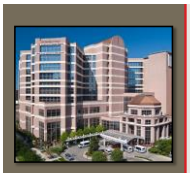


THE UNIVERSITY OF TEXAS
MD Anderson Cancer Center
Making Cancer History®

TH-CD-201
Clinical Networks
IT for Radiation Oncology

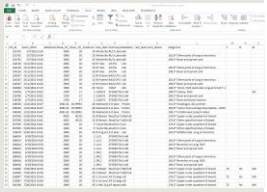
Radiation Therapy Databases (and a brief introduction to SQL)

- Peter Balter, Ph.D.

What is a database

- It is an organized collection of data
  - Could be a paper file system
  - Could be spread sheet

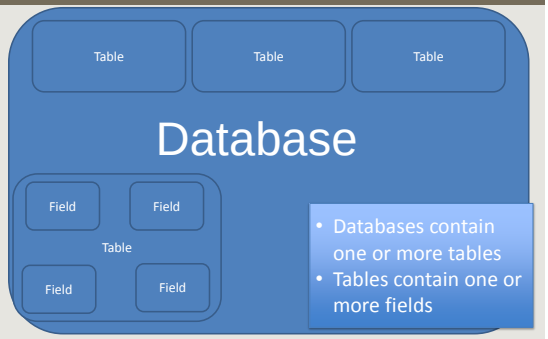
  (but neither are very good databases)



Databases in Radiotherapy

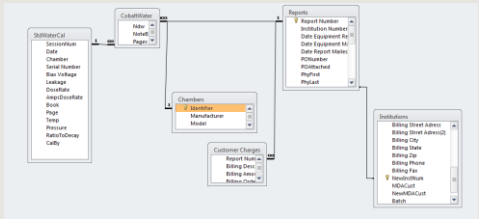
- Many of the systems in Radiotherapy are primarily databases with specialized equipment and user interfaces
  - Elketa MOSAIQ
  - Varian ARIA
  - Sun Nuclear Atlas
  - Teramedica (FUJI) Evercore
- Other systems are not primarily database but generate a large amount of data that is organized using a database
  - Varian RPM
  - Radcalc
  - Pinnacle

Relational Database



Relational Databases

- Data is grouped into tables based on its content
- The groupings have known relationships to each other
- The structure of the database is fixed and linked to real-world entites (business models)

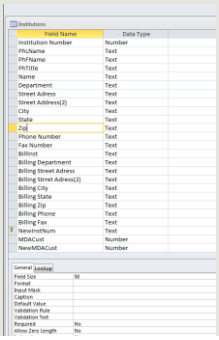


Post Relational Databases (NoSQL)

- No predefined structure: each record (document) caries its own fields and values with it (keys)
- Considered by many to be a superior way to manage data
  - Requires much less up-front work than a relational database
  - Is more flexible to changing business needs than relational databases
- Various systems available: MongoDB
- Will be covered in some Post 2015 AAPM by some one Post me

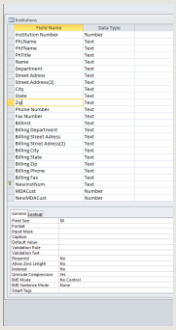
Relational Database: Tables

- Each table generally relates to a real-world entity
  - Example: all contact information for an institution may be grouped in a single table



Relational Database: Fields

- Each record is stored in a table as a collection of fixed length fields
- Each field has a data type and a length
  - Similar to data type in programing languages
  - float, real, char, text, date
  - Each takes up a fixed amount of storage space
- Data Types may be non-intuitive :
  - ZIPcode is stored as Text with 50 characters rather than numeric with 7 digits to allow free-form entry of foreign zip-codes (also worked well when +4 zips codes where introduced)
- Data validation can occur by database design or by rules enforced by input forms



Relationships

- One-to-Many
  - A row in one table can have a large number of matching rows in another table
  - This is the most common relationship
  - Example: A single plan can have many beams but each beam can only belong to a single plan
- Many-to-Many
- One-to-One

Relationships:

- One-to-Many
- Many-to-Many
  - A row in one table can have a large number of matching rows in another table and vise-versa
  - Example: Plans can have multiple imagesets and imagesets can belong to multiple plans
- One-to-One

Relationships:

- One-to-Many
- Many-to-Many
- One-to-One
  - Each row in one table can have exactly one matching row in another table
  - Not common as if this relationship exists the rows could have been placed in the same table.

