1. Add Group With Participants (Using Transaction)

```
Future<int> addGroupWithParticipants({
  required String groupName,
  required String dateIso,
  required List<Map<String, dynamic>> participants,
}) async {
  final db = await database;

  return await db.transaction((txn) async {
    //  Insert group
    int groupId = await txn.insert('groups', {
      'group_name': groupName,      'date': dateIso,
    });

    //Insert participants
    for (var p in participants) {
      await txn.insert('participants', {
        'group_id': groupId,
        'participant_name': p['name'],
        'image': p['image'],
        'mobile': p['mobile'],
      });
    }

    return groupId;
  });
}
```

2. Get ALL Groups (display list of groups in first screen)

```
Future<List<Map<String, dynamic>>> getAllGroups() async {
  final db = await database;
  return await db.query(
    'groups',
```

```
    orderBy: 'group_id DESC',
  );
}
```

3. Get Group With Participants

```
Future<Map<String, dynamic>> getGroupWithParticipants(int groupId) async {
  final db = await database;

  final group = await db.query(
    'groups',
    where: 'group_id = ?',
    whereArgs: [groupId],
  );

  final participants = await db.query(
    'participants',
    where: 'group_id = ?',
    whereArgs: [groupId],
  );

  return {
    'group': group.first,
    'participants': participants,
  };}
```

4. Update Group With Participants

```
Future<void> updateGroupWithParticipants({
  required int groupId,
  required String groupName,
  required List<Map<String, dynamic>> participants,
}) async {
  final db = await database;

  await db.transaction((txn) async {
    // 1□ Update group
```

```
await txn.update(
  'groups',
  {'group_name': groupName},
  where: 'group_id = ?',
  whereArgs: [groupId],
);

// 2️⃣ Get existing participants from DB
final existingParticipants = await txn.query(
  'participants',
  where: 'group_id = ?',
  whereArgs: [groupId],
);

List<int> existingIds =
    existingParticipants.map((e) => e['participant_id'] as int).toList();

List<int> updatedIds = [];

// 3️⃣ Insert or Update
for (var p in participants) {
  if (p['participant_id'] != null) {
    // UPDATE
    await txn.update(
      'participants',
      {
        'participant_name': p['name'],
        'image': p['image'],
        'mobile': p['mobile'],
      },
      where: 'participant_id = ?',
      whereArgs: [p['participant_id']],
    );

    updatedIds.add(p['participant_id']);
  } else {
    // INSERT NEW
    await txn.insert('participants', {
      'group_id': groupId,
      'participant_name': p['name'],
```

```
          'image': p['image'],
          'mobile': p['mobile'],
        });
      }
    }

    // 4️⃣ Delete removed participants
    for (var id in existingIds) {
      if (!updatedIds.contains(id)) {
        await txn.delete(
          'participants',
          where: 'participant_id = ?',
          whereArgs: [id],
        );
      }
    }
  });
}
```

5. Delete Group With Everything

```
return await openDatabase(
  path,
  version: 1,
  onCreate: _createDB,
  onConfigure: (db) async {
    await db.execute('PRAGMA foreign_keys = ON');
  },
);

Future<void> deleteGroupWithEverything(int groupId) async {
  final db = await database;

  await db.transaction((txn) async {
    await txn.delete(
      'groups',
      where: 'group_id = ?',
      whereArgs: [groupId],
```

```
    );
  });
}
```

6. Fetch Expenses Using groupId

```
Future<List<Map<String, dynamic>>> fetchExpensesByGroupId(int groupId) async {
  final db = await database;

  return await db.query(
    'expenses',
    where: 'group_id = ?',
    whereArgs: [groupId],
    orderBy: 'expense_date DESC',
  );
}
```

7. Fetch Participants By groupId

```
Future<List<Map<String, dynamic>>> fetchParticipantsByGroupId(int
groupId) async {
  final db = await database;

  return await db.query(
    'participants',
    where: 'group_id = ?',
    whereArgs: [groupId],
  );
}
```

8. Fetch Dropdown Participants (id + name only)

```
Future<List<Map<String, dynamic>>>
fetchDropdownParticipantsByGroupId(int groupId) async {
  final db = await database;

  return await db.query(
    'participants',
    columns: ['participant_id', 'participant_name'],
    where: 'group_id = ?',
    whereArgs: [groupId],
  );
}
```

9. Fetch Expense Splits By expenseId

```
Future<List<Map<String, dynamic>>>
fetchExpenseSplitsByExpenseId(int expenseId) async {
  final db = await database;

  return await db.query(
    'expense_splits',
    where: 'expense_id = ?',
    whereArgs: [expenseId],
  );
}
```

10. Insert Expense (Return expenseId)

```
Future<int> insertExpense({
```

```
  required int groupId,
  required String expenseName,
  required double amount,
  required int paidBy,
  required String splitType,
  String? description,
  required String expenseDate,
}) async {
  final db = await database;

  return await db.insert('expenses', {
    'group_id': groupId,
    'expense_name': expenseName,
    'amount': amount,
    'paid_by': paidBy,
    'split_type': splitType,
    'description': description,
    'expense_date': expenseDate,
  });
}
Insert Expense Split
Future<int> insertExpenseSplit({
  required int expenseId,
  required int participantId,
  required double amount,
  required int groupId,
}) async {
  final db = await database;

  return await db.insert('expense_splits', {
    'expense_id': expenseId,
    'participant_id': participantId,
    'amount': amount,
```

```
    'group_id': groupId,
  });
}
```

11. Update Expense

```
Future<int> updateExpense({
  required int expenseId,
  required String expenseName,
  required double amount,
  required int paidBy,
  required String splitType,
  String? description,
  required String expenseDate,
}) async {
  final db = await database;

  return await db.update(
    'expenses',
    {
      'expense_name': expenseName,
      'amount': amount,
      'paid_by': paidBy,
      'split_type': splitType,
      'description': description,
      'expense_date': expenseDate,
    },
    where: 'expense_id = ?',
    whereArgs: [expenseId],
  );
}
```

12. deleteExpense

```
Future<void> deleteExpense(int expenseId) async {
  final db = await database;

  await db.delete(
    'expenses',
    where: 'expense_id = ?',
    whereArgs: [expenseId],
  );}
```

13. updateExpenseSplitByExpenseAndParticipant

```
Future<int> updateExpenseSplitByExpenseAndParticipant({
  required int expenseId,
  required int participantId,
  required double amount,
}) async {
  final db = await database;

  return await db.update(
    'expense_splits',
    {'amount': amount},
    where: 'expense_id = ? AND participant_id = ?',
    whereArgs: [expenseId, participantId],
  );
}
```