Managing Research Data in SQL Server

After VINCI creates your Research Database and provisions your source data, you are free to use those resources in service of your research. But how can you best make use of those resources? This training will cover data management techniques and tips for:

- Designing/creating tables, views and schemas.
- Using indices and compression.
- Avoiding permission issues.
- Monitoring resource usage.
- Troubleshooting common issues.

In This Presentation

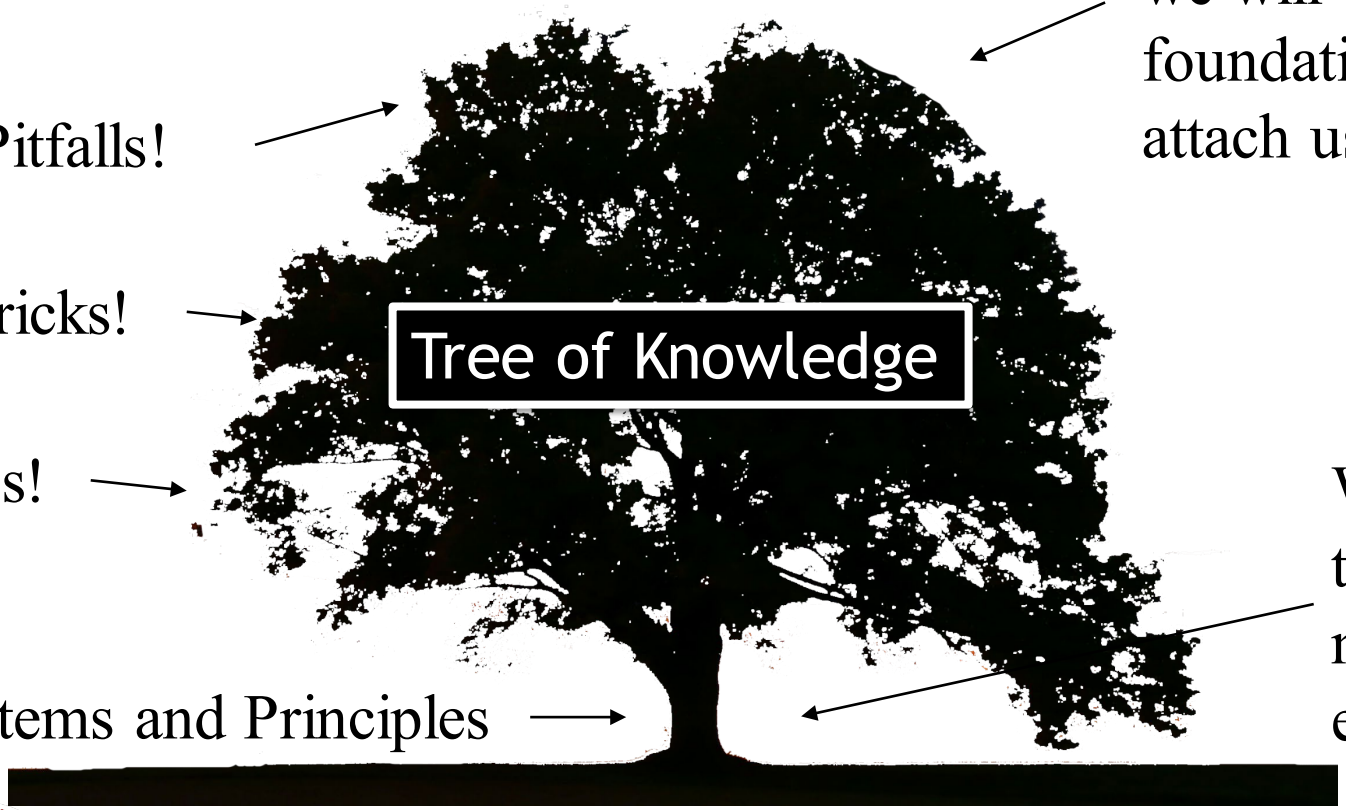
- I assume you…
  - …Are familiar with the basic architecture of VINCI-provisioned databases
    - If not, check out last month's cyberseminar: Managing a Research Project Using VINCI Data (va.gov)
  - …Can access and use:
    - VINCI Workspace
      - If not: VINCI_Workspace_User_Guide.pdf (va.gov)
    - SQL Server Management Studio
      - If not: VINCI_Database_User_Guide.pdf (va.gov)

With Freedom Comes Responsibility

"Freedom makes a huge requirement of every human being. With freedom comes responsibility. For the person who is unwilling to grow up, the person who does not want to carry his own weight, this is a frightening prospect."
-Eleanor Roosevelt

- Once your research database has been created and provisioned with source data, it is no longer managed by VINCI – that responsibility falls to the researchers.
- Research databases make use of shared resources:
  - DB space (40 GB limit by default)
  - Processing (long running and/or excessive queries will be killed)
  - TempDB space (shared at the server level, can also lead to killed queries)

The Plan



When we arrive here, we will have a solid foundation to which to attach useful details

Pitfalls!

Tricks!

Tips!

Systems and Principles

We will start at the bottom with nontechnical explanations

Database Design

- What do you start with?
  - [Src] schema
    - Cohort tables
    - CDW fact views, pre-filtered to your cohort and time frame
  - Access to CDW Dimensions.
  - ~40 GB of space.
  - Questions!

SQL

- VINCI provisions research data in SQL databases
  - Someone on your team needs some SQL skill to work with that data
  - Help is available:
    - VINCI University (va.gov)
      - And especially VINCI Training & Office Hour (va.gov)
    - Data Services & Field Support (sharepoint.com)
- I will assume you know SQL in this presentation

But I'm Not a SQL Expert!

- Unfortunately, **someone** on your team will need some SQL skill
- Even if you plan to access your data from an analysis platform like SAS or R, you still need to know SQL because **passthroughs** are recommended
- For more info on accessing your SQL DBs in SAS: VINCI SAS/Grid (va.gov)
  - More tips in the appendix

Let's Create Some Tables!

- Design backwards, from your goal.
- Think about (de-)normalization.
- Remember "one row per something"
  - Know what the primary key (PK) is for each table you create
- You can create dimensions and fact tables
  - Dimensions are "lookups" of reference information
  - Fact tables contain records of healthcare activities

What Is Normalization?

- Normalization is an organizing principle for databases to reduce redundancy and improve integrity.
- In a nutshell:
  - Use more tables. Each table should be about one thing only – "one row per something". No other tables should contain details about that one thing.
  - Use more rows. Don't widen rows with additional columns for more of the same kind of thing.
  - Never put multiple items in the same cell.
- More details in the appendix

