# Database Schema Tutorial

Prasenjit Sarkar
IBM

# Key Considerations

- Consolidation of data in one schema
  - 300+ tables and views
- High scalability and performance.
- Enable migration of data from previous installations.
- Minimize changes from earlier versions.
- Keep user-defined history of certain entities.

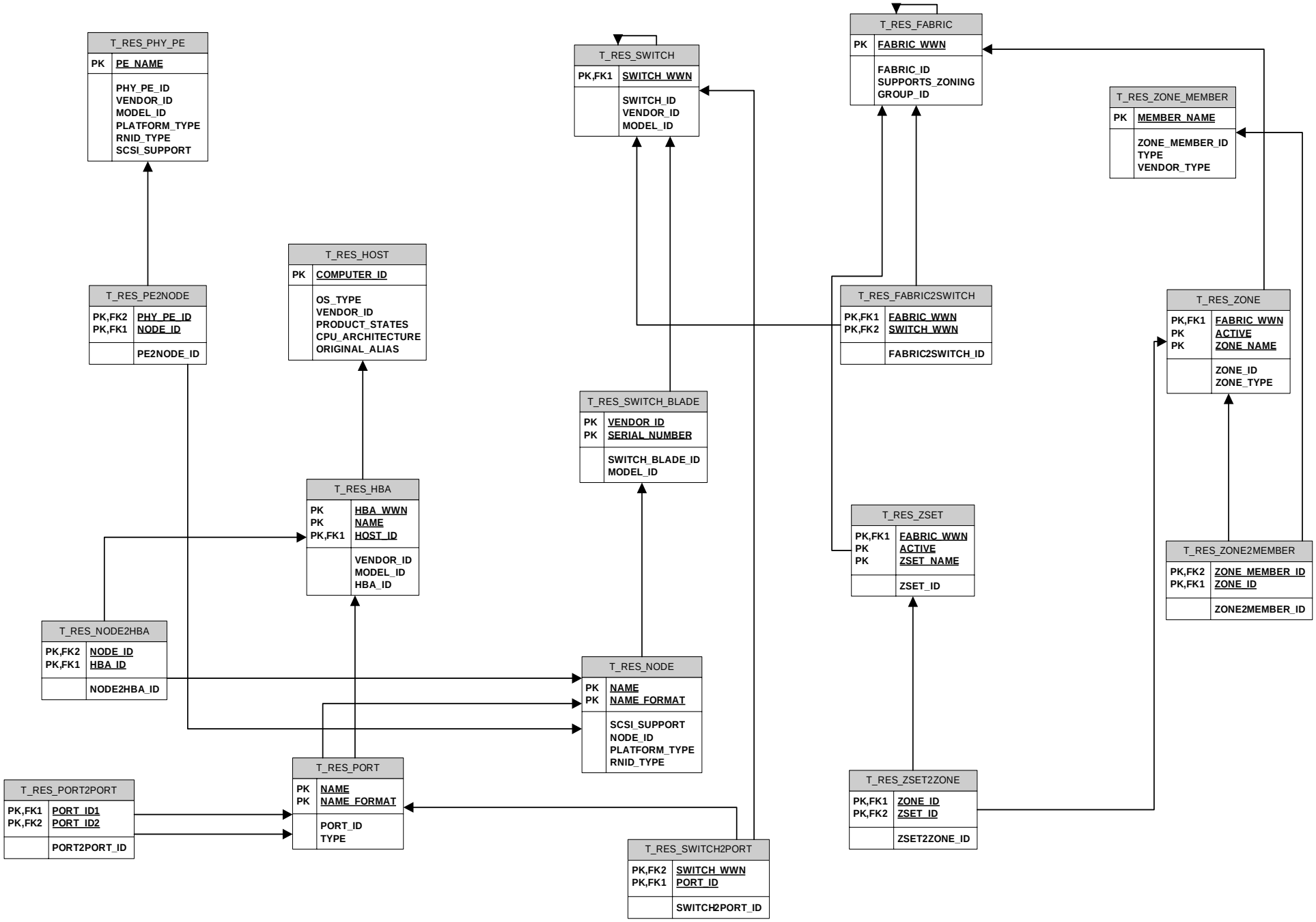
# Schema Features

- Single Schema – multiple table-spaces for components
- Table name convention:
  - T_[ENTITY_NAME] → Host entity or Ancillary table
  - T_RES_[ENTITY_NAME] → Host/Fabric/Disk entity
  - T_VIEW_[VIEW_NAME] → Views on top of the tables
  - T_STAT_[ENTITY_NAME] → Statistics about entities in question
  - T_RES_[ENTITY_NAME]_ATTRIBUTE_SNAPSHOT → Attribute Snapshot of entity
    - There are exceptions to naming convention
- Natural key = Primary key
  - Most external and inter-component interactions use natural key
- Foreign keys
  - lots of debate, left out because of stability concerns

