**Autonomous Interdependent Repositories (AIR)**

Artem Shamsutdinov

Engineering, API Consulting LLC, USA

**Keywords:** Federated multi-repository database system

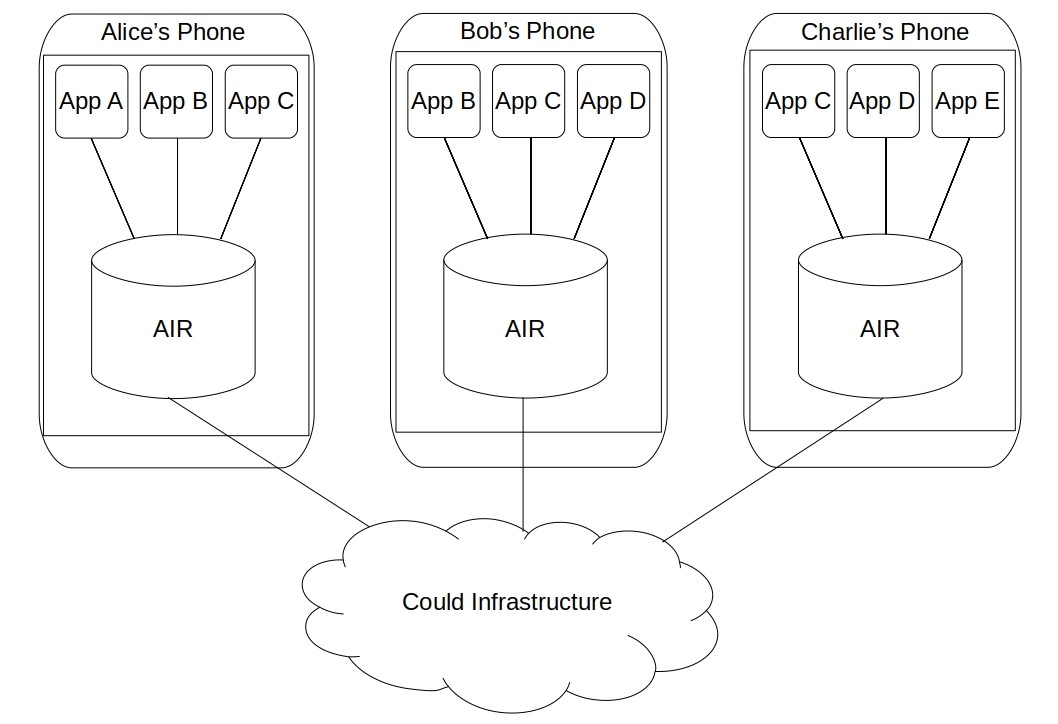
**Abstract.** AIR splits the data in a relational database into repositories. Each repository spans multiple tables. Each table contains records from any number of repositories. A repository is autonomous and represents a stand-alone unit of knowledge. Repositories are interdependent and can reference other repositories. A repository referencing other repositories can be used on its own (without others being present). A repository is stored in a ledger (transaction log) that is distributed between member devices (PC, mobile, IoT). It is secured and authenticated using standard data security mechanisms. A repository has a unique transaction order on each device. An overall order is computed for a repository as all member devices synchronize. An AIR application controls access to its database schema by other applications. CRUD (create/read/update/delete) access is determined on per-table basis. A user controls repository access by applications. She also determines access to her repositories by other users. Each device can contain only the repositories to which its user has access to. Globally repositories form an interconnected database. A device stores AIR data in a database shared by all Apps on that device.

## OVERVIEW

Autonomous Interdependent Repositories (AIR) is a federated data sharing system that forms a (globally) distributed multi-repository database. It allows any number of users to share data across any number of web sites and mobile or desktop apps. Websites and apps can come from any number of publishers. It is an “off-line first” system that allows users to have all access to all of the data they need without a network connection. It is a “privacy first” system and gives users and applications access to only the data they are allowed to access. It is a “security first” system and ensures data confidentiality and authenticity. It is a redundant system where data is stored on multiple devices and can archived to multiple locations. It is a multi-channel system where data can be shared using multiple technologies and channels which can be switched. It is an inconsistency tolerant system where data is guaranteed to be “eventually consistent”. It is a “time traveling” system where the state of its data can be restored to any point in time and space and history of all records is readily available.