Primary Keys

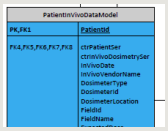
- Most tables have a primary key (field) that is used to define the relationship between that table with other tables
- The primary key can be the composite of 2 fields
- Often the primary key is a sequence number rather than a physical characteristic of the entity
  - Example: The primary key on a person could be their name but names change or could have been miss-spelled on entry so most databases with have a patient number usually independent of the hospital patient ID

Example from Varian Unified Reports Application Schema (PK is primary key)

Foreign Keys

- Foreign Keys are fields in a table that point to the Primary Key of another table
  - Are restricted to being available in the other table to enforce data integrity

In this example the primary key for PatientinVivoDatModel is also a foreign key since inVivo dosimetry must be related to a unique patient

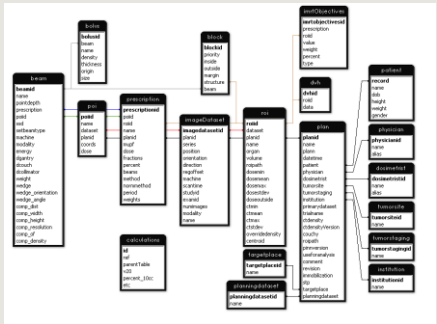


Example from Varian Unified Reports Application Schema (FK: is foreign key)

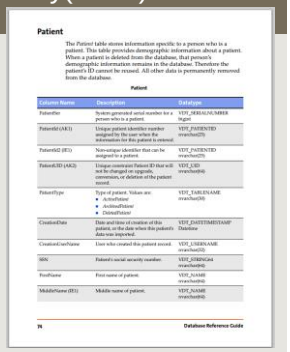
Database Schema/Data Dictionary

- Database Schema
  - The blueprint for the database
  - Shows how data is divided into tables and how tables relate to one another
  - Shows integrity constrains
  - Shows stored procedures
- Data Dictionary
  - Should include much of the same information as the schema
  - May contain further information
    - Detailed descriptions of the data

Typical Database Schema (in-house Pinnacle add-on database)



Data Dictionary(ARIA)

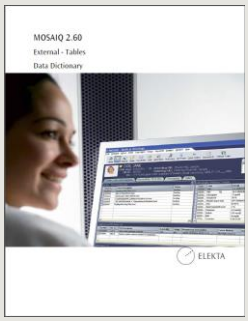

Patient

The *Patient* table stores information specific to a person who is a patient. This table provides demographic information about a patient. When a patient is deleted from the database, that person's demographic information remains in the database. Therefore the patient's ID cannot be reused. All other data is permanently removed from the database.

Patient

Column Name	Description	Datatype
PatientSer	System-generated serial number for a person who is a patient.	VDT_SERIALNUMBER (4-byte)
PatientId (AK1)	Unique patient identifier number assigned by the user when the information for this patient is entered.	VDT_PATIENTID nvarchar(25)
PatientId2 (IE1)	Non-unique identifier that can be assigned to a patient.	VDT_PATIENTID nvarchar(25)
PatientUID (AK2)	Unique constraint Patient ID that will not be changed on upgrade, conversion, or deletion of the patient record.	VDT_UID nvarchar(64)
PatientType	Type of patient. Values are: • ActivePatient • ArchivedPatient • DeletedPatient	VDT_TABLENAME nvarchar(50)
CreationDate	Date and time of creation of this patient, or the date when this patient's data was imported.	VDT_DATETIMESTAMP Datetime
CreationUserName	User who created this patient record.	VDT_USERNAME nvarchar(50)
SSN	Patient's social security number.	VDT_STRING64 nvarchar(64)
FirstName	First name of patient.	VDT_NAME nvarchar(64)
MiddleName (IE1)	Middle name of patient.	VDT_NAME nvarchar(64)

Data Dictionary(MOSAIQ)




ELEKTA

MPAC Data Dictionary Use and Non-Disclosure Agreement

Queries and SQL

- SQL (SEQUEL): Structured English Query Language
- The standard means of retrieving data from a relational database
- Is very powerful but can be used simply
- The standard form is
  - Select <fields> from <table> where <criteria>;
- Queries can returned sorted results
  - Select <fields> from <table> where <criteria> ORDER BY <field>;