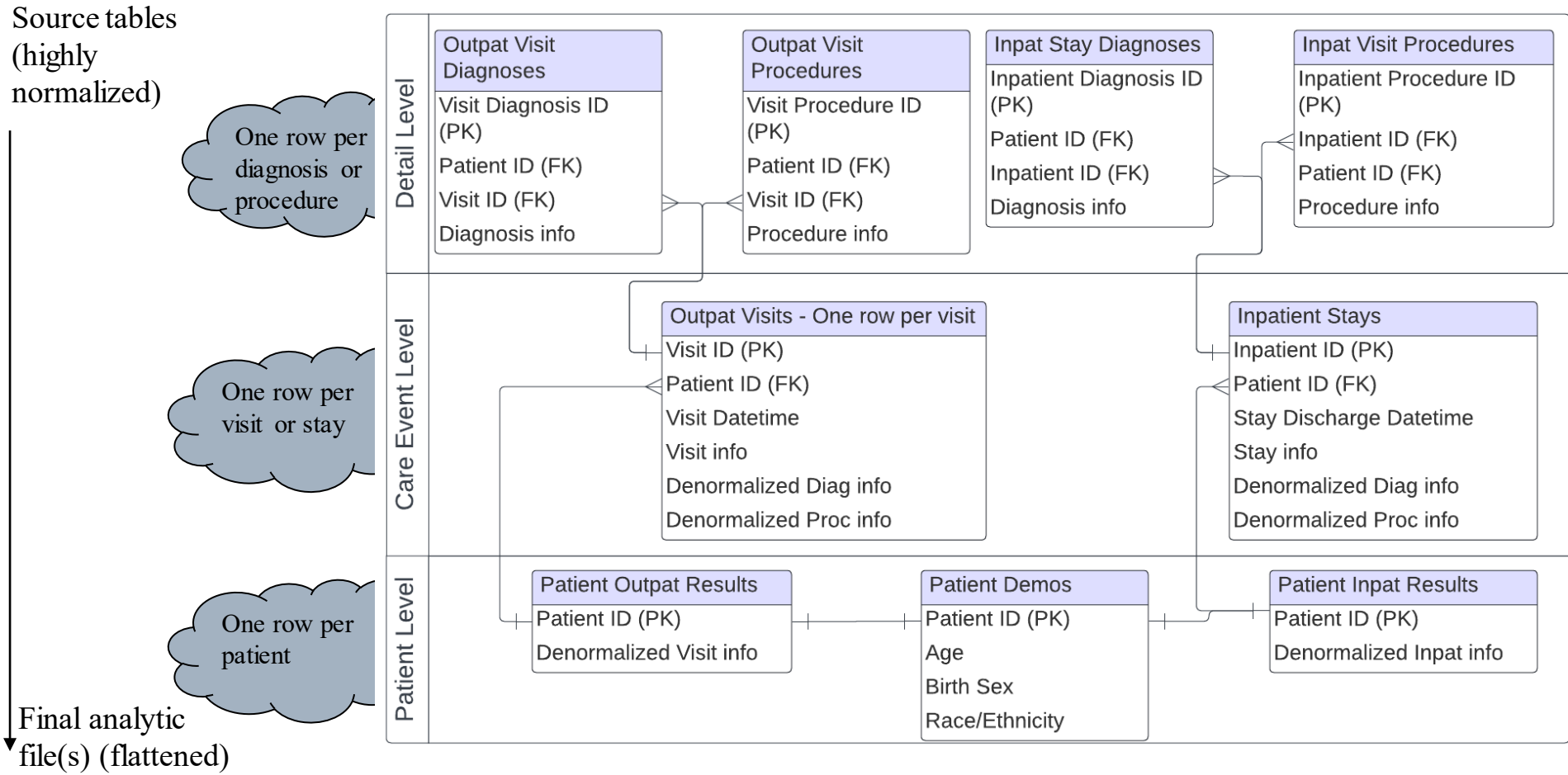
Normalization Principles

- Why normalize?
  - More reusability and flexibility.
  - Better integrity.
  - Usually smaller space footprint.
  - Usually faster queries (with good indexing and query best practices).
- Why denormalize?
  - Analysis datasets are typically “flat files” (“Flattened” = Denormalized)
  - You probably want a single large table with one row per unit of analysis and columns for all variables of interest
    - Demographics or patient attribute table – one row per patient
    - Facility metrics – one row per healthcare system

Table Design Tips

- Your SQL end product should be at the level of the unit of analysis or one level more detailed
  - E.g. patient-level – save a flat file with one row per patient
- Work towards your big flat file with multiple small flat files, e.g.:
  - One row per patient with demographics
  - One row per patient with procedure flags
  - One row per patient with wait time metrics
  - Join all of the above to create the final flat file

Denormalize Our Way to a Flat File



Bad Join Example

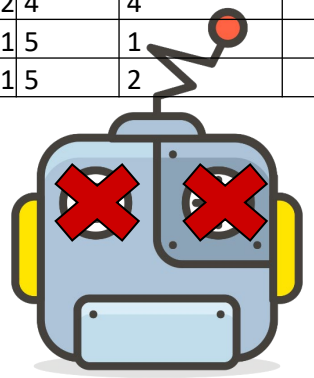
"I want to connect patient visits to patient stays. I see a Patient ID in both tables. I'll simply join on that – easy peasy!"

```
select *
from #visits as v
join #stays as s
    on v.PatientICN = s.PatientICN
```

#visits - one row per visit			
PatientICN	VisitSID	VisitDateTime	Other info
1	1		
1	2		
2	3		
2	4		
1	5		

#stays - one row per stay			
PatientICN	InpatientSID	DischargeDateTime	Other info
1	1		
1	2		
2	3		
2	4		

Result - one row per ???			
PatientICN	VisitSID	InpatientSID	Other cols
1	1	1	
1	1	2	
1	2	1	
1	2	2	
2	3	3	
2	3	4	
2	4	3	
2	4	4	
1	5	1	
1	5	2	



This is a many-to-many join. Do not do this.

Good Joins

- At least one side needs to be a primary key
  - PK = PK join is one-to-one
  - PK = FK join is one-to-many
  - FK = FK is many-to-many
- You can use metadata to help with this in the CDW, but you're on your own with tables you've created

Denormalization Tools Example 1

- For example, if we need to go from one row per diagnosis to one row per visit:
  - Filters
    - Pick the diagnosis listed as “primary”
  - Hierarchy
    - Pick a diagnosis based on S06.0X6A > S06.0X5A > S06.0X4A > …
  - Categorization
    - Assign the visit to a category (like on slide 20)
  - Flags
    - Set one or more flags per visit (like on slide 20)
  - Aggregation/calculation
    - Show how many distinct diagnoses appear per visit
  - Pivoting
    - Show all diagnoses using more columns
  - Combinations of the above

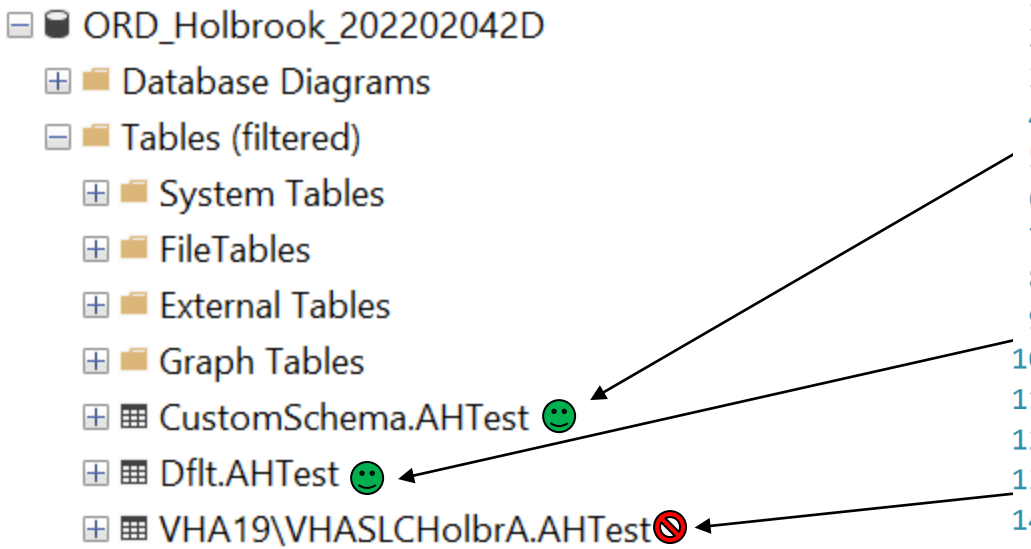
Denormalization Tools Example 2

- For example, if we need to go from one row per visit to one row per patient:
  - Filters
    - Pick the visit that fits specific criteria (e.g. a certain diagnosis, procedure, clinic…)
  - Hierarchy
    - Pick the visit that happened in a medical center instead of a CBOC
  - Categorization
    - Assign the patient to a category (like on slide 20)
  - Flags
    - Set one or more flags per patient (like on slide 20)
  - Aggregation/calculation
    - Show the first and last visits per patient
  - Combinations of the above

