SYBASE

SQL 502
Adding MobiLink Synchronization to an Existing Database

Reg Domaratzki (rdomarat@ianywhere.com)
International and Sustaining Engineering Group
iAnywhere Solutions, Waterloo, Ontario, Canada

SYBASE TechWave 2005
USER TRAINING & SOLUTIONS CONFERENCE

Outline

Introduction

Limitations and Assumptions

Challenges

Consolidating Data for Reporting Purposes

Distributing Data to Multiple Locations

Introduction

Purpose

- The goal of this talk is to show how you can add Mobilink synchronization to an existing stand-alone application without making any changes to the schema of the existing database
- You may want to do this so that you can consolidated all the data from your stand alone databases into a single location so that you can perform some reporting on the data
- You may also want to share information between the stand alone versions of your application, so that everyone has access to everyone else's data

Introduction

Target Audience

- Existing Partners
  - You currently have an existing stand alone application that you would like to add synchronization to
  - The time involved in having to re-work your database schema and possibly your application doesn't fit into your existing schedules
- Current End Users
  - You've developed an in-house application using ASA that you would like to synchronize
  - Third party vendor is unwilling to add synchronization capabilities but is willing to let you do it yourself

Outline

Introduction

Limitations and Assumptions

- Runtime engine
- Database version
- DBA or RESOURCE authority required
- Primary Key modification
- Truncation of transaction log
- No DBA at Remote Site
- Cannot Modify Existing Database Objects

Challenges

Consolidating Data for Reporting Purposes

Distributing Data to Multiple Locations

Limitations and Assumptions

Limitations

- Runtime engine
  - The runtime version of the engine cannot use a transaction log, so replication and synchronization are not possible using the freely distributed runtime engine
- Database version
  - ASA was not available as a remote database for MobiLink until version 7.0.0
- DBA or RESOURCE authority required
  - You will need access to a database user with DBA or RESOURCE authority in order to create the objects in the database needed for synchronization

Limitations and Assumptions

Limitations

- Application CANNOT modify primary keys
  - A synchronization system cannot properly handle the modification of primary keys at the remote site
  - If the application modifies primary keys, it should not be used in a synchronizing or replicating environment
- Truncation of transaction log
  - If the application starts the database engine using the –m switch, or if database events or any external event truncates the transaction log, then it cannot be used in a synchronizing or replicating environment

Limitations and Assumptions

Assumptions

- No DBA at Remote Site
  - No DBA should be needed at the remote site to apply any of the changes or run synchronization
  - No process outlined should involve and end-users doing anything more than running a setup program, or possibly double-clicking on an icon that run a batch file
  - Some expertise will be needed to setup the system, particularly if you are merging existing databases into a single consolidated database

Limitations and Assumptions

Assumptions

- Cannot Modify Existing Database Objects
  - Application often has hard coded SQL statements that assume that the column list has not been changed
    - select * from table
    - insert into table values (…)
  - No changes will be made to existing database objects at the remote
  - No triggers will be added to the remote database

Outline

Introduction

Limitations and Assumptions

Challenges

- Deployment
- Primary Key Uniqueness
- Trigger Actions and Cascading Referential Integrity
- Conflict Resolution
- distributed Deletes

Consolidating Data for Reporting Purposes

Distributing Data to Multiple Locations

Challenges

Deployment

- It is unlikely that the dbmlsync executable and the dbtools DLL were deployed when ASA was first installed
  - A few additional files will likely be needed to be added to the ASA install at the remote sites
- A minimum of three SQL commands will need to be executed against the remote database
  - If you already have a process in place for schema upgrades, this should not be difficult
- dbmlsync will need to be run at the remote site
  - It could be set up to run as a service in the background
  - A OS level event could run dbmlsync periodically
  - An icon could be added to the user's desktop

Challenges

Primary Key Uniqueness

- In a distributed environment, you need to create a way to ensure that a primary key you enter at a remote site will not conflict with a primary key entered at a different site
- Common ways of ensuring primary key uniqueness include
  - GLOBAL AUTOINCREMENT defaults on columns
  - Composite primary keys that include a unique database identifier
  - Primary key pools
- It is unlikely that any of the above techniques were used when the stand alone application was developed
- Ensuring primary key uniqueness is the main focus of the remainder of this talk

Challenges

Trigger Actions and Cascading Referential Integrity

- If the database performs actions as a results of operations performed by the end-user, it's important that these actions be reflected in the synchronization process
- This is handled by dbmlsync by using the SendTriggers extended option

Challenges

Conflict Resolution

- As soon as people are given access to the same data, they will invariably modify each other's data
- You need the ability to handle update conflicts that occur using any business rule you choose
- Conflicts in MobiLink are detected and resolved at the consolidated database, so no changes need to be made at the remote to handle conflicts

Challenges

Distributed Deletes

- If your application allows for deletes to occur on parent records, it is possible for the following situation to occur
  - Remote user 1 deletes a parent record and synchronizes
  - Remote user 2 insert a child record that references the parent that was just deleted at another remote
  - When Remote user 2 synchronizes, a foreign key violation will occur at the consolidated database
- This is best handled using logical deletes
  - Deletes that comes from a remote are not really deleted, but simply marked as deleted
  - Although the deletes will still be sent down to all the remotes, no foreign key violation will occur, and an administrator still has the ability to un-delete the row