Simple Query Example MOSAIQ

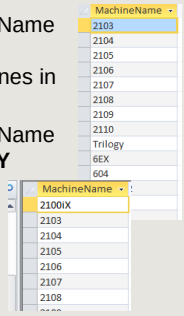
Gets a Sim Note based on MRN.

```
SELECT  "Create_DtTm",  "Sim_Name",  "Notes"
FROM
  "vw_MosaiQSimulate"
WHERE
  "Pat_ID1" = (SELECT "Pat_Id1" FROM
"vw_MosaiQIdent" WHERE IDA = 'escaped MRN value')
```

Michael Kantor

Example Query (Sun Nuclear Atlas) with and with-out sorting

- SELECT dbo_Machine.MachineName FROM dbo_Machine returns an unsorted list of machines in the database
- SELECT dbo_Machine.MachineName FROM dbo_Machine **ORDER BY dbo_Machine.MachineName;** returns a sorted list

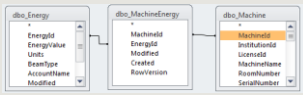


Queries and SQL using Joins

- Queries can span multiple tables using Joins
  - Select <fields> from <table> Join <table2> on <common field> where <criteria>;
  - These tables can span different databases
- Joins can be Inner or Outer Joins
  - Inner joins (default join) only returns data that has matching rows in both tables
  - Outer joins will give data that exists in either table
- Example if you have a table of equipment and one of calibrations and inner join on equipment will only give equipment with calibrations an out join will give all equipment even if no calibration exists

Example Join Query (Sun Nuclear Atlas)

- If we want to know what energies each machine has we need to use joins since that information spans 3 tables
- SELECT dbo_Machine.MachineName, dbo_Energy.EnergyValue, dbo_Energy.Units FROM (dbo_Machine INNER JOIN dbo_MachineEnergy ON dbo_Machine.MachineId = dbo_MachineEnergy.MachineId) **INNER JOIN dbo_Energy ON dbo_MachineEnergy.EnergyId = dbo_Energy.EnergyId** ORDER BY dbo_Machine.MachineName;



MachineName	EnergyValue	Units
2100iX	18	MV
2100iX	16	MeV
2100iX	20	MeV
2100iX	12	MeV
2100iX	6	MV
2100iX	9	MeV
2100iX	6	MeV
2103	6	MV
2103	18	MV

Queries and SQL using aggregate functions

- Queries can return summary data (SQL aggregation functions)
  - Count, Sum, Max, Min, Avg, etc
  - Example: Select avg(<field A>) from <table> where <criteria>
    Gives the average value of <field A>
- Generally used with the "GROUP BY" clause to define what set of data is being aggregated
  - Example: Select <field B>, avg(<field A>) from <table> where <criteria> group by <field B>
    Gives the average value of <field A> for each <field B>

Simple aggregate query example: Pinnacle (also showing use of the built in tools)

- Uses the built in PostgreSQL interactive terminal to find the number of patients in the active database

```
psql -d p3rtp -h localhost -U lpuser
Welcome to psql 8.3.5, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
p3rtp=> SELECT COUNT("patientid") FROM "patient";
 count
 1193
```

Michael Kantor

Example aggregate Query (Sun Nuclear Atlas)

- We can use aggregate functions if we want to know how many of each energy we have across all machines:
- SELECT dbo_Energy.EnergyValue, **Count(dbo_Energy.EnergyValue)** AS CountOfEnergyValue FROM (dbo_Machine INNER JOIN dbo_MachineEnergy ON dbo_Machine.MachineId = dbo_MachineEnergy.MachineId) INNER JOIN dbo_Energy ON dbo_MachineEnergy.EnergyId = dbo_Energy.EnergyId **GROUP BY dbo_Energy.EnergyValue;**

EnergyValue	CountOfEnergyValue
2	1
4	3
5	1
6	64
7	3
8	1
9	30
10	7
11	2
12	29

Example aggregate query using a derived value (Sun Nuclear Atlas)

- Since Energy = 6 can be 6 MV or 6 Mev to further specify we need to build a derived field to sort on that includes both the energy value and the units
- We can query and sort based on this derived value: **[EnergyValue] & [Units]**

