Normal Forms		
Learning Objectives	- Understand the rationale (anomalies) and definition of each of the main normal forms based on functional dependencies - Normal forms: 2NF, BCNF, 3NF	
Cases to discount:		

LHS of FDs are proper subsets of candidate key	Example:
	`R = {type, price, name, tel, address}`
	with
	`F = { {type}→{price}, {name}→{tel, address}}`
	where candidate and primary key is
	`{name, type}`
		- Forbid this because closure of `{name, type}` implies everything
		- If either `name` or `type` is removed, everything is not implied
		(since `{name, type}` is a **minimal superkey**)

LHS of FDs are proper subsets of candidate key and is a trivial FD	This FD holds on R: `{type}→{type}`	- The LHS of {type}→{type}, {type}, is a proper subset of the candidate key - This is not a problem since it is a **trivial functional dependency**

LHS of FDs are proper subsets of candidate key but RHS is also a prime attribute	Example:
	`R = {A, B, C, D}`
	with
	`F = {{A, D}→{B, C}, {B}→{A}}`
	where candidate keys are
	`{A, D}` and `{B, D}`
		- LHS of {B}→{A}, {B} is a proper subset of a candidate key but there is nothing we can do about it because {A} is a **prime attribute** (part of a candidate key)
		- Cannot separate because of entanglement (weak entity between 2 keys) {B}→{A}

R is a relation schema, with the set F of FDs For all X: X ⊂ R and for all A ∈ R, there exists a FD: X→{A} in F+		

Second Normal Form (2NF)	R is in 2NF if and only if
	A ∈ X (X→{A}) is **trivial**
	or
	X is **not** a **proper subset** of a candidate keys for R
	or
	A is **part of** some **candidate key** for R
	(A is a prime attribute)
		Example:
		`R = {A, B, C, D}`
		with `F = { {A, B}→{C}, {C}→{D} }`
		and candidate key `{A, B}`
		
		LHS of `{C}→{D}`, `{C}` is not a proper subset of a candidate key
		
		Problem since RHS `{D}` does not depend on candidate key `{A, B}`
		
		`{D}` depends transitively on `{A, B}`

Therefore, to make sure every non-trivial FD corresponds to a candidate/super key:		
Boyce-Codd Normal Form (BCNF)	R is in BCNF if and only if A ∈ X (X→{A}) is **trivial** or X is a **superkey** for R	Example: `R = {A, B, C, D}` with `F = { {A, D}→{B, C}, {B}→{A} }` and candidate keys `{A, D}`, `{B, D}` For `{B}→{A}`, `{B}` is not a superkey and `{A}` is a prime attribute Problem since we cannot untangle the FDs
Third Normal Form (3NF)	R is in 3NF if and only if A ∈ X (X→{A}) is **trivial** or X is a **superkey** for R or A is **part of** some **candidate key** for R	

FD: X→{A} in F+		
*** BCNF ⊂ 3NF ⊂ 2NF ***		
BCNF	**3NF**	**2NF**
Trivial (i.e. {A} is a subset of X)		
X is a **superkey** for R		
	A is part of some candidate key for R (i.e. **prime attribute**)	
		X is **not a proper subset** of a **candidate key** for R
BCNF > 3NF > 2NF	BCNF is the ideal design	

Normalization		
Decomposition	- A decomposition of a relation schema R is a **set of relation scheme** Ri such that ∪i Ri = R, where Ri are called 'fragments' - Unioning all the fragments allows the retrieval of all columns (of original schema) - Ris are **not** partitions	

Lossless Decomposition	- This happens if we can **recover** the **original table**:
	`SELECT *`
	`FROM (R1 NATURAL JOIN R2`
	`NATURAL JOIN R3)`
	- Therefore some **attributes** must be **repeated** in 2 fragments for a meaningful JOIN
	- R3 is typically the table of all the **primary keys** in the other tables
	- To check if lossless:
	Intersection of the tables should be over the (subsets of) primary keys only ***
		Example:
		If a relation R is decomposed into R1 and R2,
		where R1 ∩ R2 = X
		and X → R2
		
		then the decomposition is lossless
		
		The intersection of the 2 tables is only 1 attribute
		
		When we join R1 and R2 on X, for every tuple in R1, there is only one tuple in R2