Consolidating Data for Reporting

Introduction

Limitations and Assumptions

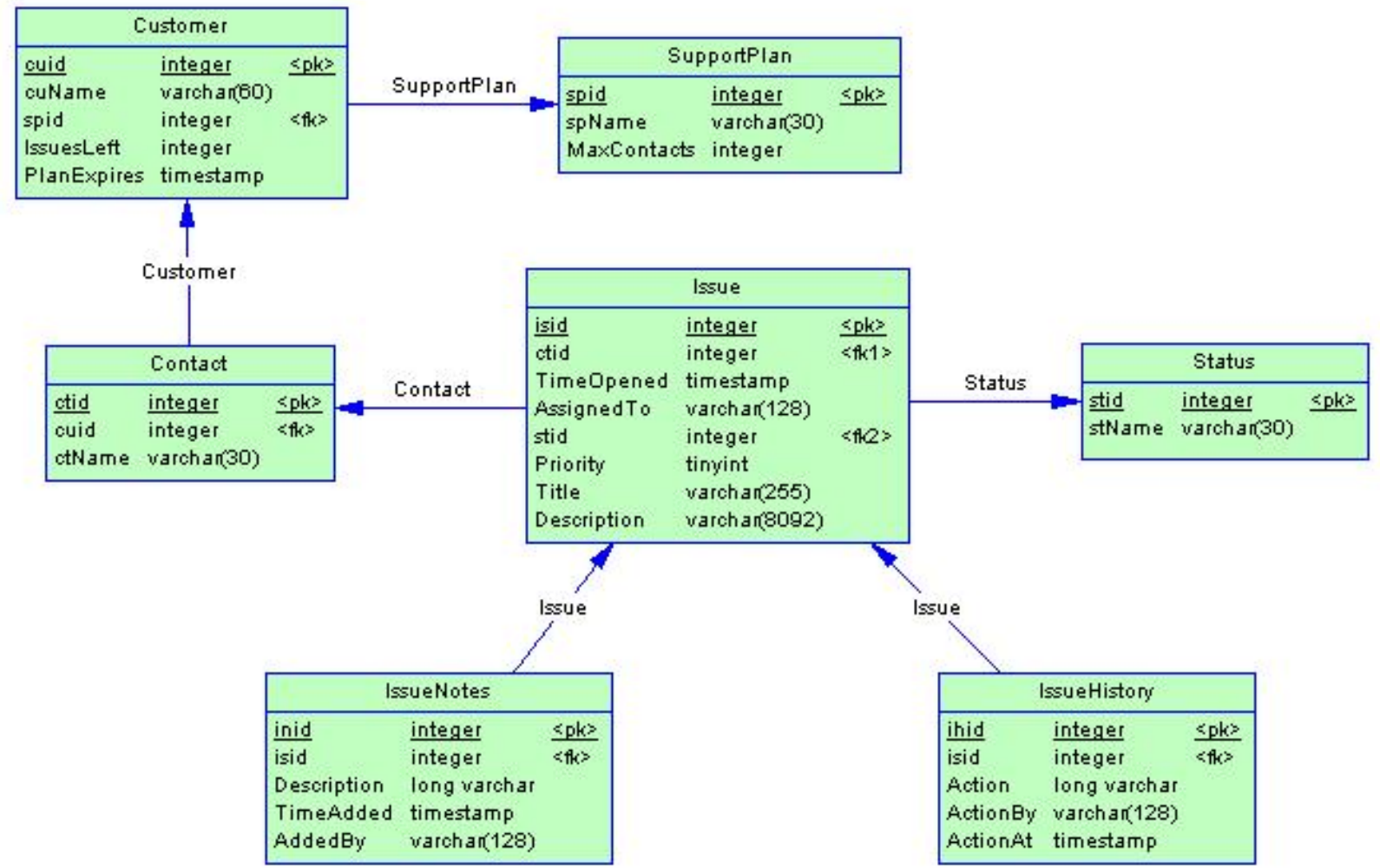
Challenges

Consolidating Data for Reporting Purposes

- Sample Schema
- Solution Overview
- Changes Needed at the Remote
- Ensuring Primary Key Uniqueness at the Consolidated
- Getting the Initial Data to the Consolidated

Distributing Data to Multiple Locations

Consolidating Data for Reporting
Sample Schema



SYBASE®

Consolidating Data for Reporting
Sample Schema

There are things in the schema added to try and make this process more difficult

- All primary keys are DEFAULT AUTOINCREMENT
- Triggers maintain the rows in the IssueHistory table
- Support representatives are defined as database users stored in the SYSUSERPERM table, which cannot be added to a publication
- There is cascading delete referential integrity defined between the Issue table and it's child tables

Consolidating Data for Reporting
Solution Overview

On the Consolidated

- Add an extra column in each table to define which remote the row came from, and make this column part of the primary key of the table
  - Modify the foreign key definitions to include both columns
- Define a begin_synchronization script that sets a global variable to store the name of the remote user that is synchronizing
- Remove trigger definitions
- Remove column and table constraints
- Create a user table to store SYSUSERPERM information from the remotes
- Write upload synchronization scripts for all tables