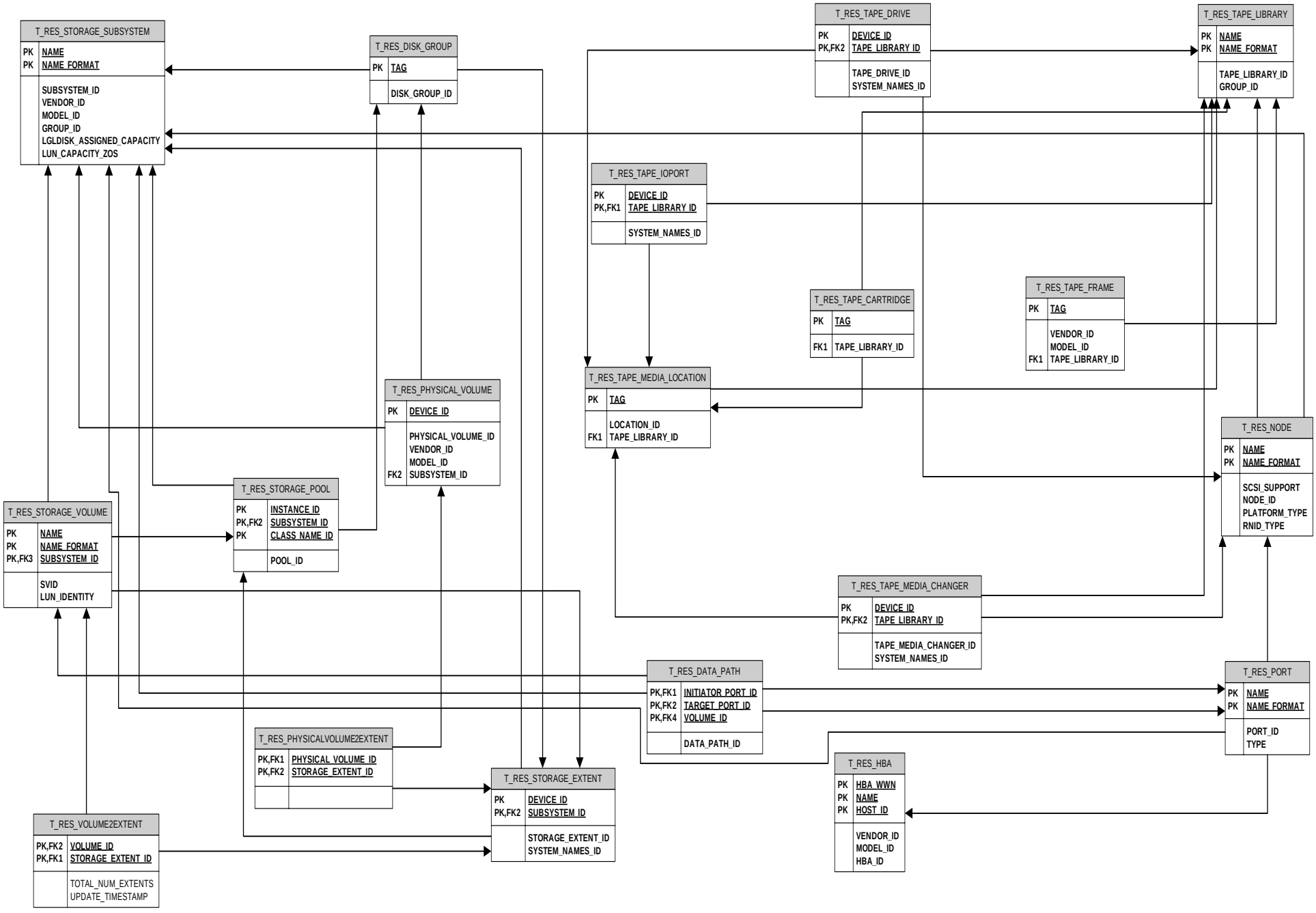
# Schema Features

- Auto-generated keys for variable-length primary key used as foreign key
  - Performance
  - Does not use database auto-generation facility
- Normalization for certain fields e.g. Vendor, Model
  - Transparent to database access layer
- Multiple CIM keys per CIM entity: T_RES_CIMKEY_
- Multiple Name/Name formats

