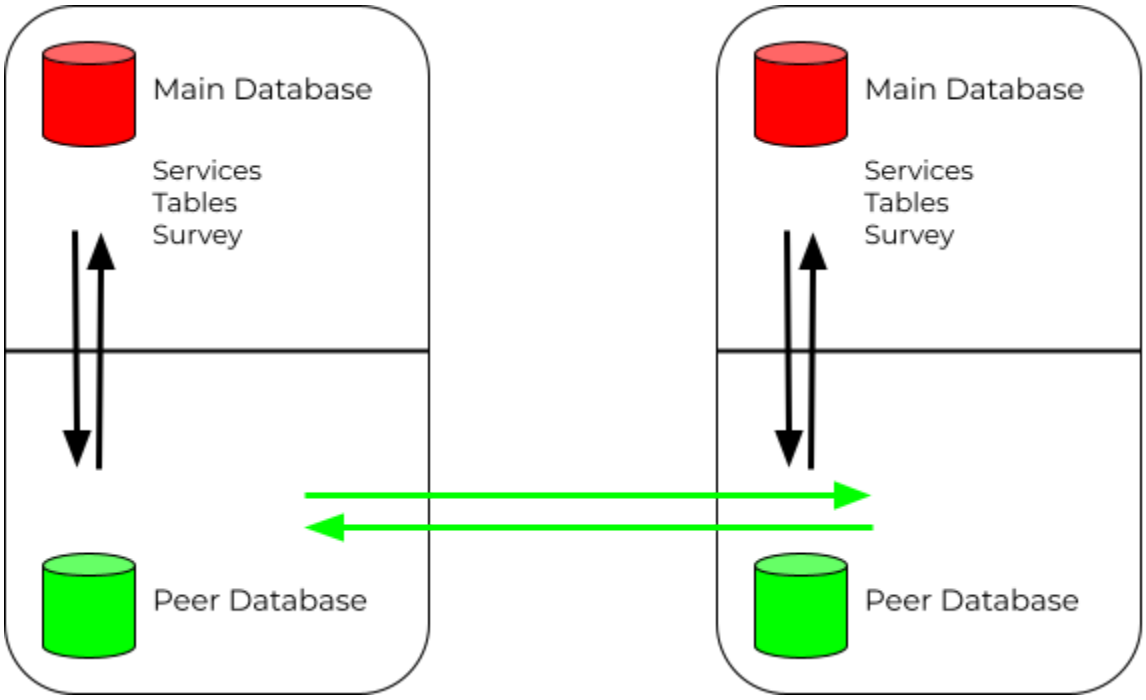
Overview



Glossary

peer db - the protected peer-to-peer replica database used by submit
main db - the mobile database used by Survey/Tables/Services
bulk acceptance - selecting a set of rows that share some common provenance

Transfer Between the peer db and main db (Submit and Services)

Before initiating and after a peer-to-peer synchronization, a synchronization between Services and Submit will be initiated. Similar to a synchronization between a device and a server, if there are checkpoints or conflicts in the main database, the user will not be allowed to move data between the peer-to-peer database and the main database.

Difference between the protected peer-to-peer replica database and the main database
The user is only allowed to modify the main database. The protected peer-to-peer replica database will mostly be transparent to the user.

Services -> Submit
The process should have no conflict. Data is copied from Services to Submit.

Submit -> Services
For the rows in conflict (those in p_conflict or p_divergent) after a peer-to-peer synchronization, the user will need to choose which are those to be available to Services. Those rows that are not selected will be kept as the user might not have sufficient information to decide.

This is where bulk acceptance would happen. The user would be presented with information on where the rows originated and will have the option to accept groups of rows. Bulk acceptance is optionally, this feature is aims to speed up the conflict resolution process. An even more aggressive form of bulk acceptance would be to accept all rows of a peer.

What would be a good way to present the choices to the user?

Transfer Between two devices (Submit on device A and Submit on device B)

When a peer-to-peer sync is initiated, we will synchronize the peer db's on both devices. The peer db is a staging area for the main db, so that we can transfer data between two devices without affecting the main db. Later, when we transfer between the main db and peer db, we will decide what to keep and what to lose. Note: unresolved checkpoints will not be transfered.