Consolidating Data for Reporting
Solution Overview

On the remote

- Create a publication that includes all tables
- Create a synchronization user
- Create a synchronization subscription
- Create a user table to duplicate SYS.SYSUSERPERM table

Consolidating Data for Reporting
Changes Needed at the Remote

One of the assumptions made is that there is no DBA at the remote site, so any changes that need to be made MUST be simple

- In order to allow the remote database to synchronize, three SQL commands need to be executed against the database
  - CREATE PUBLICATION
  - CREATE SYNCHRONIZATION USER
  - CREATE SYNCHRONIZATION SUBSCRIPTION
- These command require DBA or RESOURCE authority to execute on the remote database
- For this example, we'll also need to create another user table and procedure to copy the contents of the SYSUSERPERM

Consolidating Data for Reporting
Changes Needed at the Remote

```
create table MySYSUSERPERM (
  user_id unsigned int NOT NULL,
  user_name char(128) NOT NULL,
  password binary(36),
  resourceauth char(1) NOT NULL,
  dbaauth char(1) NOT NULL,
  scheduleauth char(1) NOT NULL,
  publishauth char(1) NOT NULL,
  remotedbaauth char(1) NOT NULL,
  user_group char(1) NOT NULL,
  remarks long varchar,
  PRIMARY KEY ( user_id )
);
```

Consolidating Data for Reporting
Changes Needed at the Remote

```
create procedure sp_hook_dbmlsync_begin ( )
begin
  insert into MySYSUSERPERM
  on existing update
    select *
      from SYSUSERPERM
     where user_id > 100;
  delete
    from MySYSUSERPERM
   where user_id not in
    ( select user_id
        from SYSUSERPERM
       where user_id > 100 );
end;
```

Consolidating Data for Reporting
Changes Needed at the Remote

```
create publication p1 (
  table SupportPlan,
  table Customer,
  table Contact,
  table Status,
  table Issue,
  table IssueNotes,
  table IssueHistory,
  table MySYSUSERPERM
);
```

```
create synchronization user
  DISTINCT_USER_NAME;

create synchronization
  subscription
  to p1
  for DISTINCT_USER_NAME
  type TCPIP
  address 'host=MLServer‘
  options
   SendTriggers='ON',
   LockTables='ON',
   UploadOnly='ON',
   ScriptVersion='sql502_1';
```

Consolidating Data for Reporting
Primary Key Uniqueness at the Consolidated

A composite primary key on the consolidated is the easiest way to ensure primary key uniqueness

The only issue with this is that the MobiLink user name is not passed into the upload_insert, upload_update or update_delete synchronization events, so the value must be stored during the begin_synchronization event so it can be referenced later

Consolidating Data for Reporting
Primary Key Uniqueness at the Consolidated

Remote Table Definition

```
create table SupportPlan (
  spid integer
    default autoincrement,
  spName varchar(30)
     NOT NULL,
  MaxContacts integer
    NOT NULL DEFAULT 1,
  primary key( spid )
);
```

Consolidated Table Definition

```
create table SupportPlan (
  spid integer
    default autoincrement,
  mlUser varchar(128),
  spName varchar(30)
     NOT NULL,
  MaxContacts integer
    NOT NULL DEFAULT 1,
  primary key( spid, mlUser )
);
```

Consolidating Data for Reporting
Primary Key Uniqueness at the Consolidated

Remote Table Definition

```
create table Customer (
  cuid integer default autoincrement,
  cuName varchar(60) NOT NULL,
  spid integer default 1 references SupportPlan,
  IssuesLeft integer NULL,
  PlanExpires timestamp NULL,
  primary key( cuid )
);
```

Consolidating Data for Reporting
Primary Key Uniqueness at the Consolidated

Consolidated Table Definition

```
create table Customer (
  cuid integer default autoincrement,
  mlUser varchar(128),
  cuName varchar(60) NOT NULL,
  spid integer,
  IssuesLeft integer NULL,
  PlanExpires timestamp NULL,
  primary key( cuid, mlUser ),
  foreign key ( spid, mlUser )
    references SupportPlan ( spid, mlUser )
);
```

Consolidating Data for Reporting
Primary Key Uniqueness at the Consolidated

```
call ml_add_connection_script(
  'sql502_1',
  'begin_connection',
  'create variable @mlu varchar(128)'
);

call ml_add_connection_script(
  'sql502_1',
  'begin_synchronization',
  'set @mlu = ?'
);
```

Consolidating Data for Reporting
Primary Key Uniqueness at the Consolidated

```
call ml_add_table_script(
  'sql502_1', 'SupportPlan', 'upload_insert',
  'insert into SupportPlan( spid, mlUser, spName, MaxContacts )
   values ( ?, @mlu, ?, ? )'
);

call ml_add_table_script(
  'sql502_1', 'SupportPlan', 'upload_update',
  'update SupportPlan set spName = ?, MaxContacts = ?
   where spid = ? and mlUser = @mlu'
);

call ml_add_table_script(
  'sql502_1', 'SupportPlan', 'upload_delete',
  'delete from SupportPlan where spid = ? and mlUser = @mlu'
);
```

Consolidating Data for Reporting
Getting the Initial Data to the Consolidated

The data that exists in the remote database at the time that the synchronization subscription is created will never be sent to the consolidated using dbmlsync