### General Sharing

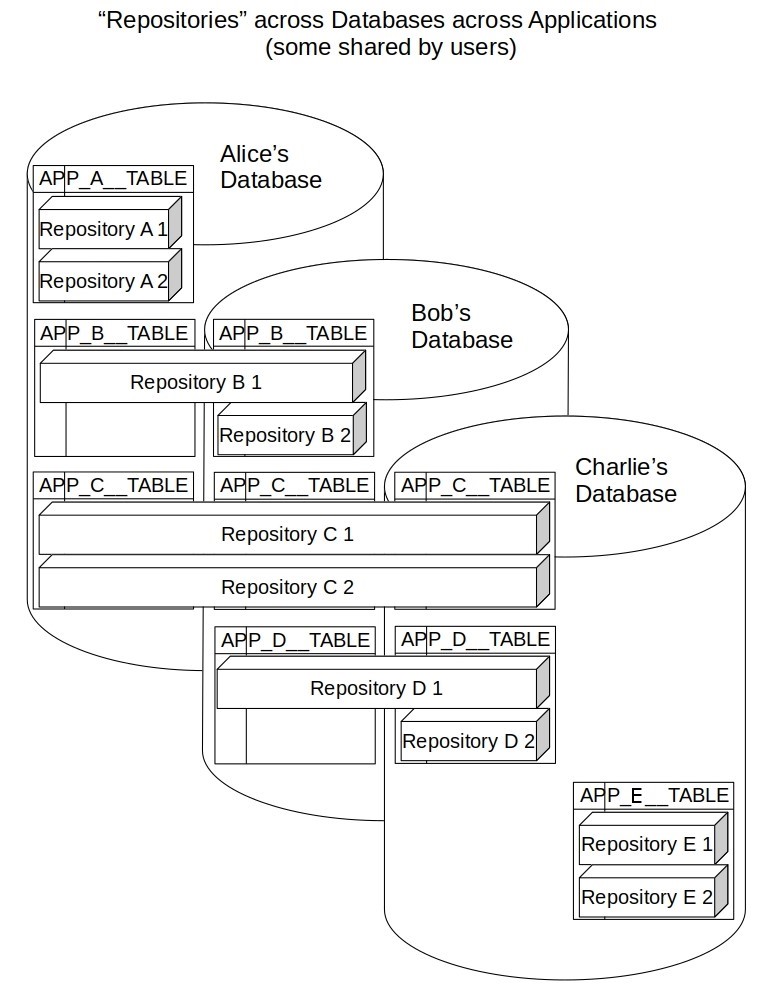
AIR data comes from multiple web sites and mobile/desktop apps and can be used on any type of device, from IOT embedded devices to large scale central databases. Any software can communicate with AIR, given proper access techniques. It is usually backed up in the cloud or network storage of individual users. No central processing server is needed but one can be used for performance reasons.



### Repositories

Data is arranged in “repositories”, which are the building blocks of the system. They are virtual collections of related records that are physically stored in a relational database.

Each database only receives repositories that its user is a part of. Hence each user has all of the data they need and none of the data they don’t need. Composition of data in each AIR database is unique to its user and to the selection of repositories the user decides to store in that database.



#### Repository Granularity

Repositories are meant to be fine grained and time bound. Such “micro-repository” would usually encompasses one (optionally shared) time bound unit of knowledge. Fine granularity helps to:

* Limit the users of a repository to only the ones that need to be its members
* Improve sharing/synchronization performance across AIR databases
* Ease archiving of data to and from the cloud

Alternatively, repositories can be maintained over a long period of time or contain large amounts of data. AIR does not enforce any constraints on the time bounds for repositories or the amount of data that can be stored in them. Such decisions are left up to the application developers and users of those applications.

#### Repository Grouping

Repositories are meant to be grouped via labels and hierarchies, which can themselves be stored in other Metadata Repositories. A Metadata Repository is a repository that stores information about other repositories. Each repository also contains internal metadata about its own data.

## Uses

AIR could be used for any purpose that requires secure data sharing between multiple users. Also, it provides value whenever a user has to work off-line for any period of time, since it has all of the data required by the user on the device they are using. AIR is beneficial when used from a single application but gains most value when used across multiple applications (that are related to each other in various ways).

AIR could be used in any environment where inspection of the order of operations (on per user/per group basis) is important. It could also be used in any environment where there is a need to travel back and forth in the state of the data (across time and member database composition).

AIR provides data redundancy and data authenticity. It tolerates data inconsistency and encourages different data compositions. It also provides automated conflict detection and configurable conflict resolution.

### User-centric Data

The key advantage AIR is that puts user data in the hands of users and flips information problem on its head. AIR applications and websites don’t have own databases and do not store user information – users do. A common setup would be for user to have information they need on the daily basis stored in an AIR database on their mobile device and have all of their data stored in their cloud account(s).

### Synergy across Apps

A key benefit of letting the user manage their information is that now multiple websites and mobile applications can access the same data. For example, user may already have data in a particular field of inquiry (entered though one app) that can be reused by multiple applications.

It is beneficial for an application (that originally provided the schema for that data) to open up its schema to other applications. This is true because other applications can enrich the functionality of the original application in ways that the publisher of the original application could never consider. Multiple applications are meant to co-exist in symbiosis benefiting from each other’s functionality and getting more interaction from the user because of the improvements to their combined feature set.

This is also beneficial to the user, since they now can gain additional value from the data they already have entered and a more comprehensive set of features.

The publisher of the original data schema can monitor what applications are using their schema objects and warn them or blacklist them from accessing that schema (and hence prevent them from accessing the accumulated data that is stored in it).

Since more utility is realized when more valid data is available for processing, AIR compounds the benefit of accumulating data across many apps. Often the utility of the app is determined by how much data it has. A new application using AIR can immediately gain access to a wealth of information already entered by its users and realize its full potential from the start.

### Fine Grained Access Control

Another key benefit of user managed information is that now the user is in charge of how much data they give out and to whom. The users can utilize any number of mechanisms (application reviews for example) to determine which applications are allowed to access which data. This doesn’t prevent applications from being proactive about which data they are asking for.