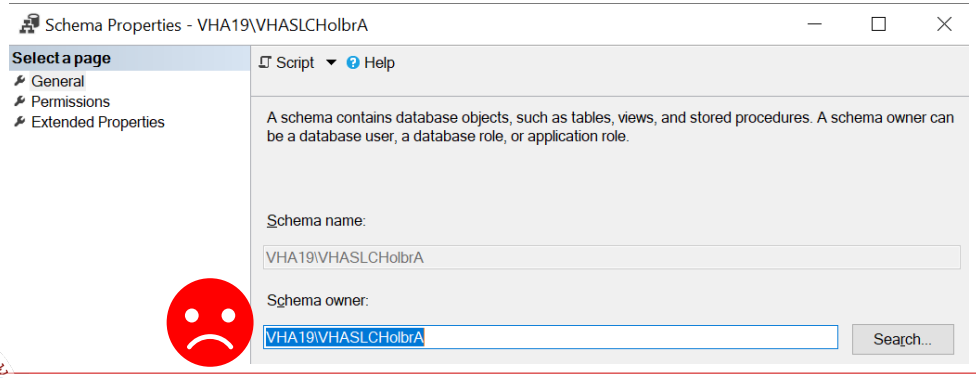
Denormalization Takeaways

- Keep your end goal in mind.
- Denormalize purposefully!
  - Do you have an exact flat file in mind as your goal? Will it ever need to change?

A Table Creation Pitfall




```
1  use ORD_Holbrook_202202042D
2
3  select *
4  into CustomSchema.AHTest
5  from cdwwork.dim.sta3n
6  --this one throws an error
7  --if you haven't already created the schema
8
9  select *
10 into dflt.AHTest
11 from cdwwork.dim.sta3n
12
13 select *
14 into AHTest
15 from cdwwork.dim.sta3n
16 --oops! You probably wanted a temp table
```

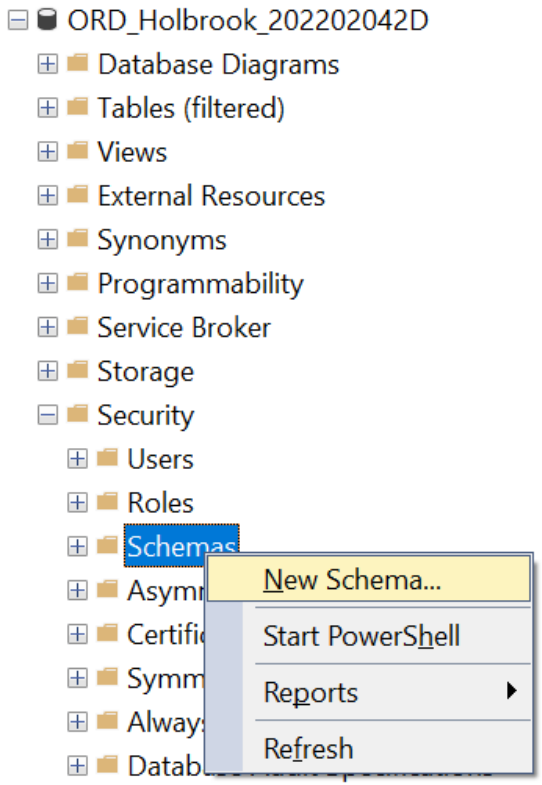


Create Schemas

- It's not necessary. [dflt] is fine to use, but watch out. [dflt] is not the default!
  - If you don't specify a schema, SQL will create one with you as the owner. This is bad.
- You can use custom schemas.
  - Custom schemas with descriptive names can be useful.
    - For example, instead of dflt.Cohort1TBIVisits:
      - Cohort1.TBIVisits
      - TBI.Cohort1Visits

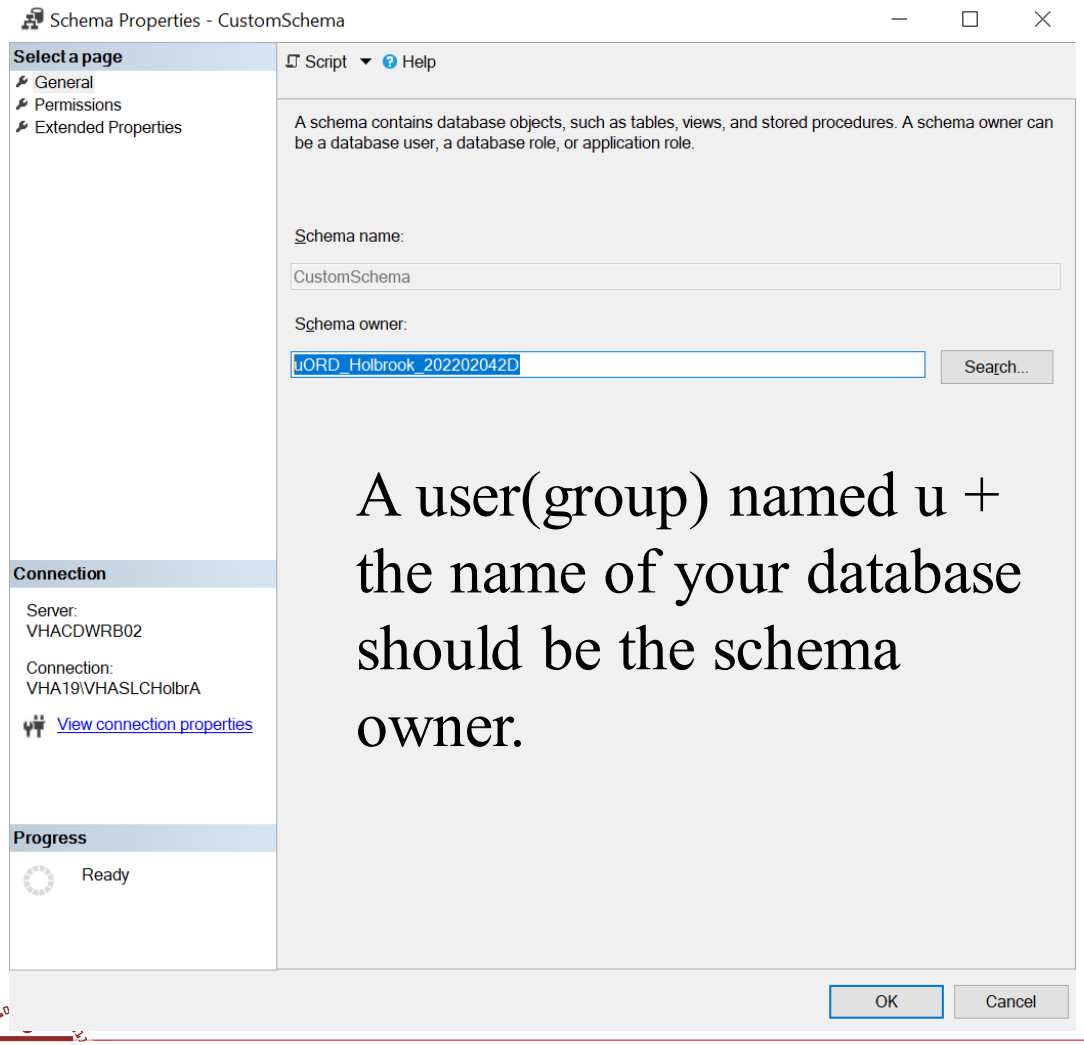
How to Create a Schema

In Object Explorer: right click on the Security folder or the Schemas subfolder in your DB:

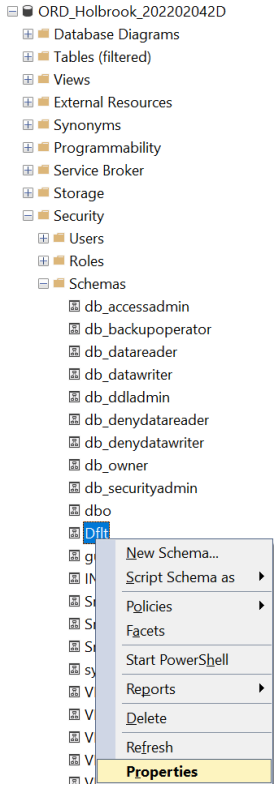


In SQL: `create schema CustomSchema authorization uORD_Holbrook_202202042D`

Correct Schema Ownership



If you can't remember this, just check the owner of your Dflt schema:



Create Dimensions

- You have access to CDW dimensions.
  - But they contain lots of stuff you (probably) don't need…
  - AND they don't contain stuff you may very well need!

