Question 1:

(a) R1 = project_{gameid} (select_{category = ‘Exploration’ or category = ‘Adventure} (gamecategories))

R2 = project_{gameid}(select_{category = ‘Exploration} (gamecategories)) intersect project_{gameid} (select_{category = ‘Adventure’} (gamecategories))

R3 = R1 – R2

R4 = project_{gameid, name, designername} (R3 * games * gamedesigners)

Return R4

(b) R1(siteid1, gameid1, isfree1, min_players1, max_players1) = gamesonsite

R2 = (R1 join_{siteid1 <> siteid and gameid1 = gameid} gamesonsite)

R3(siteid2, gameid2, isfree2, min_players2, max_players2) = gamesonsite

R4 = project_{gameid} (select_{siteid2 <> siteid1 and siteid1 <> siteid and siteid2 <> siteid and gameid2 = gameid1 and gameid1 = gameid} (gamesonsite X R2 X R3))

R5 = project_{gameid} (R2) – R4

R6 = project_{gameid, name} (R4 * games)

Return R6

Question2

A	B	C	D	E	F	G	H
a	b1	c	d	e	f1	g1	h1
a	b	c	d2	e2	f2	g2	h2
a	b	c3	d3	e3	f	g	h

Applies (AC -> DE)

A	B	C	D	E	F	G	H
a	b1	c	d	e	f1	g1	h1
a	b	c	d	e	f2	g2	h2
a	b	c3	d3	e3	f	g	h

Applies (AB -> CD)

A	B	C	D	E	F	G	H
a	b1	c	d	e	f1	g1	h1
a	b	c	d	e	f2	g2	h2
a	b	c	d	e3	f	g	h

Applies(AC -> DE)

A	B	C	D	E	F	G	H	
a	b1	c	d	e	f1	g1	h1	
a	b	c	d	e	f2	g2	h2	
a	b	c	d	e	f	g	h	<-- no subscript, lossless decomposition

Question3

AB+ = {ABCDE}
AC+ = {ACDE}
EF+ = {AGEF}

F1 = {AC -> DE}
F2 = {AB -> C}
F3 = {ABF -> G}

F1 union F2 union F3 = {AC -> DE, AB -> C, ABF -> G}
AB+ = {ABCDE}
AC+ = {ACDE}
ABF+ = {ABCFG}

F = {AB -> CD, AC -> DE, EF -> AG}
AB -> CD is true(can be deducted by F1 union F2 union F3)
AC -> DE is true
EF -> AG is not true, so they are not equivalent, which means it is not dependency preserving

Question4

R(A, B, C, D, E, F, G, H)

F = {AB -> CD, AC -> DE, EF -> AG}
Key: ABFH, BEFH
We will take AC -> DE out
AC+ = {ACDE}
R1(A, C, D, E) F1 = {AC -> DE} key: AC, in BCNF
R2(A, B, C, F, G, H) F2 = {AB -> C, ABF -> G} key: ABFH, not in BCNF

Decompose R2:
Take AB -> C out, AB+ = {ABC}
R21(A, B, C) {AB -> C} in BCNF
R22(A, B, F, G, H) {ABF -> G} key: ABFH, not in BCNF

Decompose R22:
Take ABF -> G out, ABF+ = {ABFG}
R221(A, B, F, G) {ABF -> G} in BCNF
R222(A, B, F, H) {} key: ABFH, in BCNF

Final result:
(A, C, D, E)
(A, B, C)
(A, B, F, G)
(A, B, F, H)

Question 5:

**Restaurants(restaurant name, state, street, city, zip, latitude, longitude, url, review id, review text,
cuisine type)**

1. Fds = {restaurant_name state -> url, url -> restaurant_name state,
   state street city zip -> latitude longitude,
   latitude longitude -> state street city zip,
   review_id -> review_text,
   url latitude longitude -> review_id}

First put the set of fds in a basis form using the decomposition rule

F = { restaurant_name state -> url, url -> restaurant_name , url -> state,
state street city zip -> latitude , state street city zip -> longitude,
latitude longitude -> state , latitude longitude -> street, latitude longitude -> city,
latitude longitude -> zip, review_id -> review_text, url latitude longitude -> review_id }

Remove all trivial fds, None

Suppose X->Y is in F and F'=F-{X->Y}, Compute X+ in F and F', if they are the same, then we can remove X->Y. None

Suppose XZ->Y is in F.
Construct, F'= F - {XZ->Y} union {X->Y}
Check if X+ is the same in F and F', if so, then F' becomes F.
None

Use combining rule to return a set of fds

Fds = {restaurant_name state -> url, url -> restaurant_name state,
state street city zip -> latitude longitude,
latitude longitude -> state street city zip,
review_id -> review_text,
url latitude longitude -> review_id} (The original one)

2 keys: {restaurant_name, state, street, city, zip, cuisine type},
{restaurant_name, latitude, longitude, cuisine type},
{url, street, city, zip, cuisine type},
{url, latitude, longtitude, cuisine type}

3 check for BCNF: restaurant_name state -> url, (restaurant_name state) is not super key, not ok for BCNF.

Check for 3NF: for review_id -> review_text, review_id is not super key and review_text is not prime attribute, not ok for 3NF.

4 R1(restaurant_name, state, url) {restaurant_name state -> url, url -> restaurant_name state}
(restaurant_name, state, url) {url -> restaurant_name state} – remove
R2(state, street, city, zip, latitude, longitude) { state street city zip -> latitude longitude, latitude longitude -> state street city zip }

(state, street, city, zip, latitude, longitude) { latitude longitude -> state street city zip} – remove
R3(review_id, review_text) { review_id -> review_text }
R4(url, latitude, longitude, review_id) { url latitude longitude -> review_id }
R5(url, latitude, longtitude, cuisine type) <-- (add one from key)

Final set:
R1(restaurant_name, state, url) {restaurant_name state -> url, url -> restaurant_name state}
Key: (restaurant_name, state), (url)

For (restaurant_name state) -> url, is ok for BCNF since restaurant_name state is super key;
For url -> restaurant_name state, is ok for BCNF since url is super key

R2(state, street, city, zip, latitude, longitude) { state street city zip -> latitude longitude, latitude longitude -> state street city zip }
Key: (state, street, city, zip), (latitude, longitude)

For (state street city zip) -> (latitude longitude), is ok for BCNF since (state street city zip) is super key;
For (latitude longitude) -> (state street city zip), is ok for BCNF since (latitude longitude) is super key.

R3(review_id, review_text) { review_id -> review_text }
Key: review_id

For (review_id -> review_text), is ok for BCNF since review_id is super key

R4(url, latitude, longitude, review_id) { url latitude longitude -> review_id }
Key: url, latitude, longitude

For(url latitude longitude -> review_id), is ok for BCNF since (url latitude longitude) is super key

R5(url, latitude, longtitude, cuisine type) {}
Key: url, latitude, longitude, cuisine type

It is in BCNF since (url, latitude, longitude, cuisine type) is super key

5.
R1, R2, R3, R4, R5 are all in BCNF

R1, R2, R3, R4, R5 are in 4NF since they do not have any multivalued dependency