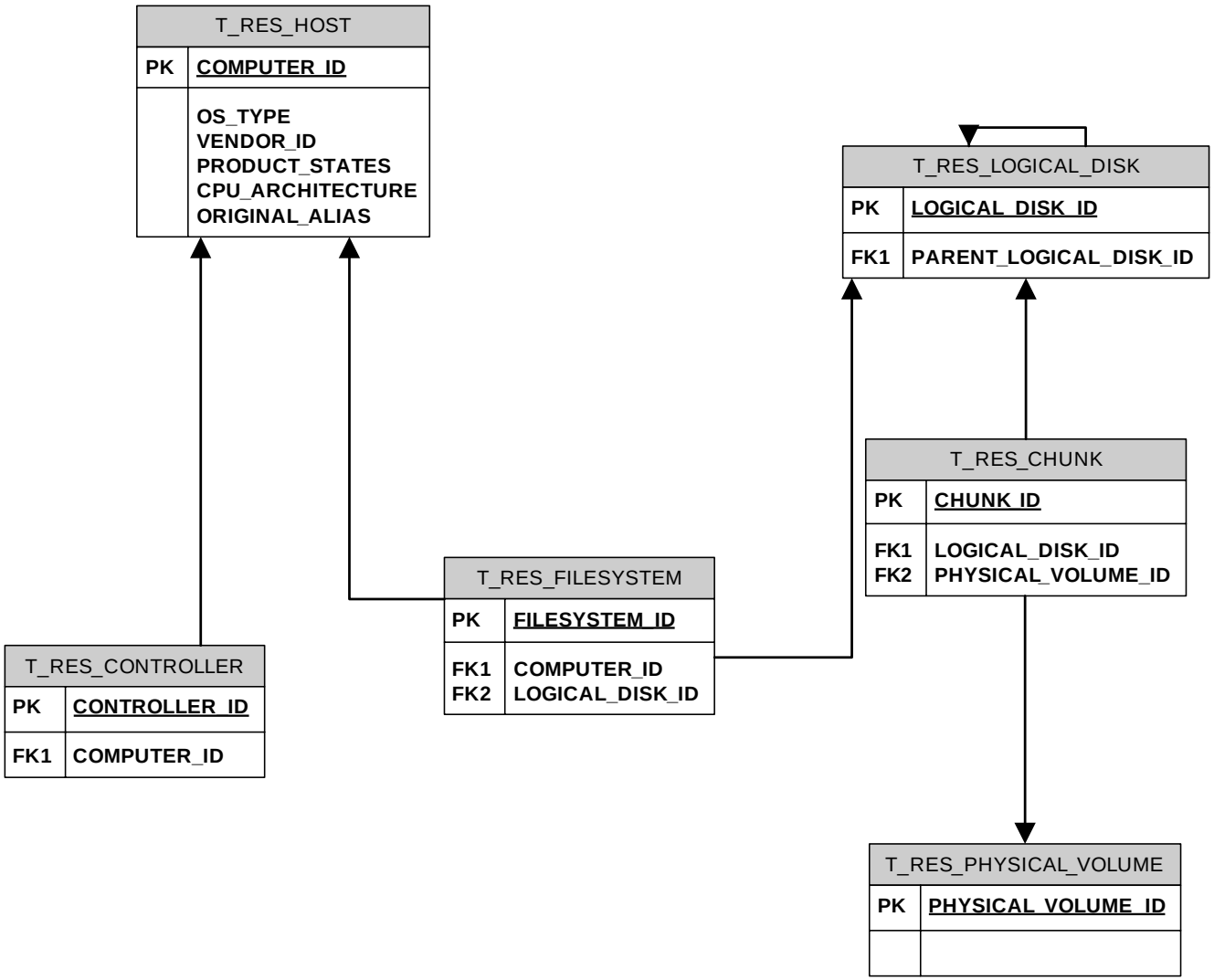
# Fabric


T_RES_PHY_PE
PK PE_NAME
PHY_PE_ID
VENDOR_ID
MODEL_ID
PLATFORM_TYPE
RNID_TYPE
SCSI_SUPPORT
T_RES_SWITCH
PK,FK1 SWITCH_WWN
SWITCH_ID
VENDOR_ID
MODEL_ID
T_RES_FABRIC
PK FABRIC_WWN
FABRIC_ID
SUPPORTS_ZONING
GROUP_ID
T_RES_ZONE_MEMBER
PK MEMBER_NAME
ZONE_MEMBER_ID
TYPE
VENDOR_TYPE
T_RES_HOST
PK COMPUTER_ID
OS_TYPE
VENDOR_ID
PRODUCT_STATES
CPU_ARCHITECTURE
ORIGINAL_ALIAS
T_RES_PE2NODE
PK,FK2 PHY_PE_ID
PK,FK1 NODE_ID
PE2NODE_ID
T_RES_FABRIC2SWITCH
PK,FK1 FABRIC_WWN
PK,FK2 SWITCH_WWN
FABRIC2SWITCH_ID
T_RES_ZONE
PK,FK1 FABRIC_WWN
PK ACTIVE
PK ZONE_NAME
ZONE_ID
ZONE_TYPE
T_RES_SWITCH_BLADE
PK VENDOR_ID
PK SERIAL_NUMBER
SWITCH_BLADE_ID
MODEL_ID
T_RES_HBA
PK HBA_WWN
PK NAME
PK,FK1 HOST_ID
VENDOR_ID
MODEL_ID
HBA_ID
T_RES_ZSET
PK,FK1 FABRIC_WWN
PK ACTIVE
PK ZSET_NAME
ZSET_ID
T_RES_ZONE2MEMBER
PK,FK2 ZONE_MEMBER_ID
PK,FK1 ZONE_ID
ZONE2MEMBER_ID
T_RES_NODE2HBA
PK,FK2 NODE_ID
PK,FK1 HBA_ID
NODE2HBA_ID
T_RES_NODE
PK NAME
PK NAME_FORMAT
SCSI_SUPPORT
NODE_ID
PLATFORM_TYPE
RNID_TYPE
T_RES_PORT
PK NAME
PK NAME_FORMAT
PORT_ID
TYPE
T_RES_ZSET2ZONE
PK,FK1 ZONE_ID
PK,FK2 ZSET_ID
ZSET2ZONE_ID
T_RES_PORT2PORT
PK,FK1 PORT_ID1
PK,FK2 PORT_ID2
PORT2PORT_ID
T_RES_SWITCH2PORT
PK,FK2 SWITCH_WWN
PK,FK1 PORT_ID
SWITCH2PORT_ID

# Disk, Tape


T_RES_STORAGE_SUBSYSTEM
PK NAME
PK NAME_FORMAT
SUBSYSTEM_ID
VENDOR_ID
MODEL_ID
GROUP_ID