### Value in Interdependence

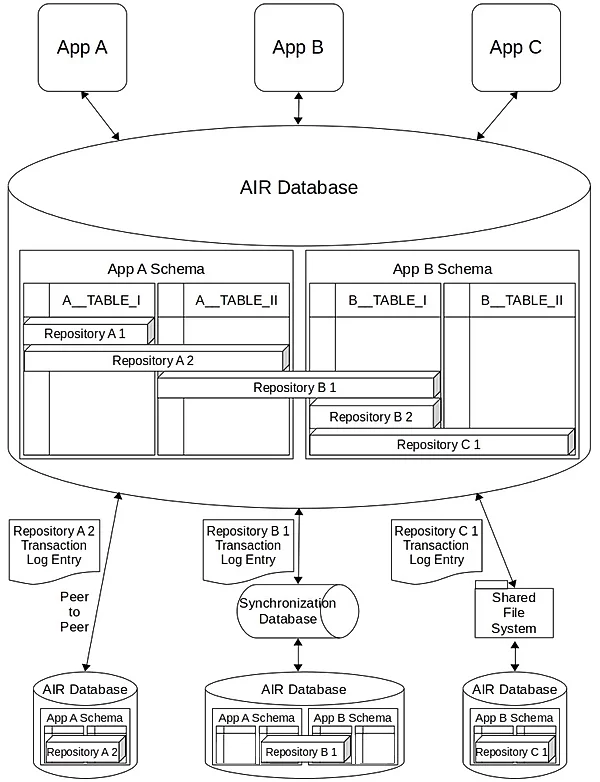
While repositories are fully autonomous, they are meant to be interdependent. Records can be referenced by one repository from another if they are guaranteed to never be deleted. By default, AIR is in “Create and Update” mode. A record that could potentially be deleted is marked as such and cannot be used as a dependency by another repository.

### Privacy by Design

User-centric data allows application providers to build in stronger privacy guarantees into their software. Application provider may only be interested in a subset of user data for their reporting and data mining. It usually isn’t in the interest of the providers of applications to be liable for the bulk of user’s private data. With AIR their applications get access to a defined set of data that limits their liability. And applications can further guarantee to the user that they themselves will only store an even smaller subset of data, leaving most of user’s data solely in AIR Databases and user’s archives.

## Technical Summary

An AIR Database consists of a database engine (such us a standard relational database) and additional logic for communicating with client applications, communicating with other AIR databases and maintaining internal state. AIR Databases may communicate with each other directly or via mechanisms such as a shared file system or a synchronization database server.



**Repositories**

Data in AIR Databases is grouped into Repositories. A Repository is a collection of records (table rows) across any number of tables. Repositories may reference other repositories. Circular dependencies between repositories are possible. Repositories are meant to be fine grained, time bound and limited to a particular shareable unit of knowledge, though exceptions are possible.

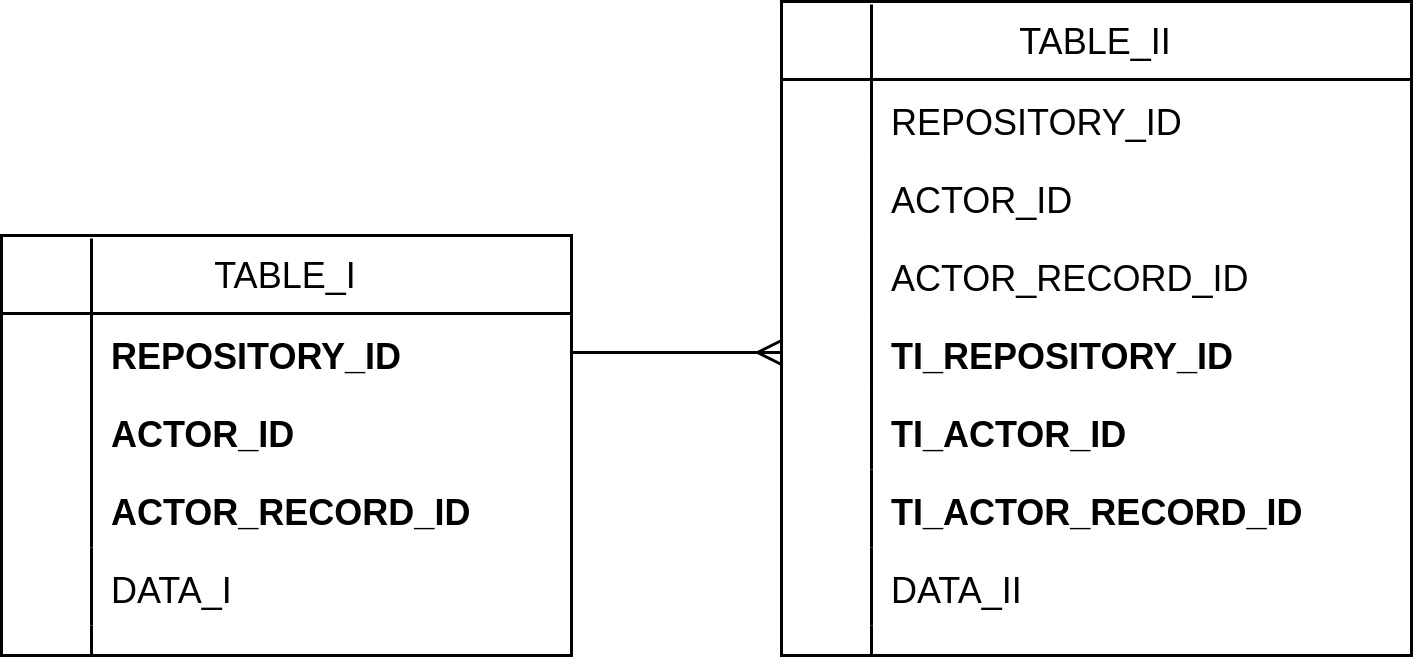
Repositories are meant to be grouped by hierarchies and labels. Definitions of hierarchies and labels are themselves stored in repositories and can be shared. Any grouping arrangement is possible. For example, hierarchies and labels can be defined in dedicated repositories that consist of references to the repositories grouped under them. Also, grouped repositories themselves can contain references to the hierarchies and labels that they are grouped by. Common hierarchies and labels may be published for all applications and users to use. Applications may publish their own hierarchies and labels and may request user to give access to (all or part of) user’s hierarchies and labels.