Why not use CDW Dimensions Directly?

```
select top 1000 *
from CDWWork.dim.ICD10 code
join CDWWork.dim.ICD10DescriptionVersion descr
    on code.ICD10SID = descr.ICD10SID
where ICD10Code like 'f10%'
--25 columns
```

Just to get the description!

Results | Messages

	ICD10SID	ICDIEN	Sta3n	ICD10Code	DRGIdentifier	DRGConditionComorbidityExclusionSID	ReproductiveSystemMDC13Flag	MultipleSignificantTraumaMDC24	HIVMDC25	ExpandedICDFlag	UnacceptableAsPrimaryFlag	ICD10POAExemptFlag
1	800514432	503622	640	F10.181	NULL	800027338	NULL	NULL	NULL	NULL	NULL	N
2	800514440	503654	640	F10.97	tC	-1	NULL	NULL	NULL	NULL	NULL	N
3	800514515	503649	640	F10.94	tC	800027338	NULL	NULL	NULL	NULL	NULL	N
4	800514801	503647	640	F10.921	tC	800027338	NULL	NULL	NULL	NULL	NULL	N
5	800514913	503623	640	F10.182	NULL	-1	NULL	NULL	NULL	NULL	NULL	N
6	800515212	503617	640	F10.14	NULL	800027338	NULL	NULL	NULL	NULL	NULL	N
7	800516745	503653	640	F10.96	tC	-1	NULL	NULL	NULL	NULL	NULL	N
8	800516815	503648	640	F10.929	C	-1	NULL	NULL	NULL	NULL	NULL	N
9	800517345	503615	640	F10.121	tC	800027338	NULL	NULL	NULL	NULL	NULL	N
10	800517425	503645	640	F10.29	tC	800027338	NULL	NULL	NULL	NULL	NULL	N
11	800517429	503637	640	F10.251	tC	800027338	NULL	NULL	NULL	NULL	NULL	N
12	800518022	503640	640	F10.27	tC	800027338	NULL	NULL	NULL	NULL	NULL	N
13	800519454	503627	640	F10.21	t	-1	NULL	NULL	NULL	NULL	NULL	N
14	800519966	503658	640	F10.988	NULL	800027338	NULL	NULL	NULL	NULL	NULL	N
15	800520353	503643	640	F10.282	NULL	-1	NULL	NULL	NULL	NULL	NULL	N
16	800520647	503634	640	F10.239	NULL	800027338	NULL	NULL	NULL	NULL	NULL	N
17	800521303	503650	640	F10.950	NULL	-1	NULL	NULL	NULL	NULL	NULL	N

--25 columns

Alternatively

Results | Messages

	ICD10SID	ICD10Code	PTSD_EBPsy	Depression_EBPsy	AUD_EBPharm	OUD_EBPharm	Depression_EBPharm	Sta3n
1	800514001	F10.259	0	0	1	0	0	640
2	800514014	F10.281	0	0	1	0	0	640
3	800514061	F10.150	0	0	1	0	0	640
4	800514092	F11.159	0	0	0	1	0	640
5	800514107	F11.221	0	0	0	1	0	640
6	800514115	F10.221	0	0	1	0	0	640
7	800514166	F10.231	0	0	1	0	0	640

Results | Messages

	cptcode	sta3n	cptsid	Encounter Type
1	90847	358	800012040	Family Therapy
2	90853	358	800012042	Group Therapy
3	99205	358	800013830	Evaluation
4	99215	358	800013835	Med Mgmt
5	97150	358	800014752	Art/Recreaction Therapy

Less greedy AND more useful!

Tips For Creating a Dimension

- Remember “one row per something” and avoid many-many joins
  - In almost all cases, you want one row per distinct SID. Choose the SID that appears as a foreign key in the fact tables you will pull from.
    - One row per diagnosis – ICD10SID or ICD9SID
    - One row per procedure – CPTSID or ICD10ProcedureSID
    - Etc.
- You should index the SID column that you will use for joins (more on this later). That SID column needs to be unique!

Dimensions are Small and Easy

- If you are learning SQL as you go, start with custom dimensions BEFORE you tackle fact tables. Please.

Create Fact tables

- Denormalize purposefully.
- Use an index.
- Use compression.
- If you're worried about space:
  - You weren't greedy, right?
  - You used compression, right?
  - Consider a view (more on this later).

Index

- Types:
  - Clustered (should always have this…)
  - Columnstore (…or this, but not both)
  - Nonclustered (sometimes helpful, trade space for speed)
  - Spatial, XML, full text (generally don't need these)
- Why bother?
  - Space
  - Efficiency
  - Mindfulness

Clustered Index

- Physically orders the rows by your chosen column.
- Great for seeking.
- Must specify compression.
- Mutually exclusive with columnstore index.
- An object can only have one clustered index.
- For example:
  - A dictionary has a clustered index on [Word] (asc)
- In general, use clustered index on the PK SID for DIM tables.

Clustered Columnstore Index (CCI)

- Physically stores columns instead of rows.
- Great for scanning.
- Includes (very effective) compression.
- Mutually exclusive with clustered index.
- In general, use CCI for fact tables if they're > 1M rows and a clustered index on your PK otherwise.

Nonclustered Index

- Like the index in the back of a book:
  - Increases the object's size.
  - Useful for columns you'll search on.
- Not usually necessary.
  - Columnstore indices already include all columns