LGLDISK_ASSIGNED_CAPACITY
LUN_CAPACITY_ZOS
T_RES_DISK_GROUP
PK TAG
DISK_GROUP_ID
T_RES_TAPE_DRIVE
PK DEVICE ID
PK,FK2 TAPE_LIBRARY ID
TAPE_DRIVE_ID
SYSTEM_NAMES_ID
T_RES_TAPE_LIBRARY
PK NAME
PK NAME_FORMAT
TAPE_LIBRARY_ID
GROUP_ID
T_RES_TAPE_IOPORT
PK DEVICE ID
PK,FK1 TAPE_LIBRARY ID
SYSTEM_NAMES_ID
T_RES_TAPE_CARTRIDGE
PK TAG
FK1 TAPE_LIBRARY_ID
T_RES_TAPE_FRAME
PK TAG
VENDOR_ID
MODEL_ID
FK1 TAPE_LIBRARY_ID
T_RES_TAPE_MEDIA_LOCATION
PK TAG
LOCATION_ID
FK1 TAPE_LIBRARY_ID
T_RES_PHYSICAL_VOLUME
PK DEVICE ID
PHYSICAL_VOLUME_ID
VENDOR_ID
MODEL_ID
FK2 SUBSYSTEM_ID
T_RES_NODE
PK NAME
PK NAME_FORMAT
SCSI_SUPPORT
NODE_ID
PLATFORM_TYPE
RNID_TYPE
T_RES_STORAGE_POOL
PK INSTANCE ID
PK,FK2 SUBSYSTEM ID
PK CLASS_NAME_ID
POOL_ID
T_RES_STORAGE_VOLUME
PK NAME
PK NAME_FORMAT
PK,FK3 SUBSYSTEM ID
SVID
LUN_IDENTITY
T_RES_TAPE_MEDIA_CHANGER
PK DEVICE ID
PK,FK2 TAPE_LIBRARY ID
TAPE_MEDIA_CHANGER_ID
SYSTEM_NAMES_ID
T_RES_DATA_PATH
PK,FK1 INITIATOR_PORT_ID
PK,FK2 TARGET_PORT_ID
PK,FK4 VOLUME ID
DATA_PATH_ID
T_RES_PORT
PK NAME
PK NAME_FORMAT
PORT_ID
TYPE
T_RES_PHYSICALVOLUME2EXTENT
PK,FK1 PHYSICAL_VOLUME_ID
PK,FK2 STORAGE_EXTENT_ID
T_RES_STORAGE_EXTENT
PK DEVICE ID
PK,FK2 SUBSYSTEM ID
STORAGE_EXTENT_ID
SYSTEM_NAMES_ID
T_RES_HBA
PK HBA_WWN
PK NAME
PK HOST_ID
VENDOR_ID
MODEL_ID
HBA_ID
T_RES_VOLUME2EXTENT
PK,FK2 VOLUME_ID
PK,FK1 STORAGE_EXTENT_ID
TOTAL_NUM_EXTENTS
UPDATE_TIMESTAMP