AIR Databases can store any number of repositories. Each AIR database is a fully functional relational database containing exactly the data that its user(s) have chosen to place into it. Additional repositories can be archived and may be retrieved from archives by an AIR database at any point. Any repository can be removed from an AIR Database at any time. AIR Database may disable referential integrity to allow Repositories it contains to reference Repositories it does not have.

**Repository Identifiers**

All repository records are identified by Repository Id, Actor Id and Actor Record Id. Actor is a specific user on a specific AIR repository. Combination of Repository Id, Actor and Actor Record Id are guaranteed to be unique within an AIR system. Once assigned, Ids cannot be updated, which guarantees fast “by Id” modification operations. Repository records reference records in other tables via (relational database) composite keys, which consist of the three keys described above.

By default, Application Id is incorporated into the Actor record making an Actor Id unique to a user using a particular Repository via a particular Application on a given device.



Repositories are modified in transactions. Client applications generate database wide transactions which are then split up into per-Repository transactions. One or more modification statements may exist in a Repository transaction. All operations in Repository transactions are always performed by unique record identifiers. An operation specified by the Client may not contain unique identifiers hence AIR Database converts all modification operations into individual “by Id” modification operations.

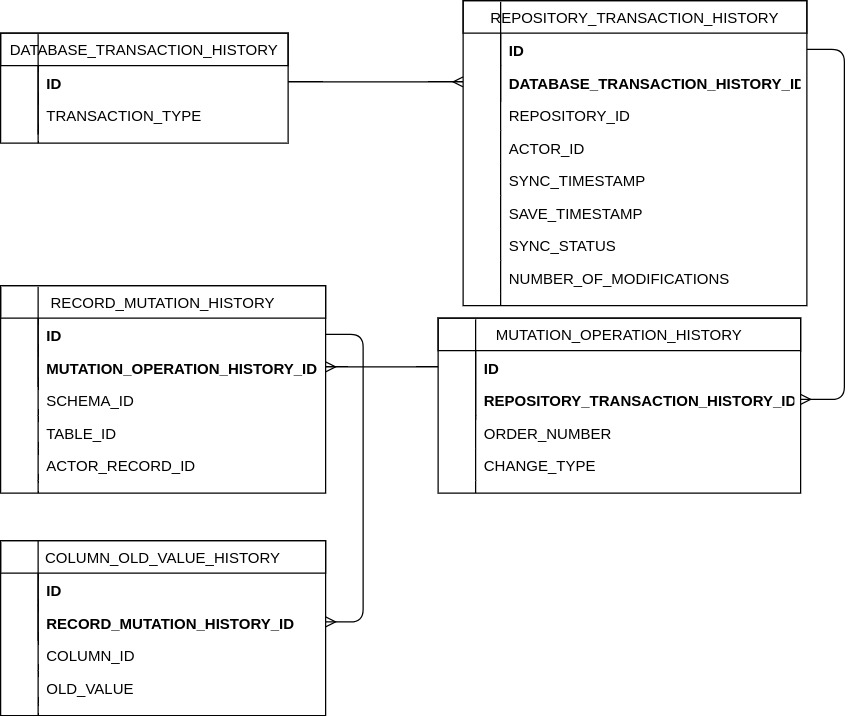
**Repository Sharing**

Repositories are shared between users. Users can be invited into a repository by the user owning the repository or any number of users granted the privilege to invite other users into a given repository. Users join a given repository and add modifications to it as Actors. Repository privileges are maintained in the metadata of those repositories. Metadata of repositories is shared using the same transaction mechanism as regular repository data.

All changes made to a given Repository are recorded in a Transaction Log with entries usually ordered by time of synchronization (which can be determined in various ways, depending on sharing mechanism and configuration). AIR databases communicate changes to a given Repository by exchanging entries of its Transaction Log. Repository Transaction Log entries are sent by AIR databases that generate them and are applied locally by the receiving AIR databases. A state of a given Repository may be different on different AIR Databases but is guaranteed to be “eventually consistent” across all databases. An archived Repository is the collection of all Transaction Log entries for that Repository, ordered by time of synchronization. To load a Repository from an archive an AIR Database re-executes all operations in the archived Transaction Log (in order of synchronization).