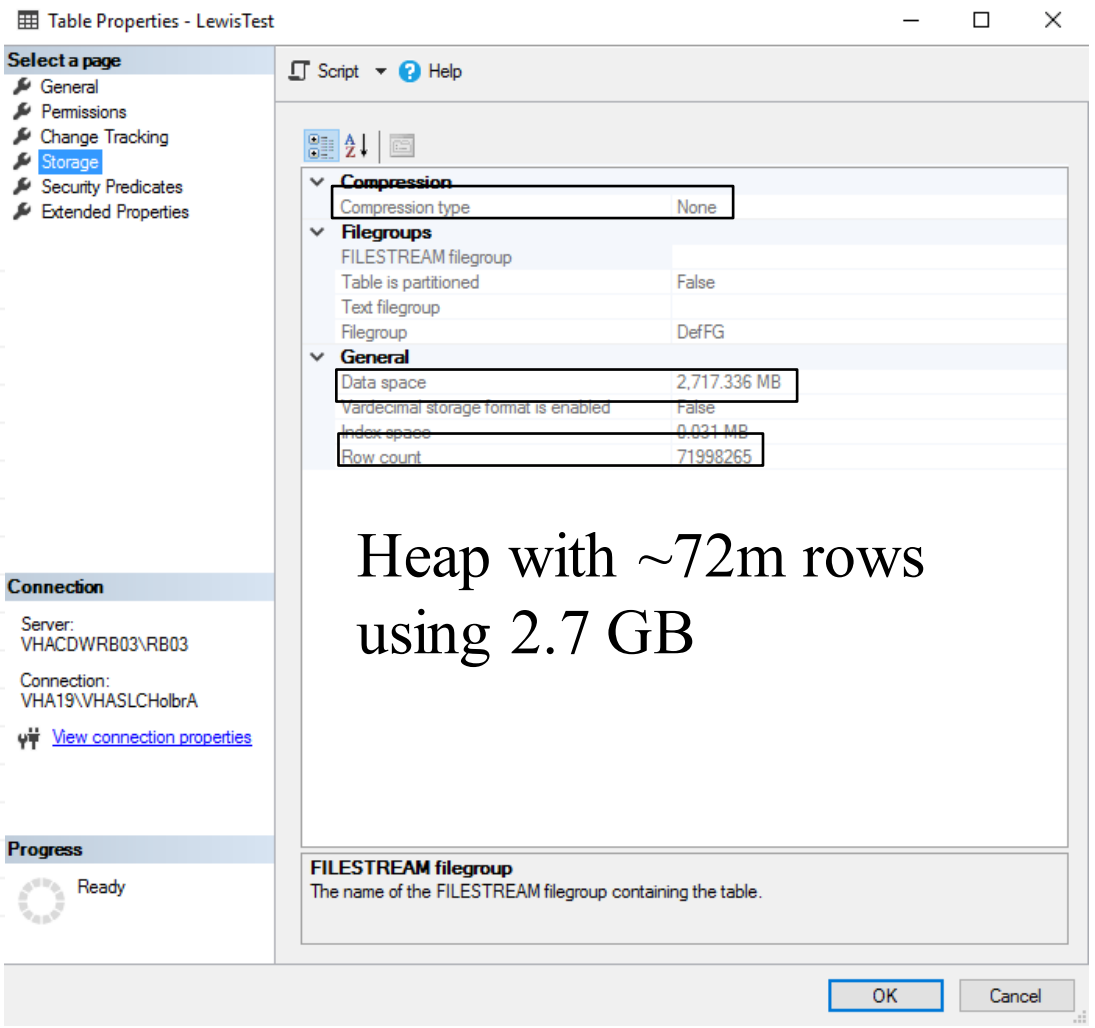
Compression

- Row
  - Shrinks columns.
  - Included in page compression algorithm.
  - Don't bother.
- Page
  - Looks for patterns and zeroes.
  - Included (sort of) in Columnstore compression algorithm.
- Columnstore
  - Is both an index and a compression algorithm.
  - Stores columns, compresses patterns heavily.
  - This is the best compression algorithm, but:
    - You only get the full benefit with > 1 million rows.
    - Makes inserts and updates slow and costly.
- More details in the appendix

Compression Example

```
select top 1000 *
from ah.LewisTest
```

	PatientICN	DOB	VisitYear	Sta3n	CY_Age
1	1004402027	1948-08-06 00:00:00	2017	436	68
2	1008384374	1948-08-23 00:00:00	2017	580	68
3	1013115908	1948-11-24 00:00:00	2017	671	68
4	1017161137	1948-09-10 00:00:00	2017	654	68
5	1001896170	1948-09-03 00:00:00	2017	598	68
6	1001143460	1948-02-17 00:00:00	2017	626	68
7	1016657004	1948-08-22 00:00:00	2017	580	68
8	1019340928	1948-11-25 00:00:00	2017	652	68
9	1015662870	1948-03-22 00:00:00	2017	573	68
10	1001029189	1948-02-17 00:00:00	2017	679	68
11	1000832922	1948-07-14 00:00:00	2017	662	68
12	1005503068	1948-10-24 00:00:00	2017	402	68
13	1017654421	1948-10-29 00:00:00	2017	540	68
14	1010318284	1948-04-29 00:00:00	2017	580	68
15	1019335777	1948-03-04 00:00:00	2017	676	68
16	1017014350	1948-08-02 00:00:00	2017	544	68
17	1010122884	1948-07-02 00:00:00	2017	564	68



Heap with ~72m rows using 2.7 GB

Wait, a Heap?

- Yes, a heap.

"Freedom makes a huge requirement of every human being. With freedom comes responsibility. For the person who is unwilling to grow up, the person who does not want to carry his own weight, this is a frightening prospect."
-Eleanor Roosevelt

Compression Example

FACT before: heap with ~72m rows using 2.7 GB

- After:
  - Row: 2.3 GB
  - Page: 1.7 GB
  - Columnstore: 1.1 GB

DIM before: heap with 100k rows using 3.8 MB

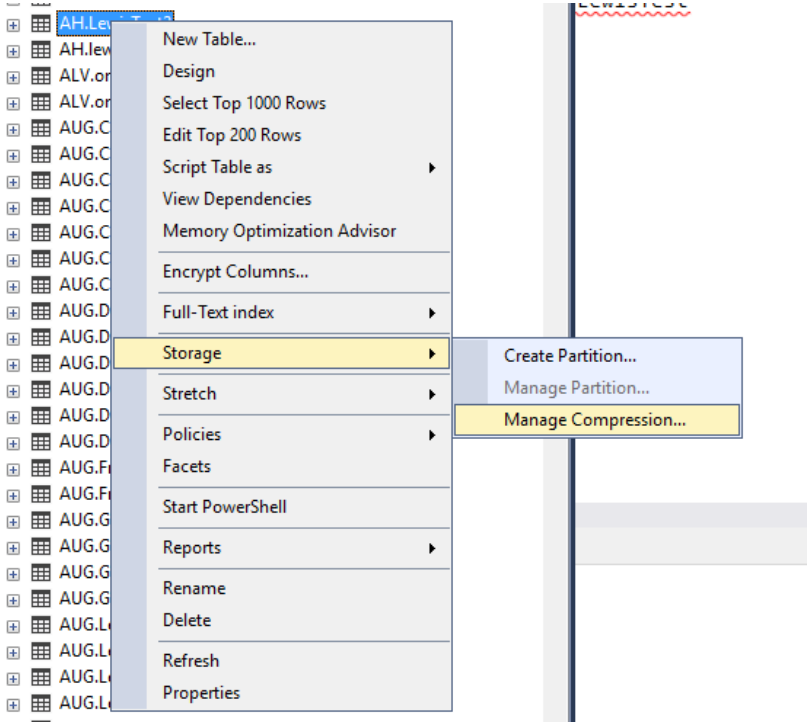
- After:
  - Row: 3.2 MB
  - Page: 2.5 MB
  - Columnstore: 2.5 MB

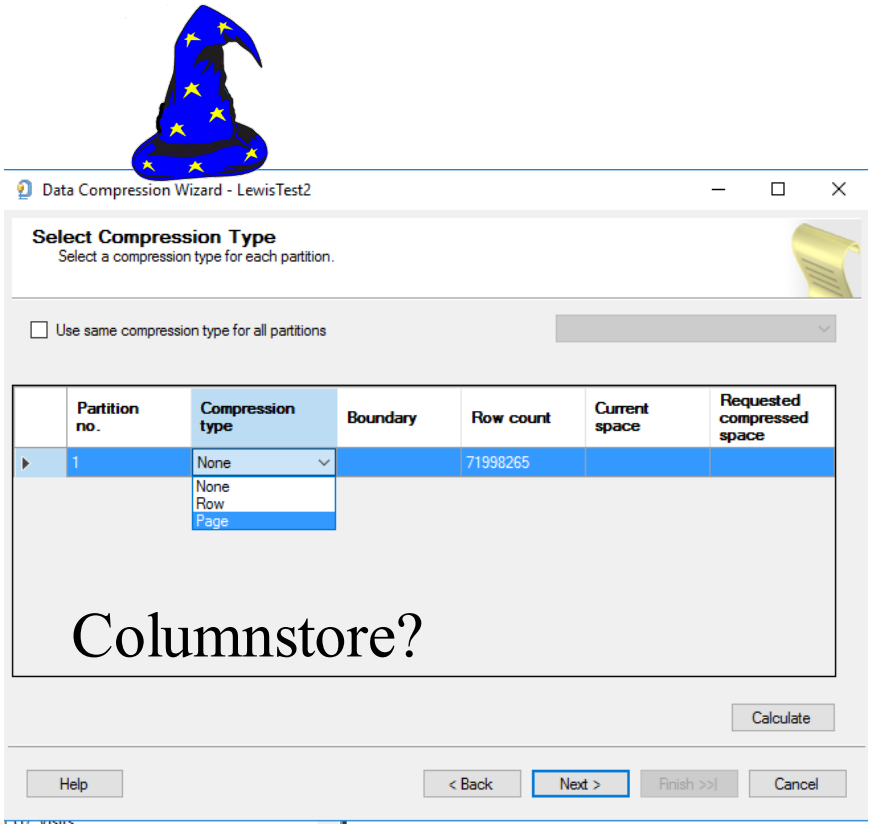
How About Speed?