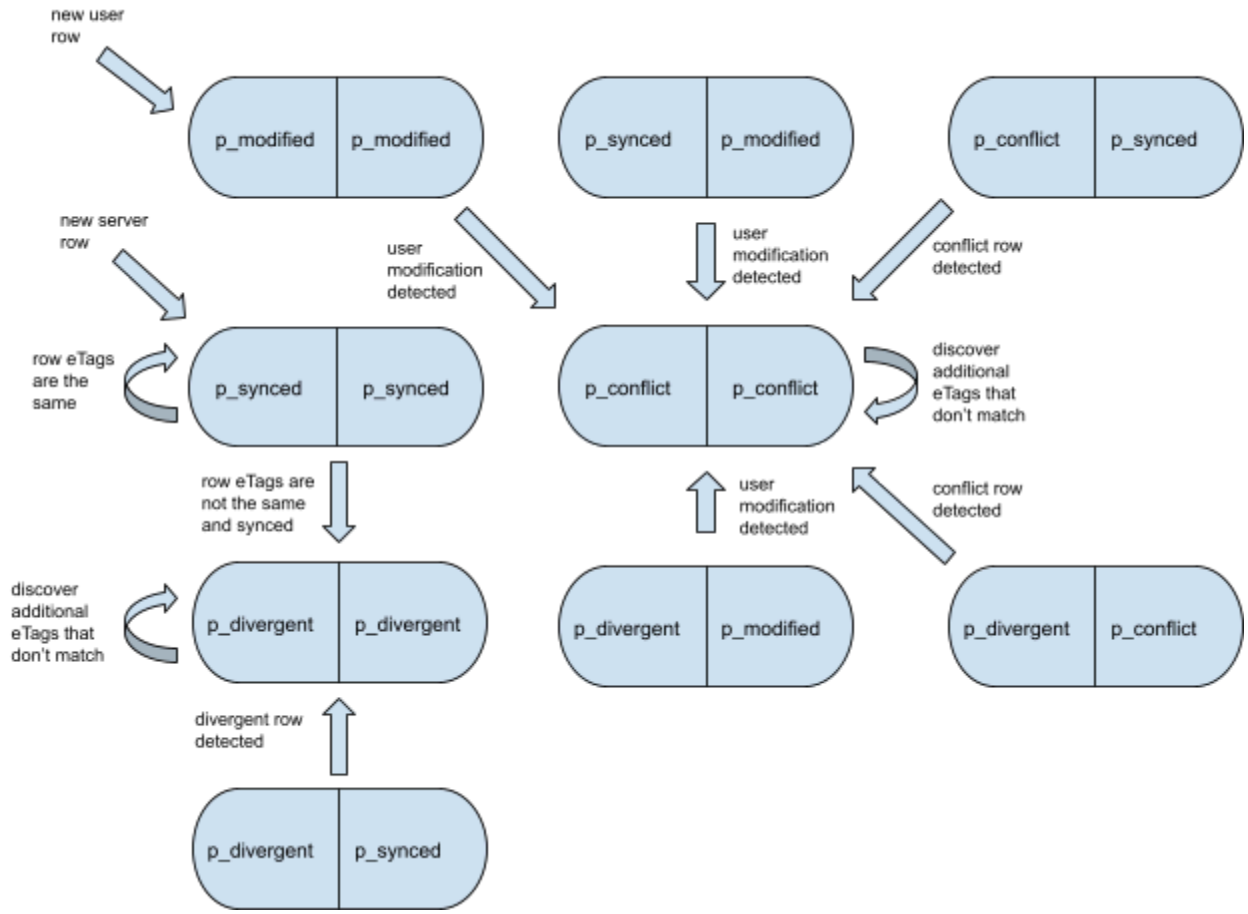
Peer db table schema

- _transfer_id - identifier for a specific sync at some point in time, to be used in bulk acceptance
  The value is populated with an UUID
- _device_id - identifier for device the row originated (was created or originally synced from server)
  This value is populated with an identifier unique to the Android device. When this field needs to be updated, the last value will be used.
- _p_state - peer-to-peer state the row is currently in (see below)

Peer db row states

- p_synced: the row is in total agreement on both devices.
- p_modified: there have been one or more changes to the row as a result of user action (includes new rows). So, it will be up to a user to resolve the differences after a sync.
- p_conflict: the rows are different on both devices as a result of user modification (the rows in main db are in different states).
- p_divergent: the rows are different on both devices, but both rows are in their main sync state. This occurs as a result of changes the server has made to both devices, and can be resolved on next contact with server

Below is a state diagram for a row with the same id on two different devices. The diagram encompasses all state transitions during a transfer between two devices.