**Local History**

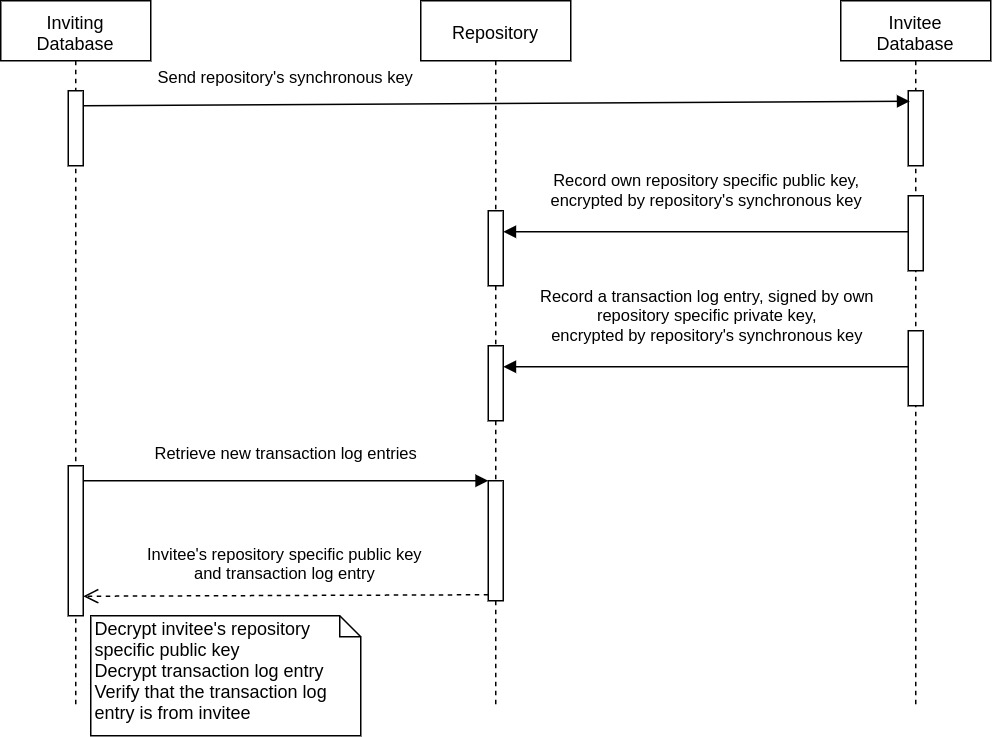
Transaction Logs are stored in an AIR database in the form of searchable local history (and can always be reconstructed from that history and vice versa). Local history is grouped by database transaction, repository transaction, operation, and modified entity (operation execution order is recorded as well).



**Security**

To ensure security, all Transaction Log entries may be encrypted using a repository specific symmetric encryption key. Repository symmetric encryption keys can be distributed via secure means with use of receiving Actor’s public key to encrypt them.

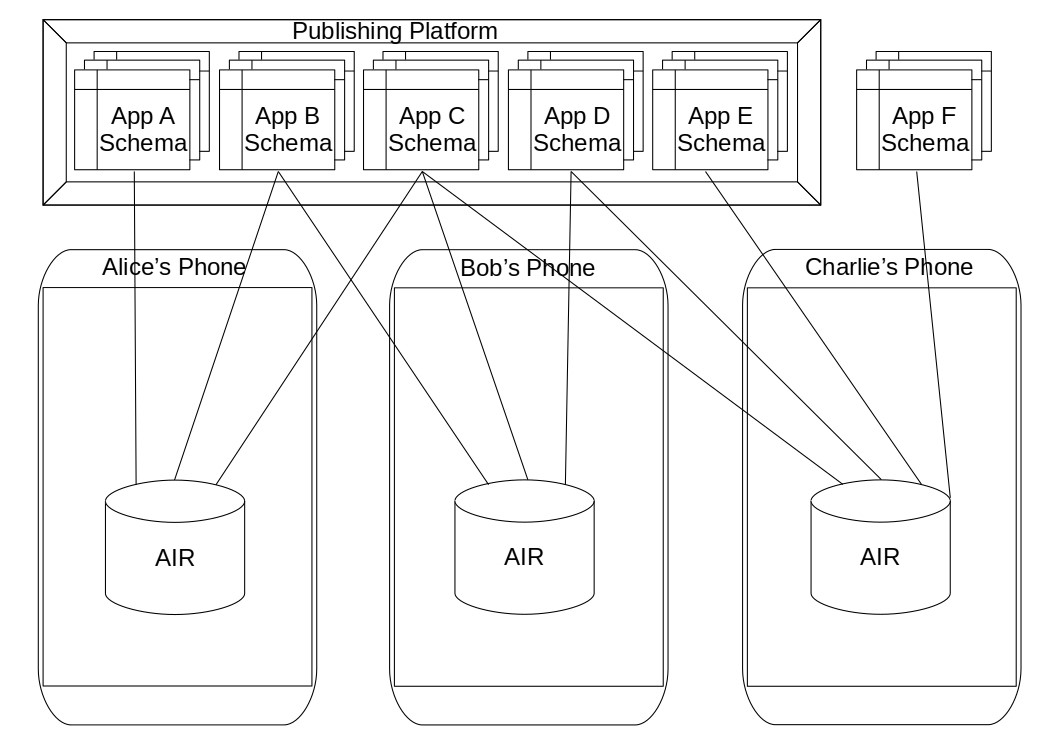
To ensure authenticity, all Transaction Log entries can be signed with Actor’s private key.



