Chapter 1

August 28, 2019 1:16 PM

- Terms
  - Data - raw facts
  - Information - the result of processing raw facts to produce a meaning
  - Knowledge - the body of information about a specific subject
  - Data management - the organization of data
- Database - a shared integrated computer structure that stores a collection of:
  - End-User Data - the most important piece of information
  - Metadata - data about data
- Database Management System (DMBS) - a collection of programs that manage a database structure and controls access to data stored in a database - gatekeeper
  - Advantages
    - Sharing - users have better access to data in an ever-changing environment
    - Security - Databases pick no favorites
    - Integration
    - Reduces Inconsistency - when different versions of the same data appear in different locations
    - Access
      - Adhoc query - a spur of the moment question
    - Decision Making - can bring different perspectives to help make decisions
    - Productivity
- Database Types (Categories)
  - Number of Users
    - Single-user database - only one user can use the database at a time
    - Multi-user database - more than on user can use the database at the same time
      - Workgroup database (usually under 50 users) - lower cost
      - Enterprise database (usually over 50 users) - higher cost
  - Location
    - Centralized database - found at 1 site and is easy to manage
    - Distributed database - found at more than one location
      - Multiple copies of the same data
  - How it is used
    - Operational database - general use database (constantly used)
      - Transactional database
      - Analytical database (data warehouse) - only used in special situations
  - Degree structured
    - Unstructured database - raw values and numbers
    - Structured database - stored in a "ready-to-use" format
    - Semi-structured database - some data is structured
- Commercial DBMS Market
  - Oracle
  - MySQL - open-source
  - SQL Server - Microsoft's commercial DBMS
  - PostgreSQL - open-source, object-oriented database (generally used by academics)
  - DB2 - IBM
- Database Design - the underlying foundation of a database
  - The File System - where all database management started from
  - Breaking a file into pieces
    - Data - raw facts
    - Field - a group of data pieces that form a meaning
    - Record - a set of fields that describes a person place or thing
    - File - a group of related records
      - Working with files
        - Requires 5 basic tasks
          - Create
          - Add
          - Delete
          - Modify

- ◇ List

- ❖ Data Redundancy
  - ○ Inconsistency
  - ○ 3 anomaly types:
    - ★ ▪ Modification anomalies - when data is modified in an inconsistent matter
    - ★ ▪ Insertion anomaly - when data is added incorrectly
    - ★ ▪ Deletion anomaly - when you remove data incorrectly
- ❖ Why databases are preferred?
  - ○ DBMS provides a degree of abstraction
- ❖ Database Systems
  - ○ 5 parts:
    - ★ ▪ Hardware
    - ★ ▪ Software
      - □ OS - operating system
      - □ The DBMS runs on the operating system
      - □ Application/Utility programs - written to work with the data
    - ★ ▪ People
      - □ System administrator - oversees the database's general operation
      - □ DBA - Database Administrator - works directly with the DBMS
      - □ Database Designer - in charge of the structure and flow of data
      - □ System Analyst (programmer) - interacts with the database
      - □ End-User - people who use the applications and perform operations
    - ★ ▪ Procedures
      - □ The instruction and rules that govern the system
    - ★ ▪ Data
- ❖ General Functions a DBMS
  - ○ Data Storage
  - ○ Backing up data
  - ○ Provides an Access Language
    - ▪ Query languages - non-procedural programming languages; tells the computer what to do not how to do it
  - ○ Allows for communication with reporting systems
  - ○ Provides tools to transform data
  - ○ Metadata management
    - ▪ Data dictionary or System catalog
  - ○ Enforces Data Integrity - minimizes redundancies and increases efficiency
  - ○ Multi-user Access is concurrent access control
  - ○ Secures the data - allows and blocks privileges for specific users

Chapter 2

September 4, 2019 2:13 PM

- Data Model
  - Simple representation of real-world data structures
  - Communication tool - helps others understand the overall concept
- Parts of a data model
  - Entity - a thing about which data will be collected
  - Attribute - describes an entity
  - Constraint - a rule, requirement, or procedure
  - Relationship - describes an association between entities
    - 1:M - "one-to-many" relationship
    - 1:1 - "one-to-one" relationship
    - M:M - "many-to-many" relationship (same as "M:N")
- Business rule - a brief, precise, unambiguous description of a policy, procedure, or principal within an organization
  - Very important for data models accurately
  - Provides a standard view
  - Business rules allows us to appropriately depict relationships
- Hierarchical Model
  - Every entity has a 1:M relationship
  - Advantages
    - Easy concept to understand
    - Implementation of a DBMS
  - Disadvantages
    - Hard to implement
    - No standard
    - Implementation of relationships
- Network Model
  - CODASYL - it's goal was to develop a standard for databases (Conference Of Data System Languages)
  - 3 major parts:
    - Schema - a conceptual organization of a database viewed as a whole
    - Subschema - a part of the whole database viewed in the eyes of the user
    - Data management Language
      - Data definition language - metadata language
      - Data manipulation language - End-user data language
  - Advantages
    - Easy concept
    - Can do a M:M relationship
  - Disadvantages
    - Very complicated to implement
    - Unable to effectively handle ad-hoc queries
- Relational Model - A relational model for data and large information data banks
  - Dr. Edgar F. Codd in 1970
  - The best model
  - Terms
    - File - Entity Set, Table, Relation
    - Record - Row, Entity, Tuple
    - Field - Column, Attribute
  - Advantages
    - Easy to understand and implement
    - Allowed for ad-hoc queries
    - Used the relationships between data to find data

- Disadvantages
  - Too simple to implement

★ ❖ Entity relationship Model - only adds graphical diagrams to the relational model
- ★ Peter Chen in 1976
- ★ Entity relationship Diagrams - a graphical tool of the relational model
  - Chen diagram
    - Extremely specific; lots of information
    - Difficult to find what you're looking for
  - ★ Crow's foot diagram - a community created diagram
    - Designed for implementation
    - Easier to understand