- Dbmlsync scans operations from the transaction log, so there is no guarantee that the transaction log will still exist when you add the synchronization subscription
- All changes made to the database AFTER the synchronization subscription is created will be sent to the MobiLink server
- The data that is to be added to the consolidated needs to be the data that existed at the remote at the time the synchronization subscription was created

Consolidating Data for Reporting
Getting the Initial Data to the Consolidated

The following process on the remote will ensure that the proper data is added to the consolidated

1. Stop the remote database
2. Take a copy of the remote database (transaction log not needed)
3. Run the script to create the synchronization subscription
4. Send the copy of the remote database from Step #2 to the consolidated site
5. Restart the remote database
6. Start running dbmlsync
   - Note that dbmlsync will FAIL until the administrators add your MobiLink user into the consolidated database

Consolidating Data for Reporting
Getting the Initial Data to the Consolidated

The following process at the consolidated will ensure that the data is added properly

1. Insert the data from the remote into the consolidated, using the unique MobiLink user name as the value in the mlUser column of the composite primary key
2. Add the MobiLink user so that synchronizations can now begin for this user

When the remote user synchronizes for the first time, all the changes since the synchronization subscription was created will be sent to MobiLink

Consolidating Data for Reporting
Getting the Initial Data to the Consolidated

The data can be added to the consolidated database using proxy tables from the remote

```
create server mlcons class 'asaodbc' using 'cons_dsn';
create existing table cons_SupportPlan at
  ‘mlcons..DBA.SupportPlan';
insert into cons_SupportPlan
  select spid, DISTINCT_USER_NAME, spName, MaxContacts
    from SupportPlan;
drop table cons_SupportPlan;
drop server mlcons;
```

Distributing Data to Multiple Locations

Introduction
Limitations and Assumptions
Challenges
Consolidating Data for Reporting Purposes
Distributing Data to Multiple Locations

- Sample Schema
- Solution Overview
- Changes Needed at the Remote
- Ensure Primary Keys Uniqueness at the Consolidated AND the Remotes
- Writing Synchronization Scripts
- Conflict Resolution
- Getting the Initial Data to the Consolidated
- Adding new MobiLink users

Distributing Data to Multiple Locations
Sample Schema

The schema for the second part of the talk will be a little simpler, because the focus will be on the synchronization scripts in the consolidated to ensure primary key uniqueness across the entire system

There will only be two tables on the remote, with a foreign key relationship between the two tables

Distributing Data to Multiple Locations
Sample Schema

Table Definitions at the Remote

```
CREATE TABLE DBA.Admin (
  admin_id integer NOT NULL DEFAULT autoincrement,
  data varchar(30) NULL ,
  PRIMARY KEY (admin_id)
);

CREATE TABLE DBA.Child (
  child_id integer NOT NULL DEFAULT autoincrement,
  admin_id integer NOT NULL REFERENCES DBA.Admin,
  data varchar(30) NULL,
  PRIMARY KEY (child_id),
)
```

Distributing Data to Multiple Locations
Solution Overview

- Similar to the first sample, an “mlUser” column will be added to each table in the consolidated, and will be included in the primary key of the table
- For each table, we will create another table in the Consolidated database to map a given composite primary key on the consolidated to a non-composite primary key for each remote
- A last modified column will be added to each table in the consolidated so only modified rows are downloaded
- A timestamp column will be added to each table to logically delete rows

Distributing Data to Multiple Locations
Changes Needed at the Remote

Similar to the first sample, only three commands need to be executed at the remote

1. CREATE PUBLICATION
2. CREATE SYNCHRONIZATION USER
3. CREATE SYNCHRONIZATION SUBSRIPTION

Distributing Data to Multiple Locations
Primary Key Uniqueness

The structure of the tables in the consolidated database will be different than the structure on the remote

- A last_modified column will be added to each table so that only changed rows will be downloaded to the remotes
- A delete_time column will be added to each table to track when a row was deleted
  - The row will never be deleted from the consolidated, but we'll use logical deletes to propagate the deletes to other remotes
- The primary key will be changed to a composite primary key that includes the MobiLink user name
  - Foreign keys will now have to reference the composite primary keys on the parent tables as well

Distributing Data to Multiple Locations
Primary Key Uniqueness

Consolidated Table Definition - Admin

```
create table Admin (
  admin_id integer default autoincrement ,
  mlUser varchar(128),
  data varchar(30),
  last_modified timestamp default timestamp,
  delete_time timestamp default NULL,
  primary key ( admin_id, mlUser )
);
```

Distributing Data to Multiple Locations
Primary Key Uniqueness

Consolidated Table Definition – Child

```
create table Child (
  child_id integer default autoincrement ,
  mlUser varchar(128),
  admin_id integer ,
  admin_mlUser varchar(128) ,
  data varchar(30),
  last_modified timestamp default timestamp,
  delete_time timestamp default NULL,
  primary key ( child_id, mlUser ),
  foreign key ( admin_id, admin_mlUser )
    references Admin ( admin_id, mlUser )
);
```

Distributing Data to Multiple Locations
Primary Key Uniqueness

The primary keys on the remotes will still be an integer, but we'll be adding rows from other remote sites that may have the same primary key value

- On the consolidated, the primary key will include the MobiLink user to ensure uniqueness, but we can't add a column to the table on the remote