# Data


T_RES_HOST
PK
COMPUTER_ID
OS_TYPE
VENDOR_ID
PRODUCT_STATES
CPU_ARCHITECTURE
ORIGINAL_ALIAS
T_RES_LOGICAL_DISK
PK
LOGICAL_DISK_ID
FK1
PARENT_LOGICAL_DISK_ID
T_RES_CHUNK
PK
CHUNK_ID
FK1
FK2
LOGICAL_DISK_ID
PHYSICAL_VOLUME_ID
T_RES_FILESYSTEM
PK
FILESYSTEM_ID
FK1
FK2
COMPUTER_ID
LOGICAL_DISK_ID
T_RES_CONTROLLER
PK
CONTROLLER_ID
FK1
COMPUTER_ID
T_RES_PHYSICAL_VOLUME
PK
PHYSICAL_VOLUME_ID

# Example: Storage Subsystem

| COLUMN NAME | COMMENT |
|---|---|
| SUBSYSTEM_ID | Auto-generated key to use as foreign key in other tables. |
| ALLOCATED_CAPACITY | Total storage capacity already used. |
| AVAILABLE_CAPACTIY | Total storage capacity available for allocation. |
| BACKEND_STORAGE_CAPACITY | Total backend disk storage capacity. |
| VENDOR | Vendor. Normalized Key. |
| MODEL | Model. Normalized Key. |
| CREATION_CLASS_NAME | SMI-S key for creation class name. Normalized Key. |
| SITE_ID | ID of site where the storage subsystem is located. |
| NAME | Name of subsystem in the format specified in column NAME_TYPE.<br>Natural Key.<br>Example: For ESS, NAME = 2105.21974, NAME_TYPE = Normalized key representing Other. For SVC, NAME = 9.11.213.191:0000020060400A8A, NAME_TYPE = Normalized key representing Other. |
| NAME_TYPE | Name format of subsystem.<br>Natural Key. Normalized Key.<br>Example: Normalized key representing Other, IP, WWN, NAA |
| CACHE | Total cache (MB) of subsystem. |
| NVS | Non-volatile storage (MB) of subsystem. |
| DG_FREESPACE | Free space of disk groups. |
| N_DISKS | Number of disks. |

# Storage Subsystem (Continued)

| COLUMN NAME | COMMENT |
|---|---|
| N_LUNS | Number of LUNs. |
| LAST_PROBE_TIME | Last probe time of subsystem. |
| PROBE_STATUS | Last probe status of subsystem. |
| GROUP_ID | Group ID (Data Legacy). |
| PROBING_SERVER_ID | Host that detects the subsystem (Data Legacy). |
| VG_CAPACITY | Capacity of Volume Groups. |
| VG_FREESPACE | Free space of Volume Groups. |
| OS_TYPE | Type of operating system (Data Legacy). |
| PRODUCT_STATES | Product states (Data Legacy). |
| TYPE | Type of subsystem. |
| IP_ADDRESS | IP Address of subsystem. |
| DNS | DNS of subsystem. |
| WWN | WWN of subsystem. |
| MAC_ADDRESS | Mac Address of subsystem. |
| SERIAL | Serial Number of subsystem. |
| CODE_LEVEL | Code Level of subsystem. |
| NAMESPACE | CIM Namespace of subsystem. |

# DB Helpers…

- Isolate most of schema/JDBC issues from application programmer
  - Programmer responsible for maintaining transaction semantics
- Code using DB Helpers is easier to read
  - Generic style:

```
update = dbConn.prepareStatement(UPDATE_ID_SQL);
update.setInt(1, idCount);
result = update.executeUpdate();
```