Lossy Decomposition	- Happens when we cannot recover the original table and **data is lost** on the way	
Dependency Preserving Decomposition	- This happens if the set of FDs on the new scheme is equivalent to the original set of FDs (∪i Fi)+ = F	R with set of FDs F is decomposed into a set of relation schemes Ri **Each Ri inherits a set of FDs, Fi**
	How to check if dependency preserving: 1) Check if original FD in (F1 ∪ F2) 2) If not, check if they are in (F1 ∪ F2)+	
Projected Functional Dependencies	- R with F is decomposed into a set of relation schemes Ri - Each Ri inherits a set of FDs Fi - The **FDs in Fi** are called **projected FDs** (inherited FDs) - Objective: Choose a (minimal) **cover** of the set of dependencies in **F+** that **only** involve the **attributes of Fi** (It may not always work to look at F only)	
Example 1	For some reason, assume we decompose into R1 = {A, B} F1 = {{B}→{A}} where {B} is the primary key R2 = {B, C, D} F2 = {{D}→{B, C}, {C}→{D}} - Here, R1 ∩ R2 = B - The decomposition is lossless -(F1 ∪ F2)+ = F+ means the decomposition is dependency preserving	R = {A, B, C, D} F = { {D}→{B, C}, {C}→{D}, {B}→{A} }

con't) Projected FDs

<table>
<tr><td>Example 2</td><td>We decompose into

R1 = {A, C}
F1 = {{C}→{A}} where {C} is a subset of the primary key

R2 = {B, C, D}
F2 = Ø

- Here, R1 ∩ R2 = B
- The decomposition is lossless
- (F1 ∪ F2)+ ≠ F+ means the decomposition is not dependency preserving (there is a loss of FD)</td><td>R = {A, B, C, D}
F = { {A, B}→{C},
{C}→{A} }</td></tr>
<tr><td>Example 3</td><td>We decompose into

R1 = {A, D}
F1 = {{D}→{A}, {A}→{D}} where {A} is a key

R2 = {A, B, C}
F2 = {{A}→{B}, {B}→{C}, {C}→{A}}

- Here, R1 ∩ R2 = A
- The decomposition is lossless
- Some functional dependencies are in F+ but not in F
- It seems like FDs {C}→{D} and {D}→{A} are lost but they are not
(they can be found in (F1 ∪ F2)+)</td><td>R = {A, B, C, D}
F = { {A}→{B},
{B}→{C},
{C}→{D},
{D}→{A}}</td></tr>
<tr><td>Too much decomposition</td><td colspan="2">- It may be tempting to decompose to the extreme
- Evaluation of queries may be inefficient since it will involve combining several relations
- Joining of tables is an expensive process in SQL</td></tr>
<tr><td>Objectives</td><td colspan="2">- Decompose a table into a lossless, dependency preserving decomposition in BCNF (if possible) or 3NF
- Be able to decompose and synthesise a schema into a lossless, dependency preserving (if possible) BCNF and 3NF decomposition

<table><tr><th>Decomposition in…</th><th>lossless</th><th>dependency preserving</th><th>algorithm</th></tr><tr><td>BCNF</td><td>always</td><td>not always (but most of the time)</td><td>BCNF Decomposition Algorithm</td></tr><tr><td>3NF</td><td>always</td><td>always</td><td>3NF Synthesis Algorithm</td></tr></table></td></tr>
<tr><td>Guidelines for designing a schema</td><td colspan="2">- Normalization theory:
▪ Minimal redundancy, no anomalies
▪ Lossless decompositions
▪ Dependency preserving decompositions
- *** Workload (queries and their effiency requirement)
- Consider the situation and explain if some rules cannot be followed</td></tr>
</table>