Another table on the consolidated will map the composite primary key on the consolidated to a distinct primary key value for each remote

- For the remainder of the talk I'll refer to this extra table as the "Pkey" table, and use "base" table to refer to the table that stores the actual data values

Distributing Data to Multiple Locations
Primary Key Uniqueness

```
create table AdminPkey (
  admin_id integer NOT NULL,
  mlUser varchar(128) NOT NULL,
  remoteMlUser varchar(128) NOT NULL,
  remoteAdmin_id integer NULL,
  primary key ( admin_id, mlUser, remoteMlUser ),
  foreign key ( admin_id, mlUser )
    references Admin ( admin_id, mlUser ) on delete cascade
);
```

Distributing Data to Multiple Locations
Primary Key Uniqueness

```
create table ChildPkey (
  child_id integer NOT NULL,
  mlUser varchar(128) NOT NULL,
  remoteMlUser varchar(128) NOT NULL,
  remoteChild_id integer NULL,
  primary key ( child_id, mlUser, remoteMlUser ),
  foreign key ( child_id, mlUser )
    references Child ( child_id, mlUser ) on delete cascade
);
```

Distributing Data to Multiple Locations
Primary Key Uniqueness

cons - AdminPkey

admin_id	mlUser	remoteMlUser	remoteAdmin_id
1	rem1	rem1	1
1	rem1	rem2	2
1	rem2	rem2	1
1	rem2	rem1	2

cons - Admin

admin_id	mlUser	Data
1	rem1	rem1_row1
1	rem2	rem2_row1

rem1 – Admin

admin_id	Data
1	rem1_row1
2	rem2_row1

rem2 - Admin

admin_id	Data
1	rem2_row1
2	rem1_row1

SYBASE®

Distributing Data to Multiple Locations
Primary Key Uniqueness

Populating the Pkey table

- An insert trigger on the base table will manage inserting rows into the Pkey table for that table
- We can only fully populate a single row in the table, since the primary key for the insert on the base table contains the pkey on the remote and MobiLink user name for the remote
- The trigger will also insert a row for every MobiLink user defined in the consolidated, but put in a NULL value for the primary key that will be used at the remote
  - This NULL value will be changed at a later time

Distributing Data to Multiple Locations
Primary Key Uniqueness

```
create trigger ai_Admin after insert on Admin
referencing new as nr for each row
begin

  insert into AdminPkey values ( nr.admin_id, nr.mlUser,
                                 nr.mlUser, nr.admin_id );

  insert into AdminPkey
    select nr.admin_id, nr.mlUser, "name", NULL
      from ml_user
     where "name" not in ( nr.mlUser );

end;
```

Distributing Data to Multiple Locations
Primary Key Uniqueness

Populating the Pkey table

- We can't populate the remoteAdmin_id column for all the remotes when a new row is inserted, because we don't know what new primary key values have been used since the last synchronization at the remote site
- When a user synchronizes, we will need to update all the NULL values in the Pkey tables after the upload is applied, but before the download stream is generated
  - The prepare_for_download event fires at the perfect time
  - Also note that the download is only generated if the upload successfully committed
  - We can guarantee that no new rows are added to the remote database while we generate these new primary keys at the consolidated if the LockTables extended option is set to 'ON'

Distributing Data to Multiple Locations
Getting the Initial Data to the Consolidated

```
create procedure sql502_part2_pfd
  ( @ldt timestamp, @sp_mlu varchar(128) )
begin
  declare cAdmin cursor for
    select admin_id, mlUser from AdminPkey
    where remoteMlUser=@sp_mlu and remoteAdmin_id is NULL;
  declare @cid integer;
  declare @cmlUser varchar(128);
  declare @maxid integer;
  select max(remoteAdmin_id)+1 into @maxid
    from AdminPkey where remoteMlUser=@sp_mlu;
  if( @maxid is NULL ) then set @maxid=1; end if;
  open cAdmin;
  fetch first cAdmin into @cid, @cmlUser;
  while ( sqlcode = 0 ) loop
    update AdminPkey set remoteAdmin_id=@maxid
     where admin_id=@cid and mlUser=@cmlUser and remoteMlUser=@sp_mlu;
    set @maxid=@maxid + 1;
    fetch next cAdmin into @cid, @cmlUser;
  end loop;
  close cAdmin;
end;
```

Distributing Data to Multiple Locations
Primary Key Uniqueness

Populating the Pkey table

- The process described on the Admin table in the preceding slides needs to be applied to the Child table as well
  - There is nothing different (as far as populating the Pkey table) that needs to be done because the table is a child in a foreign key relationship
- The prepare_for_download stored procedure that you write would update all the Pkey tables in the database
  - You could also code this in the begin_download table script for each table, but the fact that a COMMIT occurs after the prepare_for_download event fires means that the work will not have to be done twice if the download transaction is rolled back

Distributing Data to Multiple Locations
Writing Synchronization Scripts

All of the scripts that you write will need to do joins between the base and Pkey tables to figure out how the primary key value that is being passed up from the remote maps to the row as it is stored in the base table on the consolidated