  - DB Helper style:

```
my_Identifier.setIdCount(idCount);
result = my_Identifier.update(dbConn);
```

# Where to start

- Programmer deals with business objects
  - Business objects are mapped to database tables
    - Mappings are not necessarily 1-1
    - Mappings are not exposed to programmer
  - Examples of business objects:
    - StorageVolume
    - DeviceAgent
  - Business object names are closely correlated to database table names
    - StorageVolume is the business object for T_RES_STORAGE_VOLUME
  - Business objects are located in `com.ibm.tpc.infrastructure.database.objects`

# Overview of DB Helpers

- Getting a connection
- Single Instance Operations
- Cursors
- Normalization
- Multi-row updates and deletes
- Complex SQL queries
- Timestamps
- Auto-Identifiers

# Getting a connection

- How to get a connection:
  - Use the JDBC Connection Pool interface
  - Connection pool is a collection of open connections
    - Avoids penalty of reconnect
    - Takes care of stale connections
  - Get parameters from properties file

```
DBConnPoolDataSource ds =
   DBConnPoolDataSource.getInstance(driverName, dbURL);
connection = (DBConnection) ds.getPooledConnection();
```

  - Remember to return connection back to pool

```
connection.close();
```

# Single Instance Operations

- Operate on single object instances only
  - Inserts, deletes, updates and retrieves
  - Cannot be used if the database operation uses multiple instances of an object
  - Input parameters can be object member variables or an associated hash-table
  - Input parameters MUST include primary key of object to ensure single instances
  - Illustrated through examples
  - Actual interfaces documented in SDD1

# Single Instance Ops: Example 1

```
private static boolean Example1(DBConnection con)  {
   try {
        RegisteredCimom myCimom = new RegisteredCimom();
        myCimom.setServiceUrl("http://127.0.0.1:5050");
        myCimom.setInteropSchemaNamespace("/root/ibm");
        result = myCimom.insert(con);
        myCimom.setInteropSchemaNamespace(“/root/appiq”);
        result = myCimom.update(con);
        RegisteredCimom cimom2 = new RegisteredCimom();
        cimom2.setServiceUrl("http://127.0.0.1:5050");
        cimom2.retrieve(con);
        result = equals(cimom2, myCimom);
        myCimom.delete(con);
        con.commit();
   } catch (SQLException eDB) {dumpException(eDB); return false;}
   return result;
}
```

# Single Instance Ops: Example 2

```
private static boolean createTest1(DBConnection con)  {
   try {
        RegisteredCimom myCimom = new RegisteredCimom();
        Hashtable h = new Hashtable();
        h.put(RegisteredCimom.getColumnInfo("SERVICE_URL"),
   "http://127.0.0.1:5050");

        h.put(RegisteredCimom.getColumnInfo("INTEROP_SCHEMA_NAMESPACE"
   ), "/root/ibm");
        result = myCimom.insert(con, h);

        h.put(RegisteredCimom.getColumnInfo("INTEROP_SCHEMA_NAMESPACE"
   ), "/root/appiq");
        result = myCimom.update(con, h);
        RegisteredCimom cimom2 = RegisteredCimom.retrieve(con, h);
        cimom2.delete(con, h);
        con.commit();
   } catch (SQLException eDB) {dumpException(eDB); return false;}
   return result;
}
```

# Cursors

- Used to retrieve multiple instances of a business object
- The where clause is restricted to ANDs and equality
  - Eg key1 = val1 AND key2 = val2 AND .. AND keyn = valn
- Every business object has a corresponding Cursor class
  - StorageVolumeCursor is the cursor class for StorageVolume business object
- Can retrieve specific columns of the business object to speed up retrieval
- Can choose not to allocate a new object for every retrieval.

