

CS-442 Notes

October 18, 2021

Introduction to **VIEW** and **WITH**

WITH is a way to create intermediate tables / temporary tables without actually making another table.

Difference between **VIEW** and **WITH**

- Views are not temporary and are 'public property' meaning anyone can access it.
- Stick to **WITH** for now on any assignments and classwork.
- Both serve the same use case of building intermediate solutions.

Nested subquery = table.

From Chapter 3, Slide 42:

```
SELECT branch_name, avg_balance
FROM (SELECT branch_name, avg(balance)
      FROM account
      GROUP BY branch_name)
      AS branch_avg( branch_name, avg_balance)
WHERE avg_balance > 1200
```

WITH equivalent:

```
WITH br_avg AS
(
  SELECT br_name, avg(bal) AS avg_bal
  FROM acct
  GROUP BY br_name
)
SELECT br_name, avg_bal
FROM br_avg
WHERE avg_bal > 1200
```

This is the answer to question 1 on the HW!

Step 1: Get the aggregates

```
WITH agg AS
(
    SELECT cust, min(quantity) AS min_q, max(quantity) as max_q,
    avg(quantity) AS avg_g
    GROUP BY cust
)
```

Step 2: Get details for minimum

```
WITH agg AS
(
    SELECT cust, min(quantity) AS min_q, max(quantity) as max_q,
    avg(quantity) AS avg_g
    GROUP BY cust
),
min_detail
(
    SELECT a.cust, a.min_q, a.max_q, a.avg_q, s.prod AS min_prod, s.date AS
min_date, s.state AS min_state
    FROM agg AS a, sales AS s
    WHERE a.cust = s.cust
    AND a.min_q = s.q
)
```

Step 3: Final output

```
SELECT m.cust, m.min_q, m.prod, m.date, m.state, m.max_q, m.avg_q, s.prod,
s.date, s.state,
FROM min_detail AS m, sales AS s
WHERE m.cust = s.cust
AND m.max_q = s.quantity
```

October 20th

Views Continued

Imagine views as the join of two tables, but in reality they are still maintained independently

A view is just a **query**.

```
WITH T AS (  
  SELECT *  
  FROM dep d, acct a  
  where d.account_num = a.account_num  
)
```

Then we can use this temporary table...

```
SELECT cn, bal  
FROM t  
WHERE cn = 'Sam'
```

You can imagine T as a temporary table, but in reality it is actually just the query surrounded by the **WITH**.

- **WITH** is private to you.
- **VIEW** provides a public **WITH**.

How to "create" a view:

```
create view T as  
  SELECT *  
  FROM dep d, acct a  
  where d.account_num = a.account_num
```

The virtual table where views are stored:

view_name	query
T	the query for T
complex	2000 line sql query

Why are we storing queries? This allows to access the most recent data and storing tables would just store old data.

How long does a view take for the code below?

```
CREATE view COMPLEX as  
--- 2000 line sql that takes 2 hours to run ---
```

Creating the views: Very little time, you're just storing the query

Using the view: 2 hours.

View is simply a named query.

October 22nd

Modifying the database

Deletion

This will remove all of the tuples that are associated with the 'Perryridge' branch

```
DELETE FROM account  
WHERE branch_name = 'Perryridge'
```

Insertion

Take a look at slide 52 on Chapter 3.

What happens when you do something like this?

```
INSERT INTO account(branch_name)  
values('Perryridge')
```

A few things to notice:

- We don't have to assign column values
- Column values not assigned will default to NULL

DELETE and INSERT are **row** wise operations!

UPDATE is **column** wise operations!

October 25th

Advanced SQL

Child table's responsibility to know its parents. Parents have no reference.

Integrity Constraints

- ex. $1/1/1980 < \text{DOB} < 1/1/2021$

Object-Relational DB

- More complex data types than typical DBMS.
- Dates, timestamps, time, interval, etc...

Varchar vs char

- Varchar = variable length, does not pad, saves space.
- Char = fixed length, will pad if not enough chars.
- Varchar is really useful for huge databases.
- Good example is adding in very long names.

BLOB vs CLOB

- More complex types of data, media, etc...
- These are stored separately, BLOB and CLOBs are basically pointers to these special data types.

picture	transcript	...
BLOB (binary, bits)	CLOB (char, bytes)	

Homework Question 3

```
WITH base AS (
  SELECT MO, PROD, sum(q) AS TOTAL_Q
  FROM sales
  GROUP BY MO, PROD
)

SELECT MO, sum(Q) as MPPQ
FROM base
GROUP BY MO
```

October 27th

What is the significance of Question 2 on the HW?

- The difficulty comes in from trying to extract extra details, such as the date from a group by.
- We solve this by creating extra joins with main sales table.

HW2 Revision

- Need to use a full outer join at the end.

Question 4:

Group by product and month, get the sums of each product in that month, take the max and min.

November 1st

Chapter 7: Relational Database Design

All about OLTP

This unit is very conceptual...