SELECT [dbo_Energy]![EnergyValue] & [dbo_Energy]![Units] AS Expr1, Count(dbo_Energy.EnergyValue) AS CountOfEnergyValue FROM (dbo_Machine INNER JOIN dbo_MachineEnergy ON dbo_Machine.MachineId = dbo_MachineEnergy.MachineId) INNER JOIN dbo_Energy ON dbo_MachineEnergy.EnergyId = dbo_Energy.EnergyId **GROUP BY [dbo_Energy]![EnergyValue] & [dbo_Energy]![Units];**

Expr1	CountOfEnergyValue
10MeV	1
10MV	6
11MeV	2
12MeV	29
15MeV	3
15MV	6
16MeV	28
18MV	23
20MeV	26
2MV	1
4MeV	3
5MeV	1

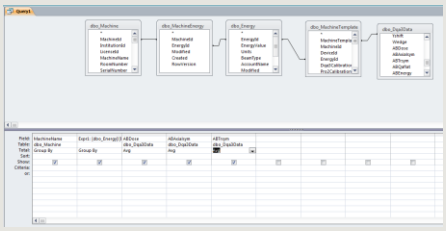
Query Building tools

- Building more useful queries like “what is the average output and symmetry for each machine by energy” can be very complicated
  - SELECT dbo_Machine.MachineName, [dbo_Energy]![EnergyValue] & [dbo_Energy]![Units] AS Expr1, Avg(dbo_Dqa3Data.ABDose) AS AvgOfABDose, Avg(dbo_Dqa3Data.ABAxialsym) AS AvgOfABAxialsym, Avg(dbo_Dqa3Data.ABTrsym) AS AvgOfABTrsym FROM (((dbo_Machine INNER JOIN dbo_MachineEnergy ON dbo_Machine.MachineId = dbo_MachineEnergy.MachineId) INNER JOIN dbo_Energy ON dbo_MachineEnergy.EnergyId = dbo_Energy.EnergyId) INNER JOIN dbo_MachineTemplate ON dbo_Energy.EnergyId = dbo_MachineTemplate.EnergyId) INNER JOIN dbo_Dqa3Data ON dbo_MachineTemplate.MachineTemplateId = dbo_Dqa3Data.MachineTemplateId GROUP BY dbo_Machine.MachineName, [dbo_Energy]![EnergyValue] & [dbo_Energy]![Units];

MachineName	Expr1	AvgOfABDose	AvgOfABAxialsym	AvgOfABTrsym
2100iX	12MeV	99.7223249300841	-0.533629961683187	0.332782401415182
2100iX	16MeV	99.3277260832826	-0.260144505217171	0.410471441043903
2100iX	18MV	101.851307208846	-3.51383400726538	18.617367738293
2100iX	20MeV	99.578883268125	-0.230178432177204	0.358647870454756
2100iX	6MeV	98.8171913501059	1.1735407609238	0.716443810496579
2100iX	6MV	100.413902185142	4.39687339268117	-20.2291871191576
2100iX	9MeV	99.1991062939368	-6.918420822848 48E-02	0.487500558486822
2103	12MeV	99.7223249300841	-0.533629961683187	0.332782401415182
2103	16MeV	99.3277260832826	-0.260144505217171	0.410471441043903

Query Building tools

- There are a large number of query building tools that have GUIs that can help you quickly build complicated queries



GUI based Query builders

- There are multiple sets off software both commercial and open source that will provide a framework for helping you build queries.
  - Many will automatically provide joins based on the database scheme
  - The will write SQL than can be manually edited, if needed, and/or copied into code for automated querying
- Examples
  - Microsoft SQL management studio
  - MS Access (used for some of the examples in this work)
  - pgAdmin (PostgreSQL)
  - Many others (check google)

Queries that change the data

- Some query keywords can be used to change the data in the database
  - Insert: adds rows
  - Update: modifies rows
  - Delete: removes rows
  - Merge: combines rows
- These can be very powerful but:
  - Can unexpectedly modify/remove large amounts of data
  - Should not be used on clinical databases (ARIA, MOSAIQ)

Queries that change the data

No examples because you shouldn't be doing this

Programing and database access

- Many programing languages have built in database support (libraries)
  - Microsoft languages
    - Support all databases directly or via ODBC
    - VBA embedded in MS Access
    - VBA from other packages (Word, Excel, etc) can be used to post data to a database or write reports
  - Python
    - Python-sql
- Some vendors have APIs to allow data access in a controlled manner
  - Varian

