Approaches to Database design

There are two approaches to database design

Decomposition is the most common technique where some universal relation containing a set of attributes (ABC,…) is then decomposed (using projections) into subrelations. Regarded by many as the standard method – often based on some normal form and two other criteria: Lossless join decomposition and Dependency preserving decomposition.

Synthesis methods take single attributes and combine them to form larger relations. It is often claimed that these algorithms are more efficient, although they typically lead to 3NF (not BCNF) and also don't mention Lossless join decomposition.

No known algorithm exists which guarantees to generate DPD BCNF schemes.

Lossless Join Decomposition (LJD)
A LJD of a relation scheme R having the set of dependencies F is a projection of R into $\{R_1, R_2, R_3 \ldots R_n\}$ such that each & every instance r of R which satisfies F is equal to the (natural) join of the instances of its projections.

If a decomposed relation is recombined by joining the projections, we would not expect spurious tuples to appear (if they do, this is known as a lossy join).

Example
Show that the decomposition of R=(ABCD) into R1 (BCD) and R2(AC) is not a LJD if r={<abcd>,<efcg>} is an instance of R.

The original R is given as

A	B	C	D
a	b	c	d
e	f	c	g

Which is decomposed into R1

B	C	D
b	c	d
f	c	g

And R2

A	C
a	c
e	c

R1 joined with R2 gives

A	B	C	D	
a	b	c	d	
e	b	c	d	Not in R
a	f	c	g	Not in R
e	f	c	g	

Hence R into R1 and R2 is a lossy join.

Dependency Preserving Decomposition (DPD)
This requires that the set of functional dependencies that hold in some scheme R should also hold in decomposition of R.

Formally: if R is decomposed into $R_1, R_2, \ldots, R_n$ with associated FDs F1, F2, F3,.. Fn then F+ must be equivalent to the closure of F1 u F2 U F3 u … Fn. Any decomposition that satisfies this requirement is called a DPD.

Example
Consider the decomposition of R=(ABCD) with F={A->B, B->C, C->D} into R_1(BCD) and R_2(AC)

R1

B	C	D

R2

A	C

F1 {B->C, C->D}

A->B has been lost so it's not a dependency preserving decomposition.

Note that F+ really needs to be calculated (not just F) which reintroduces the problem of computational expense.

A Decomposition algorithm to Reach BCNF
Assume a relation R with given FD's F.

Suppose relation R has BCNF violation X -> A.
1. Expand right side to include X^+.
2. Decompose R into X^+ and (R - X^+) u X.
3. Find the FD's for the decomposed relations.
   (Project the FD's from F -Calculate all consequents of F that involve only attributes from X+ or only from (R - X^+) u X.

Example of decomposition

R = (A, B, C, D, E) Note AC is Primary Key
With F1={ A -> B, A -> E, C-> D }

Pick BCNF violation A -> B

Expand right side: A -> B, E from A->E

Calculate decomposed relations: X+ is A+ giving (A,B,E)
(R- X+) u X is (ABCDE-ABE) u A giving (A,C,D)

Decomposed relations R1 (A, B, E) R2 (A, C, D)

Projected FD's (skipping a lot of work that leads nowhere interesting):

For R1: A -> B and A -> E.
For R2: C -> D.

Now restart with R1 and R2 - are there BCNF violations?

For R1, A is key and all left sides are superkeys.
For R2, {A, C} is the key, and C -> D violates BCNF.

Decompose R2 Expand right hand side C+={C}
Decompose: R3 (C, D)
R4 (A, C)

Resulting relations are all in BCNF hence we end up with R1 (A, B, E) R3 (C, D) R4 (A, C)

Exercises
One
Use the algorithm to generate a BCNF decomposition for R=(ABCD)
With FDs={A->B, B->C, CD->A, AC->D}

Two
Consider the relation scheme R=(CTHRSG) with FDs {C->T, HR->C, HT->R, CS->G, HS->R}
Use the algorithm to generate a decomposition into BCNF.