**Schemas**

Any number of applications from any number of different publishers may use an AIR database. Applications provide schemas to the AIR databases. Schemas can be distributed directly from an application or via a publishing platform.

Applications may store records in their own schemas or in schemas of other applications. Applications may allow other applications to use any part of their schemas for any type of standard CRUD (Create, Read Update, Delete) operations. Applications can grand access privileges to other applications and revoke those privileges from other applications at any time. Applications may use wildcards in the “access and modification” rules to specify groups of applications, groups of database tables and columns, or groups of users that are granted permissions (for any type of operation). Grants can be maintained in dedicated grant repositories and may be distributed by applications directly.



**Upgrades**

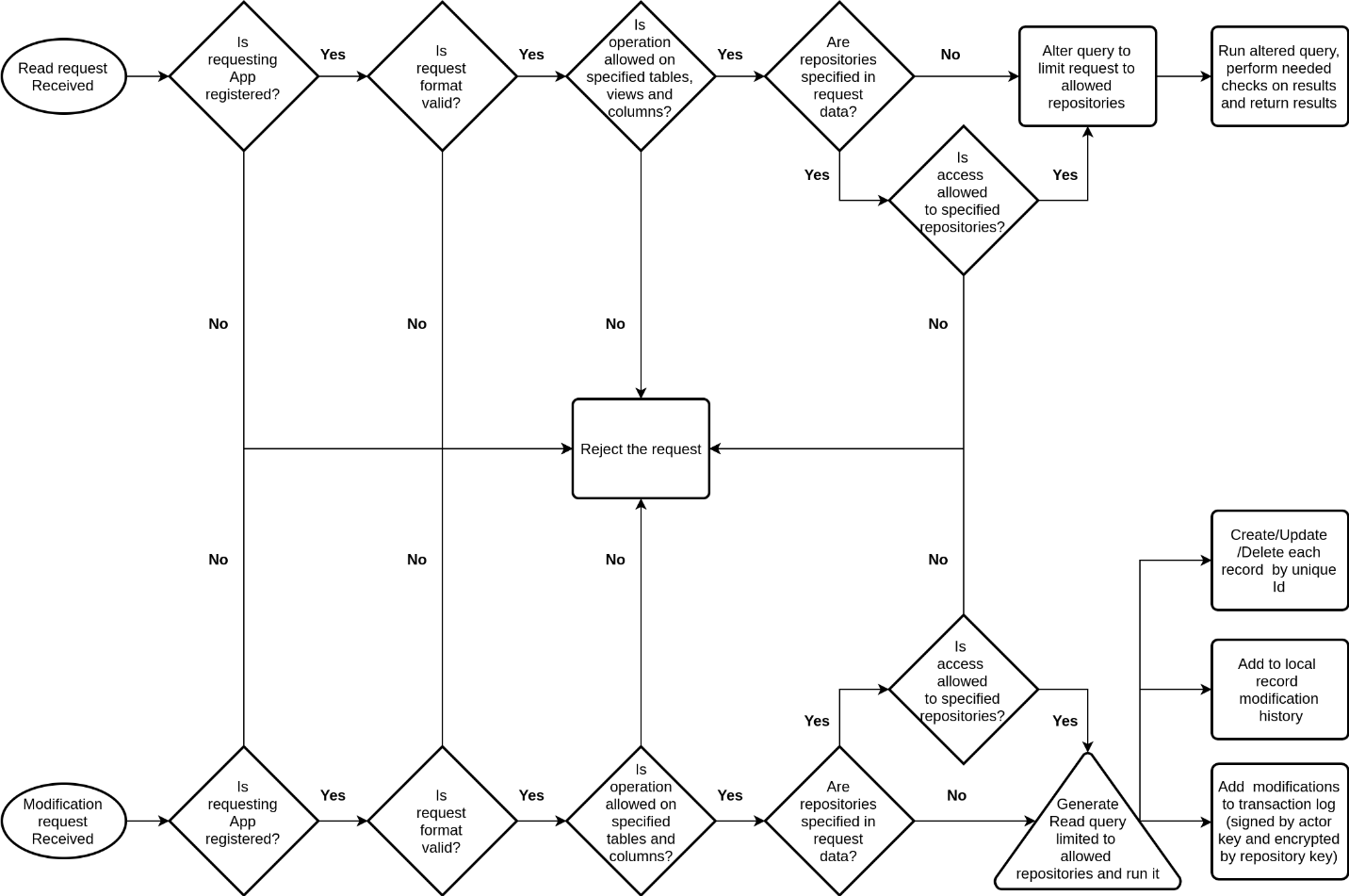
Schemas can be upgraded at any given point in time. AIR Databases require full backward compatibility across schema versions for Read operations. An application using an older version of the schema should always be able to read from a newer version of the schema. Backward read compatibility can be maintained via database views that mimic older versions of the schema. Application publishers must provide conversion code (securely executed) that upgrades data in AIR databases to new versions. They must also provide code that upgrades older versions of operations in incoming transaction logs (before the operations are applied to the databases). All operations (from applications and from transaction logs) are always validated for security and consistency.