- You'll start by defining a variable in the begin_synchronization event to keep track of the remote user that is synchronizing

```
call ml_add_connection_script(
  'sql502_part2', 'begin_connection',
  'create variable @mlu varchar(128)'
);
call ml_add_connection_script(
  'sql502_part2', 'begin_synchronization', 'set @mlu = ?'
);
```

Distributing Data to Multiple Locations
Writing Synchronization Scripts

The upload_insert event is simple to code

Remember that there is an insert trigger on the table that will be adding rows into the AdminPkey table

```
call ml_add_table_script(
  'sql502_part2', 'Admin', 'upload_insert',
  'insert into Admin values (?,@mlu,?,DEFAULT,DEFAULT)'
);
```

Distributing Data to Multiple Locations
Writing Synchronization Scripts

The upload_insert for the Child table needs to figure out how to map the admin_id value from the remote to an (admin_id, admin_mluser) combination on the consolidated

- Because of the foreign key defined on the remote, we can guarantee that the insert on the parent record was uploaded first

We need to define a stored procedure to do the insert only because the order that we use the parameters in the insert statement does not match the order that is used to pass in the parameters

- We want to use the primary key value last, but it is passed in first

Distributing Data to Multiple Locations
Writing Synchronization Scripts

```
create procedure ui_Child ( in @child_id integer,
                            in @admin_id integer,
                            in @data varchar(30) )
begin
  insert into Child
    ( child_id, mlUser, admin_id, admin_mlUser, data )
  select @child_id, @mlu,
         AdminPkey.admin_id, AdminPkey.mlUser, @data
    from AdminPkey
   where AdminPkey.remoteAdmin_id = @admin_id
     and AdminPkey.remoteMlUser = @mlu;
end;

call ml_add_table_script(
  'sql502_part2', 'Child', 'upload_insert',
  'call ui_Child( ?, ?, ? )'
);
```

Distributing Data to Multiple Locations
Writing Synchronization Scripts

The upload_update event for the Admin table needs to join to the AdminPkey table, and the upload_update event for the Child table needs to join to the ChildPkey and AdminPkey table

```
call ml_add_table_script(
  'sql502_part2', 'Admin', 'upload_update',
  'update Admin
      set Admin.data = ?
     from Admin key join AdminPkey
    where AdminPkey.remoteMlUser =  @mlu
      and AdminPkey.remoteAdmin_id = ?'
);
```

Distributing Data to Multiple Locations
Writing Synchronization Scripts

```
create procedure uu_Child
  ( in @admin_id integer, in @data varchar(30), in @child_id integer )
begin
  declare @cons_admin_id integer;
  declare @cons_admin_mluser varchar(128);

  select admin_id, mlUser into @cons_admin_id, @cons_admin_mluser
    from AdminPkey
   where remoteMlUser = @mlu and remoteAdmin_id = @admin_id;

  update Child
     set admin_id = @cons_admin_id ,
         admin_mlUser = @cons_admin_mluser , data = @data
    from Child key join ChildPkey
   where ChildPkey.remoteChild_id = @child_id
     and ChildPkey.remoteMlUser = @mlu;
end;

call ml_add_table_script(
  'sql502_part2', 'Child', 'upload_update',
  'call uu_Child ( ?, ?, ? )'
);
```

SYBASE®

Distributing Data to Multiple Locations
Writing Synchronization Scripts

When a delete is uploaded to consolidated database, we are not going to delete the row, but instead will logically delete the row by modifying the delete_time column from a NULL value (which indicates that the row has not been deleted) to the time that the row was deleted

- Note that by using logical deletes, you are assuming that no other application is connecting to your database and executing delete statements
- The paranoid amongst the crowd should consider adding a delete trigger that raises an error if a delete if ever performed on a base table

Distributing Data to Multiple Locations
Writing Synchronization Scripts

```
call ml_add_table_script(
  'sql502_part2', 'Admin', 'upload_delete',
  'update Admin
      set delete_time = CURRENT TIMESTAMP
    from Admin key join AdminPkey
   where AdminPkey.remoteMlUser =  @mlu
     and AdminPkey.remoteAdmin_id = ?'
);
call ml_add_table_script(
  'sql502_part2', 'Child', 'upload_delete',
  'update Child
      set delete_time = CURRENT TIMESTAMP
    from Child key join ChildPkey
   where ChildPkey.remoteMlUser =  @mlu
     and ChildPkey.remoteChild_id = ?'
);
```

SYBASE®

Distributing Data to Multiple Locations
Writing Synchronization Scripts

The download_cursor for the Admin table will need to join the Admin table and AdminPkey table to ensure that the proper primary key values are passed down to the remotes

- We will also need to remove rows from the result set that have been logically deleted

The download_cursor for the Child table will need to do a three way join between Child, ChildPkey and AdminPkey

- We will also need to remove rows from the result set that have been logically deleted

Distributing Data to Multiple Locations
Writing Synchronization Scripts

```
call ml_add_table_script(
  'sql502_part2', 'Admin', 'download_cursor',
  'select AdminPkey.remoteAdmin_id, Admin.data
     from Admin key join AdminPkey
    where Admin.last_modified >= ?
      and AdminPkey.remoteMlUser = ?
      and Admin.delete_time is NULL'
);

call ml_add_table_script(
  'sql502_part2', 'Child', 'download_cursor',
  'select ChildPkey.remoteChild_id, AdminPkey.remoteAdmin_id,
          Child.data
     from Child key join ChildPkey, AdminPkey
    where Child.last_modified >= ?
      and ChildPkey.remoteMlUser = @mlu
      and Child.admin_id = AdminPkey.admin_id
      and Child.admin_mluser = AdminPkey.mlUser
      and AdminPkey.remoteMlUser = @mlu
      and Child.delete_time is NULL'
);
```