"BCNF Decomposition Algorithm from lecture"	- Top-down algorithm (break, break, break) - Different possible orders (in which to consider the constraints violating the BCNF condition) in which we may consider the dependencies violating BCNF in the algorithm application may lead to different decompositions - Breaking is different in each level, breaking into smaller fragments at every level

	`Let S be the initial set of schemes Ri with Fi`
	
	`Until all relation schemes in S are in BCNF`
	`for each Ri in S`
	`if X→Y in F+ violates BCNF for Ri`
	`then let S be`
	`(S - {Ri}) ∪ {X+, (R-X+) ∪ X}`
	`endfor`
	`enduntil`

Example 1	R = {A, B, C, D, E} F = { {A, B}→{C}, {C}→{D}, {D}→{E} } Candidate key is {A, B} (finding the closure will help in identifying candidate keys) {C}→{D} violates BCNF It is not trivial and {C} is not a superkey Therefore R with F is not in BCNF Let us decompose R using {C}→{D} (or can use {D}→{E})
First decide what FD projected on R1, R2 – to decide if R1, R2 in BCNF	R1 = {C}+ = {C, D, E} F1 = { {C}→{D}, {D}→{E} } The candidate key is {C} R1 with F1 is not in BCNF because {D}→{E} is not trivial and {D} is not a superkey In this case, each FD is assigned a table (not always the case – get back?/lost?) R2 = R – {C}+ ∪ {C} = {A, B, C} F2 = { {A, B}→{C} } The candidate key is {A, B} FD corr. to candidate key R2 with F2 is in BCNF We continue to decompose R1 with F1 (split into D and everything D implies (i.e. E))
	We keep R2 which is in BCNF We decompose R1 using {D}→{E} R1.1 = {D}+ = {D, E} F1.1 = { {D}→{E} } The candidate key is {D} R1.1 with F1.1 is in BCNF R1.2 = R1 – {D}+ ∪ + {D} = {C, D} F1.2 = { {C}→{D} } The candidate key is {C} R1.2 with F1.2 is in BCNF
	We are done: R1.1 = {D, E} with F1.1 = { {D}→{E} } is in BCNF R1.2 = {C, D} with F1.2 = { {C}→{D} } is in BCNF R2 = {A, B, C} with F1 = { {A, B}→{C} } is in BCNF We have a lossless, dependency preserving decomposition in BCNF

con’t) BCNF Decomposition Algorithm

Example 2	R = {A, B, C, D} F = { {A, D}→{B, C}, {B}→{A} } The candidate keys are {A, D} and {B, D} (by computing closure) {B}→{A} is not trivial and {B} is not a superkey R with F is not in BCNF Let us decompose it using {B}→{A}
	R1 = {B}+ = {B, A} F1 = { {B}→{A} } Candidate key is {B} R1 with F1 is in BCNF R2 = R – {B}+ ∪ {B} = {B, C, D} F2 = { {B, D}→{C} } **(*** this comes from F+)** The candidate keys are {B, D} We lost FD {A, D}→{B, C} (cannot get it back with current FDs) The decomposition is not dependency preserving
	However, in R and F, {A} is a **prime attribute** R with F is in 3NF (good enough), we do not decompose (since BCNF will lose FD) +: If it is not even in 3NF in the first place, use 3NF Synthesis Algorithm from lecture
“3NF Synthesis Algorithm from lecture” (Bernstein Algo)	Decompose by recomposing/synthesizing 1) Compute minimal cover 2) Compute candidate keys 3) Q: Is it in 3NF/BCNF? 4) Q: Given answers, what are the questions?