**User access**

Users grant access and modification privileges for their Repositories to other users and Applications. These grants can be done on individual Repository basis or on a grouping of Repositories. Wildcards can be used to specify any group of users or Applications with permissions (for any type of operation). Users can specify sharing settings in metadata of Repositories.

**Access Checks**

Users use AIR databases via Applications. All Read and Modification requests coming from Applications are checked. AIR first makes sure that the calling Application is registered and allowed to access AIR for the specified operation. Then AIR verifies the validity of request. Then it checks if the calling Application is allowed to use (for the specified operation) the tables (or views, possibly used for read requests) and columns specified in the request. For this operation AIR uses access rules provided by the publishers of the schemas with the specified tables/views and columns. Then if any Repositories are explicitly referenced in a request, AIR checks if the calling Application and user are allowed to reference those Repositories (for the specified operation).

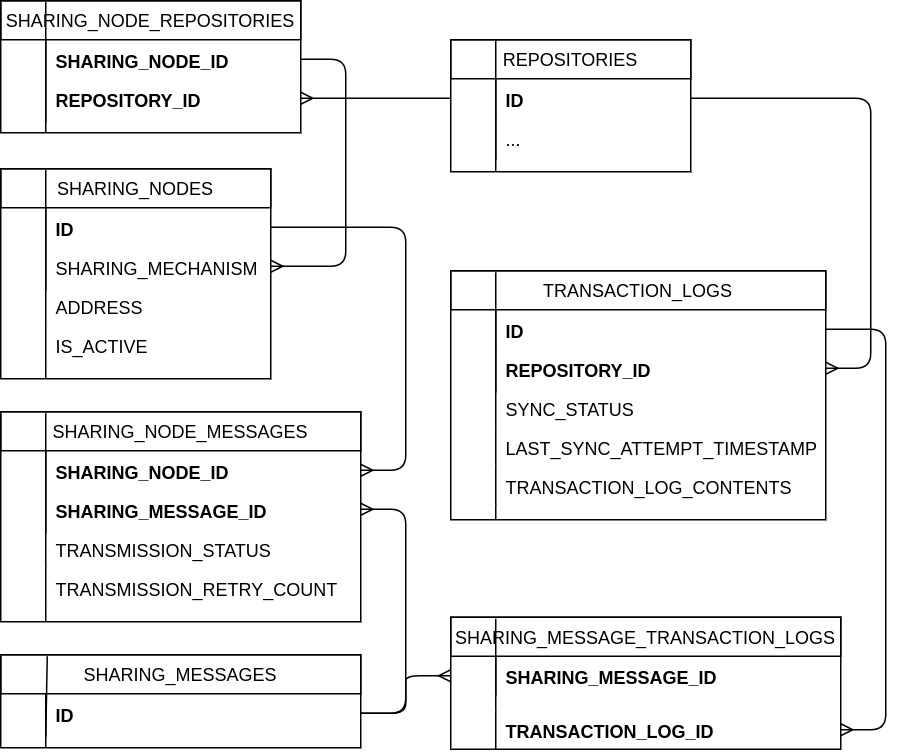


Specifically, for Read requests (once all of the above checks are performed) AIR alters the read query and joins it with Repository reference tables via Id of accessing Actor. This limits the query to only the Repositories that the calling Application and user are allowed to see for Read operations. Then AIR runs the altered query. The results of the query can then be checked (if needed, in specific cases) to further make sure that only data from the allowed Repositories is returned. Finally, the results are returned to the calling Application.

For Modification requests (once all of the checks common with Read operations are performed) AIR generates a read query, using the same limitation mechanisms as used for the Read requests. This query returns all of the records to be modified (Created, Updated or Deleted). For Create requests that explicitly specify all of the data and Ids this step is not performed (as it is not needed). Then AIR performs the corresponding modification in the database for each record (returned by the Read query or explicitly specified). One modification is run for each record (with grouping optimizations possible). Then AIR records all record modifications (with old values of the record, if any) in local record modification history. Finally, entries describing the modification are added to the Transaction Logs of the Repositories modified by the operation. One Transaction Log entry is created per database transaction, per Repository. Each Transaction Log entry may be signed by the Actor for authenticity and may be encrypted for privacy (by a symmetric key, which may be specific to a given Repository). Actor’s public/private key pair is specific to the acting AIR database is stored in the AIR database (with private key not accessible by any other entities).