OLTP:

- Maintenance
- Updates
- Write-oriented

OLAP:

- Analytical
- Read
- Fact/finding
- Knowledge discovery

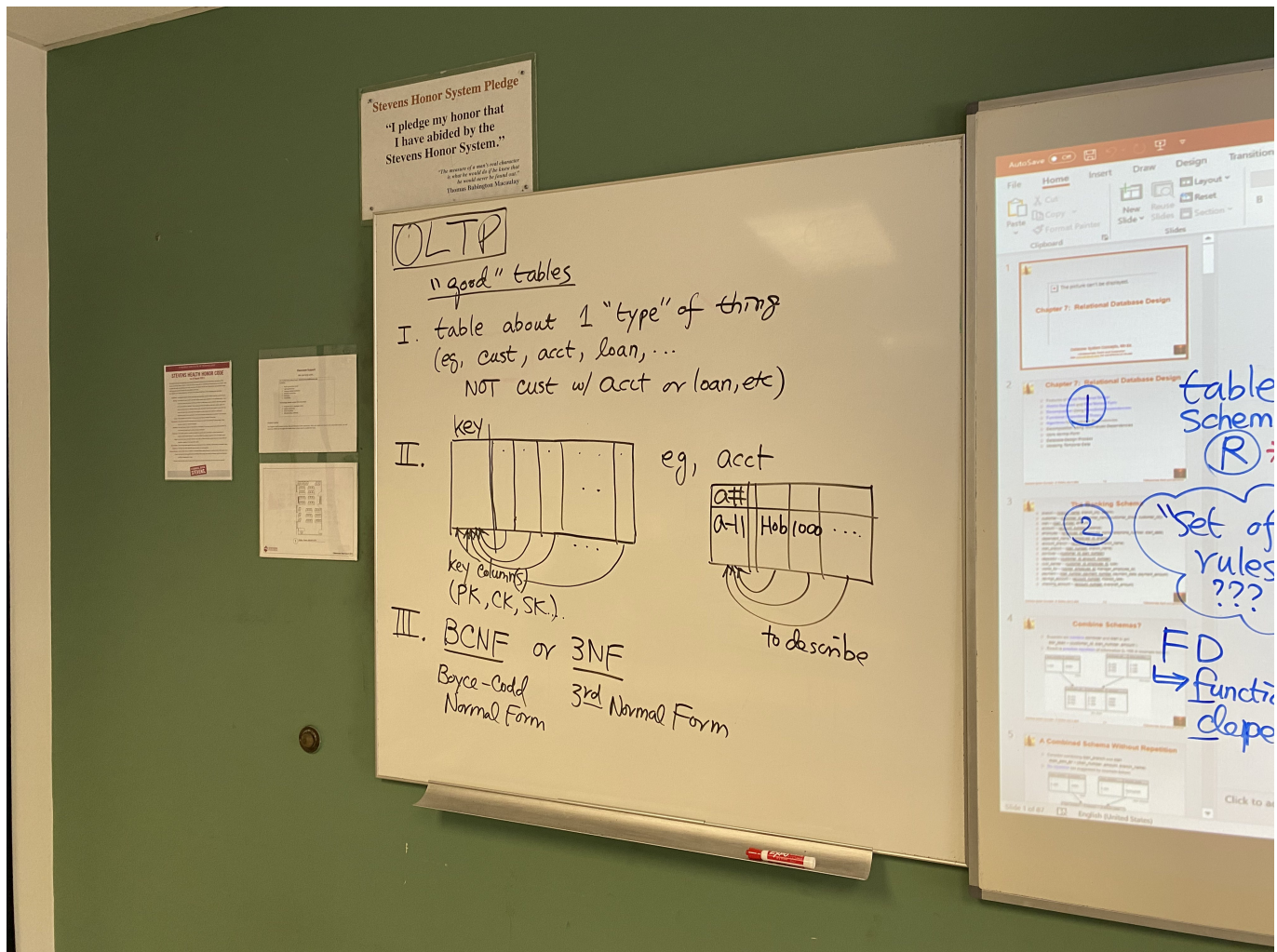
ETL: **E**xtract, **T**ransform, **L**oad

- Transitioning from one type of DB to another

For OLTP we want to minimize updates, OLAP minimize joins.

What are "good" tables?

1. Table about 1 "type" of thing (eg. cust, acct, loan, ... NOT cust w/acct or loan, etc).
2. There is one key column, where all the other columns in the table are uniquely identified by that key column. Key column to identify, other columns to describe what the entities are.
3. BCNF: Boyce-Codd Normal Form or 3NF: 3rd Normal Form.