Writing Synchronization Scripts

- Most of the work for tracking deletes has been done in the upload_delete event, when we set the delete_time column in the table to the time that the row was logically deleted
- The download_delete_cursors for tables need only join the base and Pkey tables and download the primary key values for that remote for rows that have been logically deleted since the last synchronization

Distributing Data to Multiple Locations
Writing Synchronization Scripts

```
call ml_add_table_script(
  'sql502_part2', 'Admin', 'download_delete_cursor',
  'select AdminPkey.remoteAdmin_id
     from Admin key join AdminPkey
    where Admin.delete_time >= ?
      and AdminPkey.remoteMlUser = ?'
);

call ml_add_table_script(
  'sql502_part2', 'Child', 'download_delete_cursor',
  'select ChildPkey.remoteChild_id
     from Child key join ChildPkey
    where Child.delete_time >= ?
      and ChildPkey.remoteMlUser = ?'
);
```

Distributing Data to Multiple Locations
Conflict Resolution

Once you start sharing data, you'll might need to write conflict resolution scripts if the default of "last one in wins" is not suited to your business needs

The conflict resolution is slightly trickier since we'll need to map the primary key values from the remote to the actual values on the consolidated to determine if a conflict has occurred

First, we'll create global temporary tables to store the before, after and current values of the row, *as they exist on the consolidated database*

Distributing Data to Multiple Locations
Conflict Resolution

```
create global temporary table AdminConflict (
  admin_id integer,
  mlUser varchar(128),
  state varchar(1) check ( @col in ( 'o', 'n', 'c' ) ),
  data varchar(30),
  primary key ( admin_id, mlUser, state )
);

create global temporary table ChildConflict (
  child_id integer,
  mlUser varchar(128),
  state varchar(1) check ( @col in ( 'o', 'n', 'c' ) ),
  admin_id integer,
  admin_mluser varchar(128),
  data varchar(30),
  primary key ( child_id, mlUser, state )
);
```

Distributing Data to Multiple Locations
Conflict Resolution

The upload_fetch event will need to gather what the consolidated database believes that row on the remote looks like right now to compare with the old row values that are passed up in the upload stream

A two table join is needed for the Admin table, and a three table join is needed for the Child Table

Distributing Data to Multiple Locations
Conflict Resolution

```
call ml_add_table_script(
  'sql502_part2', 'Admin', 'upload_fetch',
  'select AdminPkey.remoteAdmin_id, Admin.data
     from Admin key join AdminPkey
    where AdminPkey.remoteMlUser =  @mlu
      and AdminPkey.remoteAdmin_id = ?'
);

call ml_add_table_script(
  'sql502_part2', 'Child', 'upload_fetch',
  'select ChildPkey.remoteChild_id,
          AdminPkey.remoteAdmin_id, Child.data
     from Child key join ChildPkey, AdminPkey
    where ChildPkey.remoteMlUser = @mlu
      and ChildPkey.remoteChild_id = ?
      and Child.admin_id = AdminPkey.admin_id
      and Child.admin_mlUser = AdminPkey.mluser
      and AdminPkey.remoteMlUser = @mlu'
);
```

Distributing Data to Multiple Locations
Conflict Resolution

The upload_new_row_insert and upload_old_row_insert table events will be used to populate the global temporary table with the old, new and current values of the data as it exists in the consolidated database

- Stored procedures are used again only because the order that the parameters are passed in does not match the order in which we are using them
- The resolve_conflict stored procedures in this sample ensure that the remote user that initially created the data will win all conflicts, and “first one in” wins in all other situations
- The conflict resolution code for the Child table is very similar
  - The upload_fetch event is provided, but the remaining scripts are left as an exercise to the reader, but can be found in the attached samples

Distributing Data to Multiple Locations
Conflict Resolution

```
create procedure uori_admin
  ( in @admin_id integer, in @data varchar(30) )
begin
   insert into AdminConflict
   select AdminPkey.admin_id, AdminPkey.mlUser, 'o', @data
     from Admin key join AdminPkey
    where AdminPkey.remoteMlUser =  @mlu
      and AdminPkey.remoteAdmin_id = @admin_id;
end;

call ml_add_table_script(
  'sql502_part2', 'Admin', 'upload_old_row_insert',
  'call uori_admin ( ?, ? )'
);
```

Distributing Data to Multiple Locations
Conflict Resolution

```
create procedure unri_admin
  ( in @admin_id integer, in @data varchar(30) )
begin
   insert into AdminConflict
   select AdminPkey.admin_id, AdminPkey.mlUser, 'n', @data
     from Admin key join AdminPkey
    where AdminPkey.remoteMlUser =  @mlu
      and AdminPkey.remoteAdmin_id = @admin_id;
   insert into AdminConflict
   select AdminPkey.admin_id, AdminPkey.mlUser, 'c', Admin.data
     from Admin key join AdminPkey
    where AdminPkey.remoteMlUser =  @mlu
      and AdminPkey.remoteAdmin_id = @admin_id;
end;

call ml_add_table_script(
  'sql502_part2', 'Admin', 'upload_new_row_insert',
  'call unri_admin( ?, ? )'
);
```