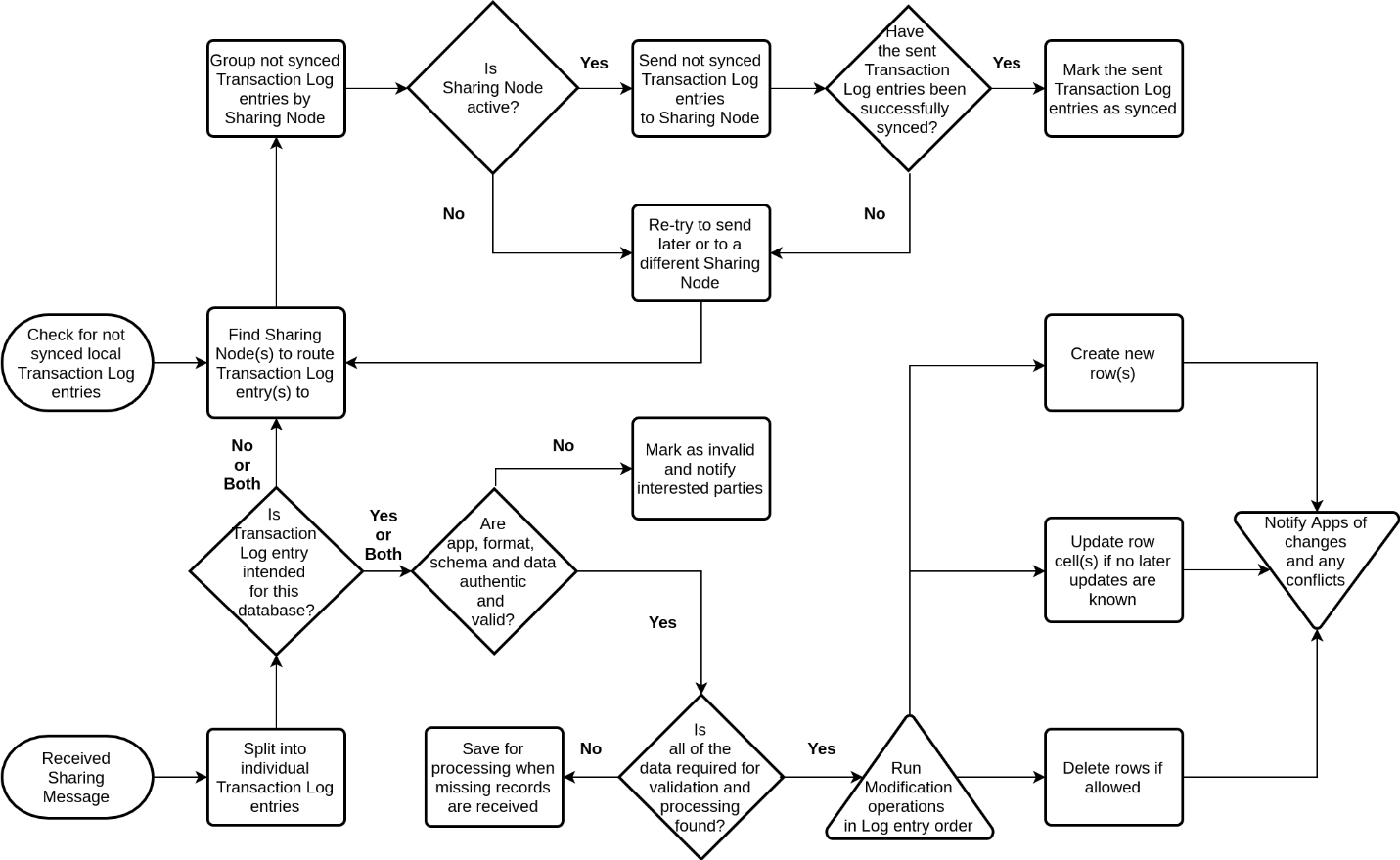
**Database Synchronization**

Every AIR database can act as a standalone database for any period of time. When a given Repository in it is shared, AIR database acts as a Sharing Node. A Sharing Node shares data with other AIR databases (directly or indirectly). It is responsible for sending out messages containing Transaction Log entries for the shared Repository to the correct Sharing Nodes (via the sharing mechanism(s) chosen for that Repository). It is also responsible for receiving messages containing Transaction Log entries for the Repositories it shares. In peer-to-peer sharing configurations it may also be responsible for maintaining routing configuration and for routing the messages (containing Transaction Log entries) to the appropriate Sharing Nodes.



**Synchronization Checks**

Periodically (or as a response to an appropriate event) an AIR Database checks its locally generated Transaction Log entries for any that haven’t been synchronized to other appropriate AIR Databases. If such entries are found then AIR looks up active Sharing Nodes to which these entries should be routed to. Then it groups the Transaction Logs to be synced to these Sharing Node(s) (a given Transaction Log could be sent to multiple Sharing Nodes) and sends them in sharing messages. Then, for each message, (either immediately upon confirming successful reception of the message, or after additional steps are performed to ensure propagation of changes to other Sharing Nodes or a sharing database server) the Transaction Log entries are determined to be synced (or not). If the Sharing Nodes cannot come to an agreement on marking the entries in question as being synced, then these entries may be resent (from any Sharing Nodes that have them). If an agreement on the synced status is achieved the Transaction Log entries in question are marked as synced.



When an AIR Database receives sharing messages it splits them into individual Transaction Log entries. Then for each entry AIR Database determines if this entry is intended for its local storage. If the message isn’t intended for this database (or if it is intended for this database and is expected to be forwarded to other AIR Databases as well) then the message is shared with other AIR databases (as described in the paragraph above). If the message is intended for this database then it is decrypted (where applicable), its authenticity is checked (where applicable) and is validated using a process alike to the validation of local modification operations. If all of the data required to process an incoming Transaction Log entry is found (data needed for validation, the data being updated or deleted, etc.) then all received entries for a given Repository are run in the order provided. For Update operations, specific record values (table row cells) are updated to the received values only if no subsequent updates to these values have been made. For Delete operations, records are deleted if they haven’t been subsequently updated or if configuration is specified allowing deletes of subsequently updated records. Finally, Applications subscribed to receive notifications of appropriate data changes are notified of the corresponding changes and (and of any resulting conflicts and conflict resolutions).