- For selecting and joining, performance depends on how the table will be used.
  - Scan: Looking at lots of rows at a time, e.g. when aggregating.
    - CCI should be faster.
  - Seek: Looking at individual rows at a time, e.g. when looking up a value.
    - Clustered index should be faster.

How to Compress

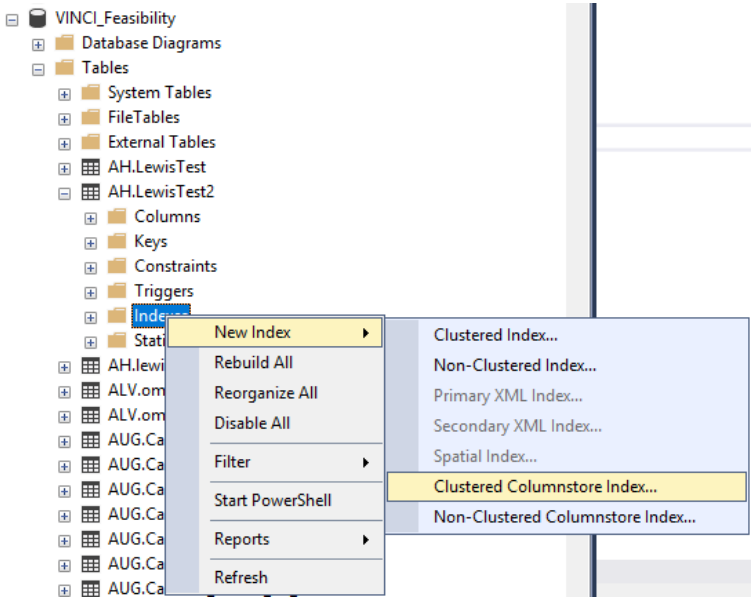
You can right-click on a table in the object explorer:





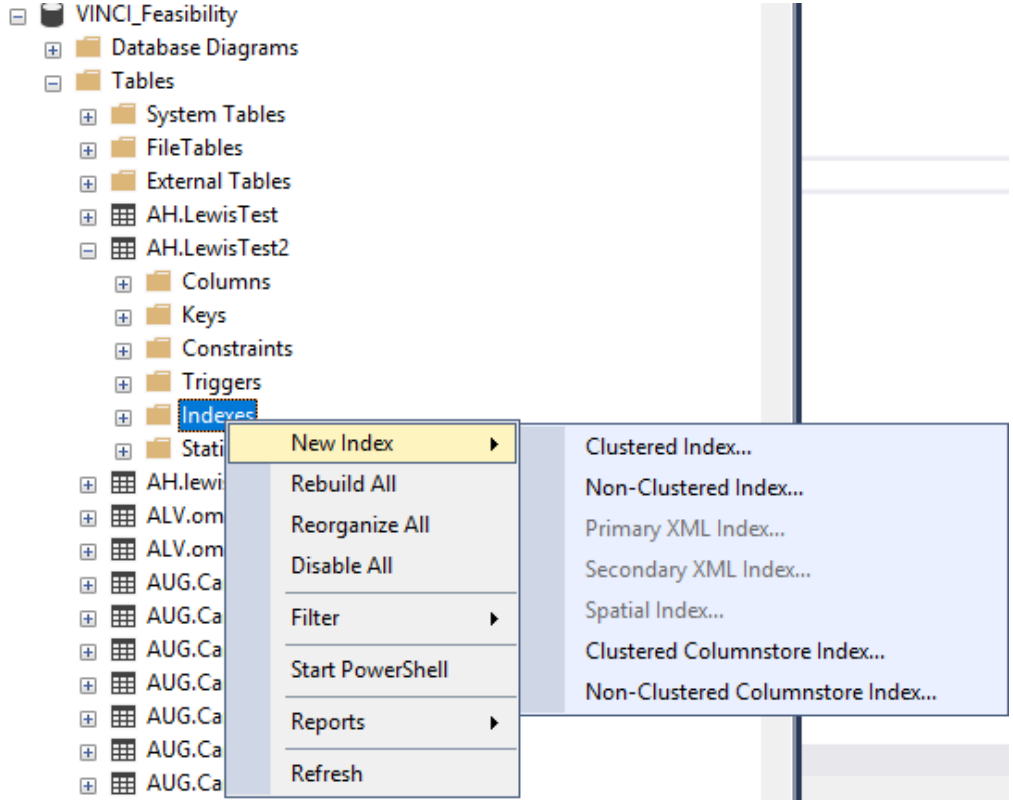
How to Compress

- For now, columnstore compression can't be implemented via the compression "wizard"
  - You can still do it in object explorer by browsing to the Index folder.



How to Index

You can expand a table in the object explorer and right-click on the Indexes folder:



How to Index/Compress

Or, in the script, implement compression as you create the index on a table that already exists:

```
use ORD_Whatever;
```

Clustered index:

```
CREATE CLUSTERED INDEX [index_name] ON
[MySchema].[MyTable]
([MyColumn] ASC )
WITH (SORT_IN_TEMPDB = ON, DATA_COMPRESSION = PAGE)
      ON [DefFG]
```

CCI:

```
create clustered columnstore index CCI on
[MySchema].[MyTable] on [DefFG]
```

Sort in TempDB?

```
WITH (SORT_IN_TEMPDB = ON
```

- Uses TempDB to create the index.
  - So you don't hit the space limit in your ORD,
  - and it should complete faster.
- Recommended by OIT if you're on any of these servers:
  - VHACDWA01
  - VHACDWA06
  - VHACDWRB01
  - VHACDWRB02
  - VHACDWRB03
  - VHACDWDWHSQL33

Compress/Index an Object That Already Exists

- Index and compress an empty object OR a full one – choose one option but not both.
- The most common practice is to create, fill, and then index/compress a table.
  - This is NOT the best practice, merely the easiest.

Index and Compress Tables On Creation

- The actual best practice.
- More complicated to do.

```
CREATE TABLE [AH].[LewisTest4](
    [PatientICN] [varchar](50) NULL,
    [DOB] [datetime2](0) NULL,
    [VisitYear] [int] NULL,
    [Sta3n] [smallint] NOT NULL,
    [CY_Age] [int] NULL,
    index CCI Clustered Columnstore
    index idx_ICN clustered ([PatientICN] asc) with
    (data_compression = page)
)
ON [DefFG]
```

Remember, one or the other!

If an Index Already Exists

```
ALTER INDEX [index_name] ON
[MySchema].[MyTable]
REBUILD WITH (SORT_IN_TEMPDB = ON,
DATA_COMPRESSION = PAGE)
```

Which Column to Index?