# Cursors: Example 1

```
htWhereClause.put(RegisteredCimom.getColumnInfo("INTERO
   P_SCHEMA_NAMESPACE"), "/root/ibm");
htWhereClause.put(RegisteredCimom.getColumnInfo(“ALIAS"
   ), “OEM_BOX");
RegsiteredCimomCursor cursor =
   RegisteredCimom.getCursor(con, htWhereClause, null);
while (cursor.next()) {
   RegisteredCimom cimom = (RegisteredCimom)
   cursor.getObject();
}
```

# Cursors: Example 2

```
htWhereClause.put(RegisteredCimom.getColumnInfo("INTERO
   P_SCHEMA_NAMESPACE"), "/root/ibm");
htWhereClause.put(RegisteredCimom.getColumnInfo(“ALIAS"
   ), “OEM_BOX");
RegsiteredCimomCursor cursor =
   RegisteredCimom.getCursor(con, htWhereClause, null);
while (cursor.next()) {
   RegisteredCimom cimom = (RegisteredCimom)
   cursor.getAttribute(); //no object created
}
```

# Cursors: Example 3

```
Vector attribs  = new Vector (“SERVICE_URL”, “PROTOCOL”);
htWhereClause.put(RegisteredCimom.getColumnInfo("INTEROP_SCHE
   MA_NAMESPACE"), "/root/ibm");
htWhereClause.put(RegisteredCimom.getColumnInfo(“ALIAS"),
   “OEM_BOX");
RegsiteredCimomCursor cursor =
   RegisteredCimom.getCursor(con, htWhereClause, attribs);
while (cursor.next()) {
   Object[] cols = (RegisteredCimom) cursor.getObject();
   //you get back columns only
   String service_url = (String) cols[0];
   String protocol = (String) cols[1];
}
```

# Normalization

- Certain fields in database tables are normalized
  - E.g. Model, Vendor
- Normalization is made transparent to business object
- For example:
  - In T_RES_SWITCH, Model field is type SHORT (normalized)
  - However, in business object Switch, you can use:
    - void Switch.setModel(String modelName)
    - String Switch.getModelString()
      - Java does not allow String Switch.getModel()
- Works with hashtables also

# Normalization: Example 1

```
Switch sw = new Switch();
sw.setWwn("0123456789ABCDEF");
sw.setSwitchId(2);
sw.setVendor("A");
result = sw.insert(con);
String vendor = sw.getVendorString();
```

# Normalization: Example 2

```
Switch sw = new Switch();
Hashtable h = new Hashtable();
h.put(Switch.getColumnInfo("WWN"),
  "0123456789ABCDEF");
h.put(Switch.getColumnInfo("VENDOR"), "C");
h.put(Switch.getColumnInfo("SWITCH_ID"), "2");
result = sw.insert(con, h);
String vendor = sw.getVendorString();
```

# Multiple Row Update and Delete

- Some tables do not have primary keys
- Also may need to update or delete multiple rows
- Interface provided

```
public static int update(DBConnection con,
  Hashtable htinput, Hashtable htWhereClause)
  throws SQLException;
public static int deletes(DBConnection con,
  Hashtable htWhereClause) throws
  SQLException;
```

# Complex SQL Statements

- Cant avoid complex SQL
- Support in package com.ibm.tpc.infrastructure.sql
  - class is of the name Sql<ObjectName>.java where <ObjectName> is the name of business object
  - Class is of the name Sql<ObjectNames in alphabetical order>.java when multiple business objects are involved
- Two interfaces:
  - `DBResultSet sqlQuery(DBConnection con, int namedSqlIndex, Hashtable htparams);`
  - `int sqlUpdate(DBConnection con, int namedSqlIndex, Hashtable htparams);`
- The Hashtable `htparams` are given as parameters to the sql statement

# How to add or update queries

- Makes sure that index refers to an array of queries in the class
  - Allows pooling of queries for general use
- Content management an issue
  - Never overwrite checked in queries
    - Checked in queries are supposed to be unit tested
- Annotate query with users of the query
  - Needed to communicate defect fixes
    - E.g. @user: John Doe (johndoe@us.ibm.com)

# Complex SQL Queries: Example

- Consider SqlStorageVolume.java

```
public static final int maxSizeQuery = 0;
public static final int minSizeQuery = 1;
private static String sqlArray[] = {  “SELECT
  MAX(SIZE) FROM TPCDB.TPC_STORAGE_VOLUME”,
                        “SELECT MIN(SIZE) FROM
  TPCDB.TPC_STORAGE_VOLUME” }
```

# Timestamps

- Database responsible for timestamp generation
  - Avoids synchronization issue among multiple servers

```
public class CurrentTime {


  public static Timestamp
  getTimestamp(DBConnection dbcon)
  throws SQLException
```