Varian API for data access

```
// getting all treatment plan's data under "C2" course in selected patient
var allPlansInfo = from Course c in ThePatient.Courses
                   where c.Id == "C2"
                    select new
                    { plans = c.PlanSetups,
                      Course = c, };

//query finds the first PTV structure:

Structure target = (from s in StructureSet.Structures
                    where s.DicomType == "PTV"
                    select s).FirstOrDefault();
```

Amy Liu

Queries that return a large amount of data

- Poorly written (or thought out) queries can adversely affect the performance of the database server the one that is also running your clinic in the case of MOSAIQ or ARIA)
  - Queries should have a "LIMIT" clause during testing if supported by your SQL server
  - Queries should be tested on non-clinical systems first, if possible.
    - If no development system exists consider testing during off-hours
  - For relatively small databases consider making a backup of the database and run queries against the backup during debugging

Views, Triggers, and Stored procedures

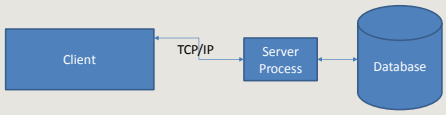
- Most SQL servers support the serval non-data objects in the database used for efficient data access and database integrity:
  - Views: Predefined queries to simplify the users access to the database
  - Stored Procedures: Pre-compiled SQL statements
    - Run faster than normal queries
  - Triggers: Procedures designed to run automatically based on other events
    - Can be used for data integrity checks
    - Often used to create audit logs

Reports

- Forms with areas filled in by queries embedded within the form
- Often the only "official" way to get summary data out of many systems
- Are useful when the same data is to be queried and presented multiple times
  - Physics weekly check list from MOSAIQ/ARIA
  - Billing documents
  - Treatment summaries
- Some database systems have integrated report generators
  - MS Access
  - Microsoft SQL Server Reporting Services (SRSS)
- There are many 3rd party report generators
  - Crystal Reports is a combination query builder and report generated used as the OEM report generator for many systems

The Client Server Database Model

- The database runs as it's own process either on the local computer or on a distant computer
- Applications send transactions to the database
- Transactions maybe processed via an intermediate layer (driver)
  - Microsoft ODBC driver allows many different types of databases to talk to one another



Advantages of the client server model

- Separates database development from application development
- Client applications are independent from physical location of data
- Client systems can be optimized for display and user interface while database server can be optimized for performance
- Reduce traffic on the network as only processed data is transferred
- Can handle concurrent access by multiple users (better than a shared file)

Common Database systems

- Oracle: High performance "main frame" database
- Microsoft SQL: High performance clustered system or local "lite versions" available
- PostgeSQL: Scalable open source database
- MySQL: Scalable open source database
  - RedCap (front end on MySQL) for "building and managing online surveys and databases"
- MS Access: great general purpose database that hides much of coding.
- SQLite: open source database used as an embedded database in many other applications
- Many many more (just google open source SQL)

Databases in Radiotherapy

- MOSAIQ → Microsoft SQL
- Aria → Microsoft SQL
- Sun Nuclear Atlas → Microsoft SQL
- Teramedica (FUJI) Evercore → Microsoft SQL, Oracle
- Radcalc → SQLite
- Pinnacle (for a limited amount of the data) → PostgreSQL
- Varian RPM → mdb file(MS Access)

Note the SQL formalism is general enough that many system can support more than one type of database

Backup

- Database systems have backup utilities that the end user should be able to use
  - Simple systems may be just a file backup
  - Other systems have backup and restore functions within the SQL server workspace
  - Many systems exist for real-time (or near-time) backup between database clusters
  - Many enterprise systems allow “rewind” back to a state at an earlier time
- Work with your vendor to understand what backup systems they support and how to best implement backups in your environment

Summary

- Many of the Systems used in radiotherapy have back-end open standards databases
- If we understand some database basics
  - We can retrieve data more systematically and efficiently than the tools provided by the GUIs
    - Including some “hidden” data not available in the GUI
  - We can build automated system for reporting and QA
  - We can properly ensure data available and security

THE UNIVERSITY OF TEXAS
MD Anderson Cancer Center
Making Cancer History®

- Great thanks to Michael Kantor for many of the examples in this talk as well as a great expansion of the content
- Thank you to Amy Liu for Eclipse API examples
- Thank you to the audience for, I assume, staying awake while I spoke about SQL