- Columns that you use to search/join.
- The more unique, the better.
- Don’t worry about this for CCI – just regular indices

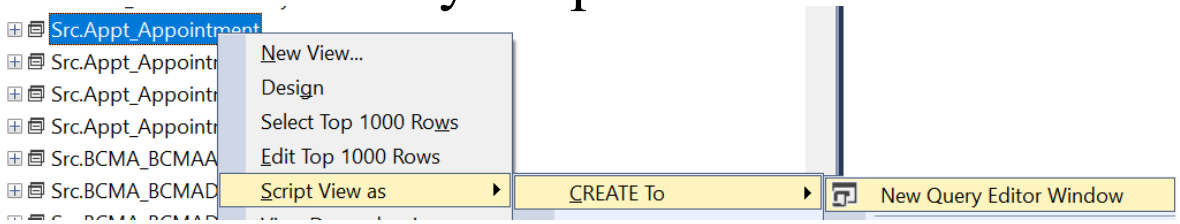
Consider a View

- If you have a complicated query that runs reasonably fast but results in a huge table, a view is a good solution.
  - The view is basically a saved query ("View Script" or "View Definition") masquerading as a table
  - You use it just like a table, but when SQL fetches records from it, it has to run the saved query first

Consider a View

- Simply add “create view [view name] as” in front of your select query

You can check one of your provisioned views for the syntax:




```
create view [Src].[Appt_Appointment] as
  select
     c.CohortName
    ,x.*
  from [CDWWork].[Appt].[Appointment] as x with (nolock)
  join [Src].[CohortPatientSID] as c with (nolock)
    on c.[PatientSID] = x.[PatientSID]
  where 1 = 1
```

Reports

- Top Tables
- Sp_report_compression
- If you run into a permissions issue, contact VINCI

Top Tables

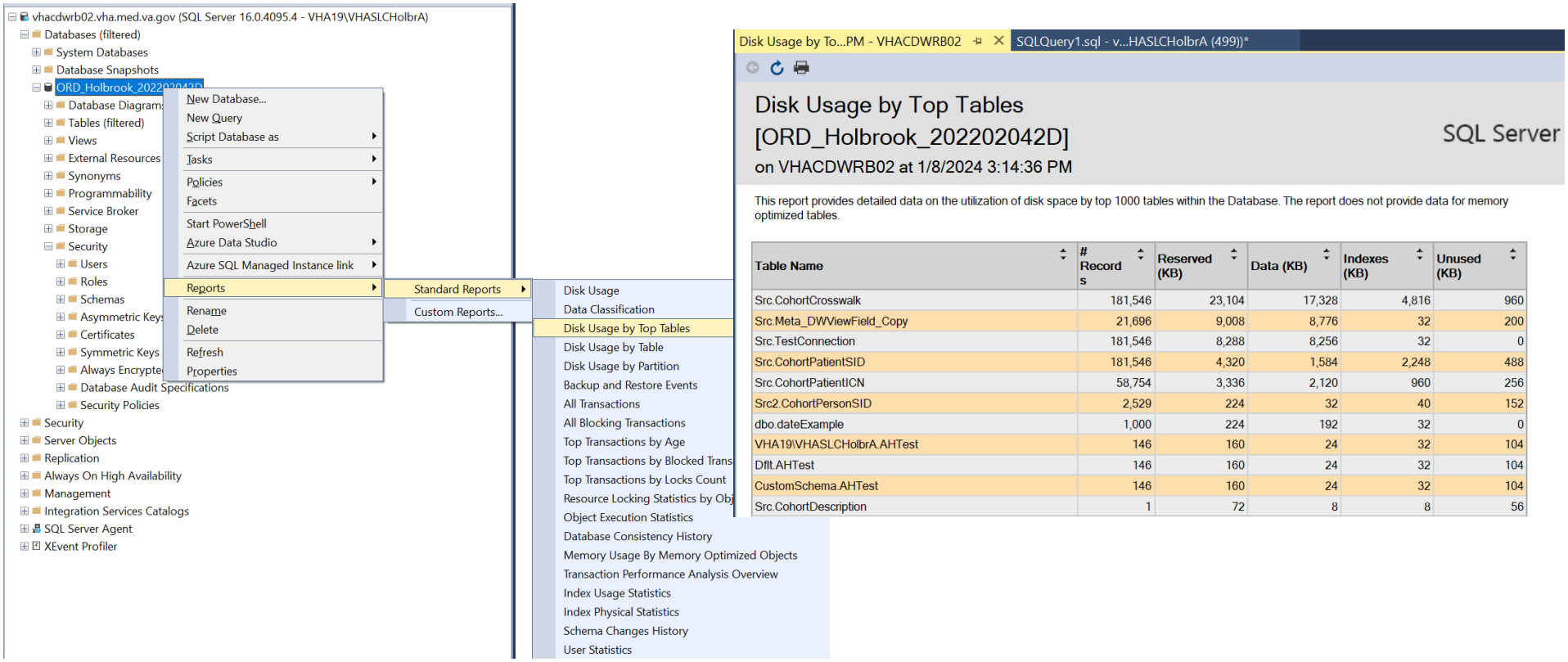


Table Name	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
Src.CohortCrosswalk	181,546	23,104	17,328	4,816	960
Src.Meta_DWViewField_Copy	21,696	9,008	8,776	32	200
Src.TestConnection	181,546	8,288	8,256	32	0
Src.CohortPatientSID	181,546	4,320	1,584	2,248	488
Src.CohortPatientICN	58,754	3,336	2,120	960	256
Src2.CohortPersonSID	2,529	224	32	40	152
dbo.dateExample	1,000	224	192	32	0
VHA19\VHASLCHolbrA.AHTest	146	160	24	32	104
Dflt.AHTest	146	160	24	32	104
CustomSchema.AHTest	146	160	24	32	104
Src.CohortDescription	1	72	8	8	56

SP_Report_Compression

```
1  use ORD_Holbrook_202202042D
2
3  exec sp_report_compression
```

214 %

Results | Messages

	Schema Name	Table Name	Index Name	Constraint Type	Index Type	File Group	Row Count	Reserved (mb)	Data (mb)	Index (mb)	Comp Type	Comp Candidate
1	CustomSchema	AHTest	NULL		Heap	DefFG	146	0	0	0	None	No
2	dbo	dateExample	NULL		Heap	DefFG	1,000	0	0	0	None	No
3	Dflt	AHTest	NULL		Heap	DefFG	146	0	0	0	None	No
4	Src	CohortCrosswalk	ccix_CohortCrosswalk		Clustered columnstore	DefFG	181,546	22	16	4	Columnstore	No
5	Src	CohortCrosswalk	pk_CohortCrosswalk	PK Constraint	Nonclustered	DefFG	181,546	22	16	4	Page	No
6	Src	CohortDescription	PK_CohortName	PK Constraint	Clustered	DefFG	1	0	0	0	Page	No
7	Src	CohortPatientICN	CCIX_CohortPatientICN		Clustered columnstore	DefFG	58,754	3	2	0	Columnstore	No
8	Src	CohortPatientICN	pk_CohortPatientICN	PK Constraint	Nonclustered	DefFG	58,754	3	2	0	Page	No
9	Src	CohortPatientSID	CCIX_CohortPatientSID		Clustered columnstore	DefFG	181,546	4	1	2	Columnstore	No
10	Src	CohortPatientSID	PK_CohortPatientSID	PK Constraint	Nonclustered	DefFG	181,546	4	1	2	Page	No
11	Src	Meta_DWViewField_Copy	NULL		Heap	DefFG	21,696	8	8	0	None	Yes
12	Src	TestConnection	NULL		Heap	DefFG	181,546	8	8	0	None	Yes
13	Src2	CohortPersonSID	PK_CohortPersonSID	PK Constraint	Clustered	DefFG	2,529	0	0	0	Page	No
14	VHA19\VHASLCHolbrA	AHTest	NULL		Heap	DefFG	146	0	0	0	None	No

Key Takeaways

- Don’t be greedy.
- Design purposefully.
- Use indices and compression.
- Monitor your use of resources.