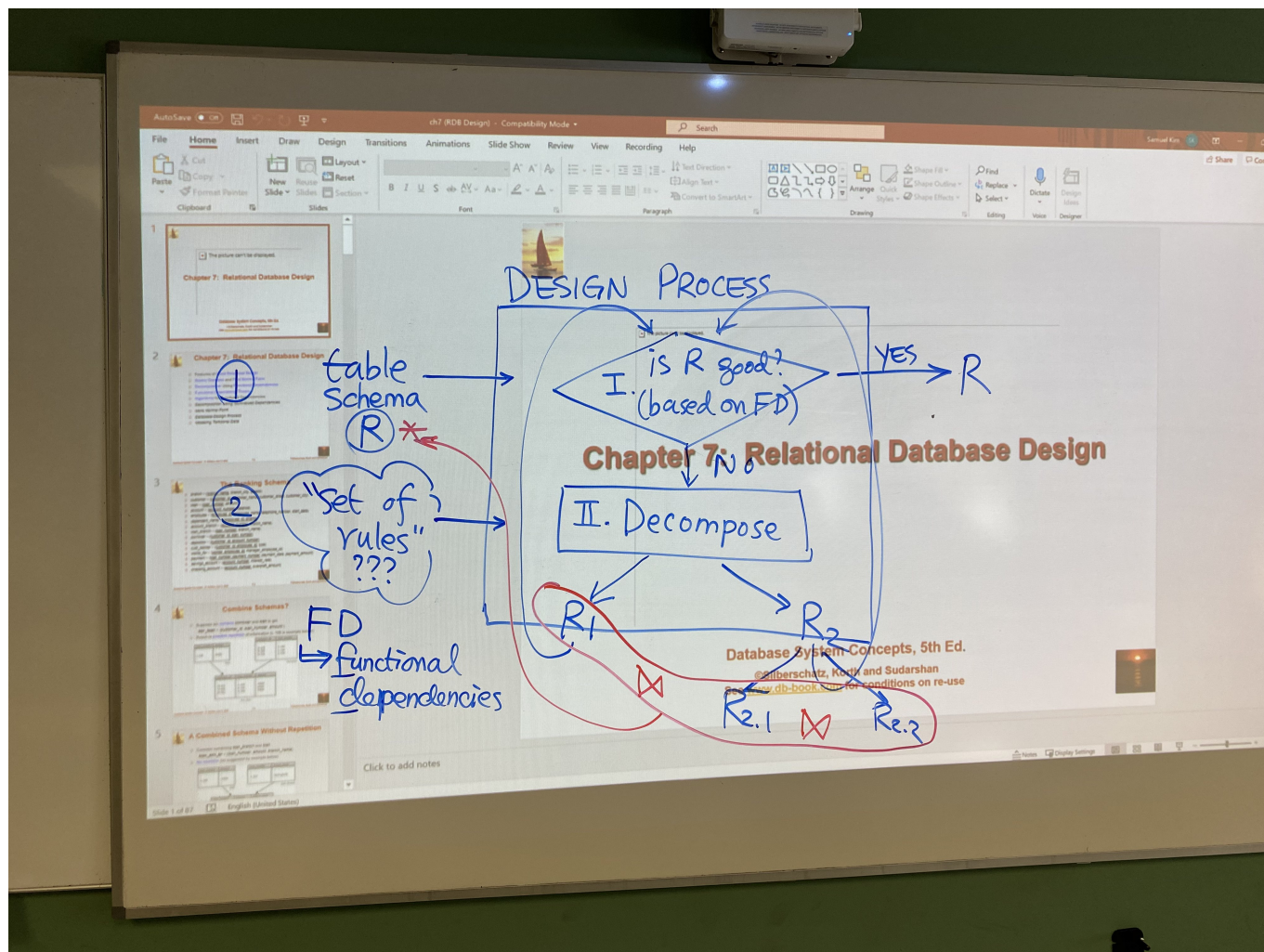
Design Process:

Table Schema → is X good? based on FD?

GOOD → then **do nothing** (design process is done for that schema)

NOT GOOD → Decompose it. Separate and split into multiple schemas.

"Set of rules" or Functional Dependencies (FD)



November 3rd

Chapter 7 Continued

When you have more than one thing in the table you will have repetition of the data on your table.

A good sign that work needs to be done on your table.

Splitting / Decomposing technique is very important.

Functional Dependencies:

- Left hand side column(s) functionally determines what shows up on the right hand side.
- If I know what is on the left side then I'll know what will be on the right hand side.
- Functional Dependency is similar to the key concept and is used to describe the idea behind keys.

If I know the CWID then I should know the major for example, and duplicate entries should be easy to determine as well.

November 5th

Chapter 7 continued again...

Decomposition of the tables should be done in a way so that we can compose it again through a natural join.

LLJD: Loss Less Join Decomposition

- AKA, we get the exact same table back, no modification of anything.
- Lossy Join Decomposition is not good. Lossy doesn't necessarily mean you're losing data, it means you're losing the original semantics or structure of the original table.
- Check slide 7.7 for a good example.

Functional Dependencies:

- alpha determines beta ($\alpha \rightarrow \beta$)
- alpha and beta are just a set of columns
- We use the idea of FD to determine the concept of keys.
- Examples:
 - SSN \rightarrow Name
 - CWID \rightarrow Addr.
 - In this case SSN and CWID would be primary keys.

When you have a column that functionally determines all of the columns, we know that said column will be a key.

We use functional dependencies to determine what a key is