Examples

2 peers with distinct rows

Device A

_id	_sync_state	_p_sync_state
1	synced	p_synced

Device B

_id	_sync_state	_p_sync_state
2	synced	p_synced

After a peer-to-peer synchronization

Device A

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
1	synced	p_synced	A	
2	synced	p_synced	B	1

Device B

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
2	synced	p_synced	B	
1	synced	p_synced	A	1

2 peers with the same row (rows in synced state)

Device A

_id	_sync_state	_p_sync_state	_data_etag	value
1	synced	p_synced	10	100

Device B

_id	_sync_state	_p_sync_state	_data_etag	value
1	synced	p_synced	20	200

After a peer-to-peer synchronization

Device A

_id	_sync_state	_p_sync_state	_device_id	_transfer_id	_data_etag	value
1	synced	p_divergent	A		10	100
1	synced	p_divergent	B	1	20	200

Device B

_id	_sync_state	_p_sync_state	_device_id	_transfer_id	_data_etag	value
1	synced	p_divergent	B		20	200
1	synced	p_divergent	A	1	10	100

2 peers with rows created locally

Device A

_id	_sync_state	_p_sync_state	value
1	synced	p_synced	A100

Device B

_id	_sync_state	_p_sync_state	value
1	new_row	p_modified	B100
2	new_row	p_modified	B200

After a peer-to-peer synchronization

Device A

_id	_sync_state	_p_sync_state	_device_id	_transfer_id	value
1	synced	p_conflict	A		A100
1	new_row	p_conflict	B	1	B100
2	new_row	p_modified	B	1	B200

Device B

_id	_sync_state	_p_sync_state	_device_id	_transfer_id	value
1	new_row	p_conflict	B		B100
2	new_row	p_modified	B		B200
1	synced	p_conflict	A	1	A100

Propagation of _device_id and _transfer_id (no conflict)

Device A

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
1	synced	p_synced	A	

Device B

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
2	synced	p_synced	B	

Device C

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
3	synced	p_synced	C	

After a peer-to-peer synchronization between A and B

Device A

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
1	synced	p_synced	A	
2	synced	p_synced	B	1

Device B

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
2	synced	p_synced	B	
1	synced	p_synced	A	1

After a peer-to-peer synchronization between B and C

Device B

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
2	synced	p_synced	B	
1	synced	p_synced	A	1

3	synced	p_synced	C	2

Device C

_id	_sync_state	_p_sync_state	_device_id	_transfer_id
3	synced	p_synced	C	
2	synced	p_synced	B	2
1	synced	p_synced	A	2

Note that even though Device C never peer-to-peer synchronized with Device A, it can still have a row marked with _device_id A.

Possible Future Features

Automatic resolution of p_divergent

Rows are in the p_divergent state when multiple synced copies of a row exist simultaneously. Among the copies, 1 copy is the latest copy but in a disconnected environment it cannot be determined which copy is the latest. When a server is present, because it has authority over the versioning of data it could tell which row is the latest. An extension to that would be to cache enough information during a synchronization between a Sync Endpoint and a device such that there is sufficient information to pick out the latest version of a row. One approach would be to retrieve the count of dataETags from a Sync Endpoint and associate that count with the latest dataETag. Since the count increases monotonically, the count could be used to determine the latest copy of a row.

Automatic resolution of repeated conflicts

Consider 3 devices, A, B and C. They all have 1 row and that row has _id 1. The remaining columns of that row is identical on Device A and B while Device C has conflicting values. Suppose the following interaction:

1. Device A and C peer-to-peer synchronize and produce conflict on both devices.
2. After the synchronization, the conflict is resolved manually on Device C.
3. Device B and C peer-to-peer synchronize and produce conflicts.
4. After the synchronization, the conflict is resolved manually on Device C.

Since Device A and B share the identical set of values, the identical conflict resolution process would need to be performed twice on Device C (step 2 and 4). If there is a way for the peers to retain memory of transitions and conflict resolution decisions, the peer could perform conflict resolutions automatically in some cases. One possible solution is to store the hash of the state of a row together with its previous state. For example, hash = hash_func(prev_hash + state). The previous hash needs to be incorporated to distinguish the directionality of a transition, i.e. X -> Y and Y -> X.