC tuples stored only for checking FD! *(Redundancy!)*

Normalization Procedure: Summary

- Input= (Set of dependencies F, Set of attributes – single table schema)

1. Use attribute set closure algo to find (a) keys and (b) prime attributes
   - Prune the search using the various "tricks"
2. Test each FD in F to see if it satisfies 3NF/BCNF properties
3. Decompose into smaller relations using decomposition algorithm
4. If BCNF is not dependency preserving, then go with a 3NF decomposition

Testing for 3NF, BCNF

- Is the schema in BCNF ?
  - Check if there are non-BCNF dependencies
- Is the schema in 3NF ?
  - Check if there are non-3NF dependencies

    Is there a dependency to non-prime attribute from something that is not a key ?

Conclusion

- ***Normalization*** is produces relations with desirable properties and reduces redundancy and update anomalies.
- Normal forms indicate when relations satisfy certain properties.
  - 1NF - All attributes are atomic.
  - 2NF - All attributes are fully functionally dependent on a key.
  - 3NF - There are no transitive dependencies in the relation.
  - BCNF – 3NF and all LHS are superkeys.
- In practice, normalization is used to improve schemas produced after ER design and existing relational schemas.
  - Full normalization is not always beneficial as it may increase query time.
  - Problem with relational DBs and large data sets…..NoSQL !!

In-Class Exercises

- 1. R0 = (A,B,F) with dependencies AB $\rightarrow$ F and B $\rightarrow$ F
- 2. R1= (C,T,H,R,S)
  - Course (C), Time (T), Hour (H), Room (R), Section (S), Grade (G)

    C $\rightarrow$ T　　CS $\rightarrow$ G
    HS $\rightarrow$ R　　HR $\rightarrow$ C
    HT $\rightarrow$ R
- (a) What are the keys ?
- (b) Is it in 3NF?
- (c) Is it in BCNF ? If not, then decompose into BCNF
- (d) Is the decomposition dependency preserving ?