	`Let R be a relation scheme;`
	`Let F be a set of functional dependencies;`
	`S = Ø;`
	
	`compute the minimal cover F’`
	
	`for each X→Y in F’`
	`    if no relation in S contains X ∪ Y`
	`      then create relation with scheme X ∪ Y`
	`    if not relation in S contains a candidate key for R`
	`      then create a relation with scheme any candidate key for Ri`
	`end for`

Example 1	R = {A, B, C, D} F = { {A, D}→{B, C}, {B}→{A} } The candidate keys are {A, D} and {B, D} F is an extended minimal cover (otherwise compute it) For table with key: ▪ {A, D}→{B, C} gives R1 = {A, B, C, D} it is in 3NF by construction but not in BCNF ▪ {B}→{A} should give R2 = {A, B} but it is included in R1. We do not create it. it is in 3NF by construction {A, B, C, D} already contains a candidate key The decomposition is dependency preserving by construction

con't) 3NF Synthesis Algorithm

Example 2	R = {A, B, C, D, E} F = { {A, B}→{C}, {C}→{D}, {D}→{E} } The candidate key is {A, B} F is an extended minimal cover (i.e. redundancy removed, otherwise compute it) For table with key: ▪ {A, B}→{C} gives R1 = {A, B, C}, it is in 3NF by construction and also in BCNF ▪ {C}→{D} gives R2 = {C, D} it is in 3NF by construction and also in BCNF ▪ {D}→{E} gives R3 = {D, E} it is in 3NF by construction and also in BCNF {A, B} already contains the candidate key The decomposition is lossless and dependency preserving by construction The decomposition is in BCNF by chance (by odds are very good)
Example 3	R = {A, B, C, D, E} F = { {A}→{B, C}, {D}→{E} } The candidate key is {A, D} F is an extended minimal cover (otherwise compute it) For table with key: ▪ {A}→{B, C} gives R1 = {A, B, C} it is in 3NF by construction and also in BCNF ▪ {D}→{E} gives R2 = {D, E} it is in 3NF by construction and also in BCNF There is no fragment containing the key We create a fragment with a candidate key (for tables to be able to join back) ▪ {A, D} gives R3 = {A, D} it is in 3NF by construction and also in BCNF The decomposition is dependency preserving by construction The decomposition is in BCNF by chance (by odds are very good)
Remarks on both algorithms	- Do the 3NF Synthesis algorithm most of the time (useful and better) - Usually the result will be in BCNF as well - Main problem/difficulty is the projection of FDs

Exercise Questions		
Given	Let us consider the relation R(A, B, C, D, E) with the following set F of FDs: F = { {A, B}→{B, C}, {A, B}→{C, D}, {D}→{D, E}, {C}→{A}, {D, E}→{D}, {B, C}→{E} }	
Question 1	Which of the following functional dependencies are not in F+?	a) {B, C}→{D} b) {A, B, C}→{A, B, C, D, E} c) {E}→{D} d) All of the above (none of them is in F+) e) None of the above (they are all in F+)
Question 2	Which of the following functional dependencies is trivial?	a) {B, C}→{B, C, D} b) {A, B, C}→{A, B, C, D, E} c) {E, D}→{D} d) All of the above e) None of the above
Question 3	Which of the following functional dependencies is completely non-trivial and in F+?	a) {B, C}→{D} b) {A, B, C}→{A, B, C, D, E} c) {E, D}→{D} d) All of the above e) None of the above
Question 4	Which of the following is not a superkey of R with F?	a) {A, B, E} b) {A, B, C} c) {A, D} d) All of the above (none is a superkey) e) None of the above (all are superkeys)
Question 5	Which of the following is a candidate key of R with F?	a) {C, B} b) {D, B} c) {E, C} d) All of the above e) None of the above
Question 6	Which of the following sets of functional dependencies is a minimal cover of F?	a) { {A, B}→{C, D}, {C}→{A}, {D}→{E}. {B, C}→{A} } b) { {A, B}→{C, D}, {C}→{A}, {D}→{E}, {B, C}→{E} } c) { {A, B}→{C, D}, {C}→{A}, {D}→{E} } d) All of the above e) None of the above