# Auto-Indentifiers

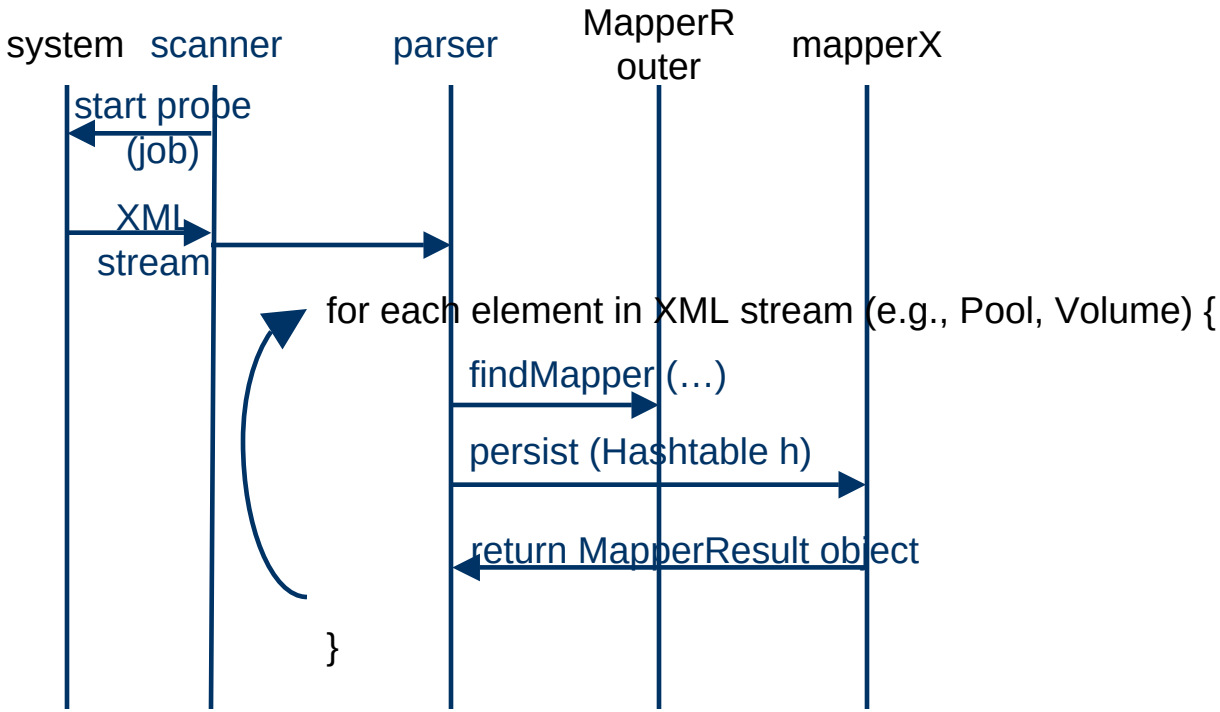
- Needed to efficiently lookup tables with complex primary keys
- Interface is as follows:

```
public class AutoIdentifier {
   //Enumeration of id type values
   public static final int RESOURCE_ID = 1;
   public static final int RUN_NUMBER_ID = 2;
   public static final int SCHEDULE_ID = 3;
   public static final int JOB_ID = 4;
   public static final int CALENDAR_ID = 5;
   public static final int ALERT_LOG_ID = 6;
   public static final int DEVICE_SERVER_RUN_ID = 7;
   public static short getIdentifier(
     int idType,
     int idCount,
     DBConnection dbConnection)
     throws SQLException
```

# Mappers

- The Mapper Layer was created in order to mask database implementation information from the Discovery Process
- All intricacies of mapping information related to a discovered object (ex: IBMTSESS_StorageSubsystem) to the appropriate columns and tables in the database are handled by the object’s mapper

# High Level Flow



# Mapper Usage

As the flow diagram indicates, two calls to the mapper layer are necessary to persist a discovered storage object or relationship

1. MapperRouter.findMapper – This function takes the CIMClass of the discovered entity as well as any desired additional specifics (SMIS version, device model, Dedicated property value, etc) and returns the appropriate mapper object based on the information provided by the mappers themselves (via mapper registration functionality).
2. DBAttributeMapper.persist – There are two implementations of this function, one for discovered objects representing storage entities and another for those representing relationships between storage entities. Both take a hashtable of (XMLAttribute name, value) pairs for each storage entity involved and then do the necessary work to persist the information. Both also require a valid connection to the database.

# Usage (continued)

- Although the Mapper Layer hides details of the database design, it does not create its own connections to the database – users of the Mapper Layer must provide a valid DBConnection object themselves and decide when to commit the actions taken by the Mapper Layer
- The Mapper Layer returns a complex object which provides information on what data was inserted and/or updated by the call to the persist function.