SYBASE®

Distributing Data to Multiple Locations
Conflict Resolution

```
create procedure resolve_conflict_admin ()
begin
  declare c1 cursor for select admin_id, mlUser from AdminConflict group by admin_id, mlUser;
  declare @cAdmin_id integer;
  declare @cMlUser varchar(128);
  declare @OldData varchar(30);
  declare @NewData varchar(30);
  declare @CurrentData varchar(30);

  open c1;
  fetch first c1 into @cAdmin_id, @cMlUser;
  while ( sqlcode = 0 ) loop
    select data into @OldData from AdminConflict
     where admin_id = @cAdmin_id and mlUser = @cMlUser and state = 'o';
    select data into @NewData from AdminConflict
     where admin_id = @cAdmin_id and mlUser = @cMlUser and state = 'n';
    select data into @CurrentData from AdminConflict
     where admin_id = @cAdmin_id and mlUser = @cMlUser and state = 'c';
    -- Make sure "real" owner wins conflict
    if( @cMlUser = @mlu ) then
      -- Current User is the "owner", they should win
      update Admin set data = @NewData where admin_id = @cAdmin_id and mlUser = @cMlUser;
    else
      -- do nothing, first one in wins.
    end if;
    fetch next c1 into @cAdmin_id, @cMlUser;
  end loop;
  close c1;
end;
```

Distributing Data to Multiple Locations
Conflict Resolution

```
call ml_add_table_script(
  'sql502_part2', 'Child', 'upload_fetch',
  'select ChildPkey.remoteChild_id,
          AdminPkey.remoteAdmin_id, Child.data
     from Child key join ChildPkey, AdminPkey
    where ChildPkey.remoteMlUser = @mlu
      and ChildPkey.remoteChild_id = ?
      and Child.admin_id = AdminPkey.admin_id
      and Child.admin_mlUser = AdminPkey.mluser
      and AdminPkey.remoteMlUser = @mlu'
);
```

Distributing Data to Multiple Locations
Getting the Initial Data to the Consolidated

Getting the initial data to the consolidated is identical to the process that was followed in the first section

- As long as the inserts on the base table you use through proxy tables populate the mlUser column, the insert triggers on the base table will ensure that the Pkey table is populated

Distributing Data to Multiple Locations
Adding New MobiLink Users

When a new MobiLink user is added to the system, this will need to trigger inserts into the Pkey tables so that the first time the user synchronizes, we can download the rows that were created at the other remote sites

- A trigger on the ml_user table can be used to take care of this

When a MobiLink user is removed from the system, you could choose to also delete (logically) the rows owned by this user

Distributing Data to Multiple Locations
Adding New MobiLink Users

```
create trigger ai_ml_user after insert on dbo.ml_user
referencing new as nr for each row
begin
  insert into DBA.AdminPkey
    select admin_id, mlUser, nr."name", NULL
      from DBA.Admin;
  insert into DBA.ChildPkey
    select child_id, mlUser, nr."name", NULL
      from DBA.Child;
end;
```

iAnywhere at TechWave 2005

MobiLink Usability Testing

- Be the first to check out the new “MobiLink Administration Tool” which guides administrators through all aspects of the data synchronization process using a simple set of wizards and graphical tools.
- Your feedback will provide valuable guidance in the overall direction of this tool.
- Plus a special gift for all testers!
- Located in the Experts Area – Exhibit Hall

iAnywhere at TechWave 2005

Ask the iAnywhere Experts on the Technology Boardwalk (exhibit hall)

- Drop in during exhibit hall hours and have all your questions answered by our technical experts!
- Appointments outside of exhibit hall hours are also available to speak one-on-one with our Senior Engineers. Ask questions or get your yearly technical review – ask us for details!

TechWave ToGo Channel

- TechWave ToGo, an AvantGo channel providing up-to-date information about TechWave classes, events, maps and more –now available via your handheld device!
- www.ianywhere.com/techwavetogo

iAnywhere Developer Community - A one-stop source for technical information!

Access to newsgroups,new betas and code samples

- Monthly technical newsletters
- Technical whitepapers,tips and online product documentation
- Current webcast,class,conference and seminar listings
- Excellent resources for commonly asked questions
- All available express bug fixes and patches
- Network with thousands of industry experts

http://www.ianywhere.com/developer/

SYBASE®

SQL Anywhere 'Jasper' Release

Learn more about **'Jasper', the upcoming SQL Anywhere release**, loaded with features focused on:

- Enhanced data management including performance, data protection, and developer productivity
- Innovative data movement including manageability, flexibility and performance, and messaging

Attend the following sessions:

SQL Anywhere 'Jasper' New Feature Overview
Session SQL512 will be held **Monday, August 22nd, 1:30pm**

MobiLink 'Jasper' New Feature Overview
Session SQL515 will be held **Wednesday, August 24th, 1:30pm**

... **and** remember to look for sneak peeks in other sessions and morning education courses!

Register for the Jasper Beta program:
www.ianywhere.com/jasper

SYBASE®