Additional Resources

- VINCI Office Hours every Wednesday at 3 PM ET.
- SQL Office Hours with BISL on Tuesdays and Fridays.
- CDW Guide Using Page Compression
- CDW Guide Query Best Practices.docx (va.gov)
- Best_Practice_on_Building_Tables_final_ref4.docx (sharepoint.com)

Acknowledgements

- VINCI Data Services
- SAS Admins
- CDW Data Services and Field Support

Questions?

- Appendix Slides after this!

OK…but what is Normalization?

Visit table

Patient Name	Visit date	Diagnosis	Patient SSN
Bruce Banner	10/23/2020	Irritability and anger, acute radiation syndrome	1234
Bruce Banner	10/23/2020	Irritability and anger, acute radiation syndrome	1234
Peter Parker	9/15/2020	Ailurophobia	5678
Sue Storm	10/20/2020	Body dysmorphia	1537

bad better

Diagnosis table

Patient Name	Visit date	Diagnosis	Patient SSN
Bruce Banner	10/23/2020	Irritability and anger	1234
Bruce Banner	10/23/2020	Acute radiation syndrome	1234
Peter Parker	9/15/2020	Ailurophobia	5678
Sue Storm	10/20/2020	Body dysmorphia	1537

First normal form - every row is distinct and no groups of things in a field. Use more columns and rows if you have to.

Diagnosis table

Patient Name	Visit date	Diagnosis	Patient SSN
Bruce Banner	10/23/2020	Irritability and anger	1234
Bruce Banner	10/23/2020	Acute radiation syndrome	1234
Peter Parker	9/15/2020	Ailurophobia	5678
Sue Storm	10/20/2020	Body dysmorphia	1537

bad better

Patient table

Patient ID	Patient Name	Patient SSN
1	Bruce Banner	1234
2	Peter Parker	5678
3	Sue Storm	1537

Diagnosis Table

Patient ID	Visit date	Diagnosis
1	10/23/2020	Irritability and anger
1	10/23/2020	Acute radiation syndrome
2	9/15/2020	Ailurophobia
3	10/20/2020	Body dysmorphia

Second normal form - columns can't depend on part of the key. Use more tables if you have to.

A key is the column or combination of columns that uniquely define a row.

Diagnosis Table

Patient ID	Visit date	Diagnosis	Clinic	Location
1	10/23/2020	Irritability and anger	MH	1st floor
1	10/23/2020	Acute radiation syndrome	Rad	3rd floor
2	9/15/2020	Ailurophobia	MH	1st floor
3	10/20/2020	Body dysmorphia	MH	1st floor

bad better

Clinic Table

Clinic	Location
MH	1st floor
Rad	3rd floor

Diagnosis Table

Patient ID	Visit date	Diagnosis	Clinic
1	10/23/2020	Irritability and anger	MH
1	10/23/2020	Acute radiation syndrome	Rad
2	9/15/2020	Ailurophobia	MH
3	44124	Body dysmorphia	MH

Third normal form - No transitive dependencies, i.e. every column depends on the key.

- A table in third normal form said to be "normalized." Note that splitting everything into different rows and tables as we've done here makes everything clearer, more logical, and more efficient BUT more difficult to query because we need to look across rows and tables to get the information.
- The CDW is highly but inconsistently normalized, so there are many tables and we have to use lots of joins to get the information we want. For research, we often denormalize for reporting and analysis, creating a "flat file."

Row Compression

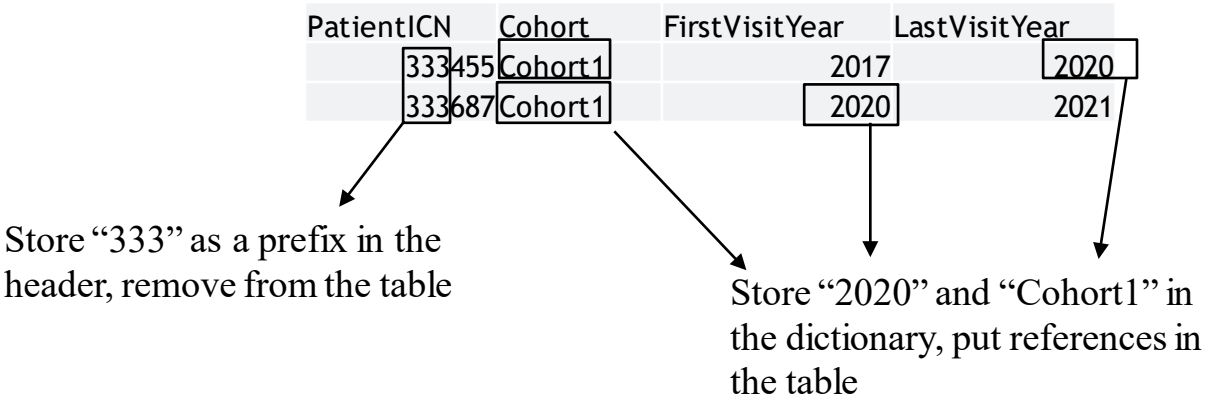
- Doesn’t store blank or zero.
- Remove trailing spaces and zeroes.
- Stores time/money/etc as integers.

number	number	char	char
34.000	0	blah space space space	blank

number	number	char	char
34		blah	

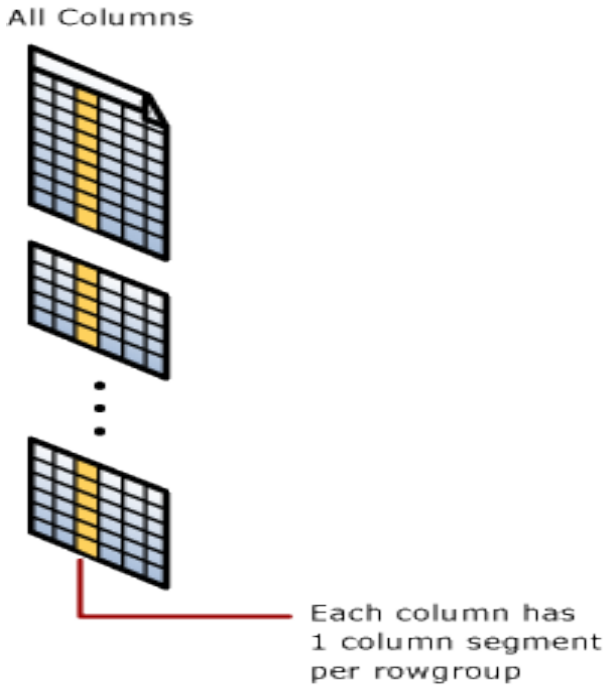
Page Compression

- Implements row compression first.
- Then looks for patterns and stores prefix/dictionary values in the header and references in the table.



Clustered Columnstore

- Stores columns instead of rows.
- Similar algorithm to page compression, but better, especially for larger tables.



SAS Data Access Tips

- No need for total replication
  - Don't simply bring in all tables/views into SAS – use SQL passthroughs to limit instead
- SQL commands can be "passed through" using T-SQL or SAS-translated from SAS syntax
  - Latter method is discouraged
- SQL familiarity is a requirement because of the size and architecture of the CDW
- SAS datasets can be indexed

SAS Data Management Tips

- SAS datasets can be indexed
- Delete or archive temp or work data
  - No need to save every intermediate file
- Hash coding if you're an advanced user
- SAS can be used to prep data for R
  - Use SAS and R together
- CSV files are a common but not recommended practice
- For more info: VINCI SAS/Grid (va.gov)