★ ❖ Object-oriented Model - only adds objects to the relational model
- Object instances are "rows"
- Objects include attributes and methods
- ★ Uses UML (Unified Modeling Language) diagrams

★ ❖ Big Data - a movement to find new and better ways to organize enormous amounts of data
- 3 characteristics (The 3 V's)
  - Volume
  - Velocity
  - Variety
- No SQL - not based on the relational model (it still supports SQL)
  - More concerned about its performance rather than the accuracy of the data

❖ Degrees of Abstraction
- ★ External - subschema (how a user sees the data)
- ★ Conceptual - schema (how the administrator sees the data)
  - We will design databases at this level
- ★ Internal - the implementation of the database at the software level
- ★ Physical - both hardware and software dependent

Chapter 3

September 9, 2019 2:21 PM

- ❖ Relational Database Model
  - Focus on tables (files)
  - Characteristics of tables
    - A table is perceived as a 2-dimensional structure composed of rows and columns
    - Each table row (tuple) represents a single entity occurrence within the entity set
    - Each table column represents an attribute, and each column (field) has a distinct name
    - Each intersection of a row and column represents a single data value
    - All values in a column must conform to the same data format
    - Each column has a specific range of values known as the attribute domain
      - Has constraints
    - The order of the rows and columns is unnecessary to the DBMS
    - Each table must have an attribute or combination of attributes that uniquely identifies each row
  - ★ Data Dictionary - stores metadata (data about the data)
    - Stored in system tables
    - Everything is stored and managed in the exact same way
  - Keys - the main focus of the relational model
    - Determination
      - ★ Functional dependence - the state of knowing the value of one attribute giving you access to other attributes
        - ★ Determinant - is used to find the determinant
        - ★ Dependent - is found using the determinant
  - Key types
    - ★ Composite Key - the key is made up of more than one attribute
    - ★ Super key - any key that identifies a row uniquely
    - ★ Candidate Key - a subset of a super key (all unnecessary attributes are removed)
    - ★ Primary Key - the best candidate key used to uniquely identify the entity
      - Limitations:
        - Must always have a value
        - No part of the primary key can be missing
    - ★ Foreign Key - a key on a table that is the primary key for a different table (allowed to be NULL)
    - ★ Secondary Key - used for data retrieval and to limit data
  - NULL - the lack of a value; the absence of something
    - Has no data type
    - Typeless and valueless
    - Meaning:
      - Could be an unknown value or a known value
      - Could be a non-applicable situation
    - Can cause issues with calculations
  - Table Integrity
    - Entity integrity - the primary key of every table must be unique and can never be NULL
    - Referential Integrity - the foreign key must have values that are either valid primary key values of the table being referenced or it must be NULL
- ❖ Relational Algebra
  - 8 Operators
    - ★ Union - A u B - starts with a base and adds to it
      - Union compatible - must have the same number of columns and the columns must be of the same data type
      - Can have a different number of rows

- ★ Intersection - A n B - only displays the intersecting data (is possible to never intersect)
- ★ Difference - A - B - start with a set of values and remove any values that match
  - All three above automatically remove duplicates (must be union compatible)
- ★ Product - A x B - works with any type of table
  - Cartesian - Every combination of row with every combination of another
  - The product of two tables will always result in the product of rows with rows and columns with columns
- ★ Select - A*o* (age > 20) - restricts the rows
  - The "*o*" in "A*o*" is lower-case *sigma*
- ★ Project - A*pi* (name, id) - combines equal columns
- ★ Divide - A / B - see page 90
- ★ Join - A *horizontal hourglass* B - most people take about an inner join
  - Inner Join - Connects two tables and finds where they actually connect
    - Natural join - the selection condition makes this natural - has 3 operators:
      - Cartesian product of 2 tables
      - Selection - is done in columns with the same name and the same value in both tables
      - Projection - removes one of all the duplicates
    - Equijoin - *horizontal hourglass theta* a = b
    - Theta Join - *horizontal hourglass theta* a > b
  - Outer Join - the result of an inner join plus the result of the unmatched items
    - Left Outer Join - =*horizontal hourglass* - keep all the items on the left table whether there is a match or not
    - Right Outer Join - *horizontal hourglass*= - keep all the items on the right table whether there is a match or not
    - Full Outer Join - =*horizontal hourglass*= - left outer join union with a right outer join

❖ Controlled Redundancy - we can find information as it is needed
- Tables share a common attribute

❖ Making Relationships - Primary and Foreign keys
- ★ 1:M
  - Primary to Foreign key (most common)
    - The Foreign key always goes on the many side of the relationship
- ★ 1:1
  - Primary to Primary key
  - Or Primary to Foreign key with a restriction
- ★ M:M
  - More complicated
  - Linking Table (Bridge Table, Composite Table)
    - Split the problem into two separate relationships by creating a new table
    - You will have two 1:M relationships connecting to the linking table

❖ General thoughts
- History? - history can often change relationships
- Primary Keys - the fewer the attributes the better
- Indexing - allows for quick access to data (often associated with secondary keys)
- Dr. Codd's Database Rules - 12 rules pg. 105

Chapter 4

September 23, 2019 2:00 PM

- Entity Relational Model created by Peter Chen
  - Chen - very academic version
    - Diagram
  - Crow's foot - we will use this to make our programs
    - list
- Entities and Attributes
  - Entities - rectangle box
  - Attributes are represented differently in the Chen and Crow's Foot models
    - Ellipses
  - Shorthand form:
    - TABLE_NAME(primary key, attribute, attribute,....)
    - CLASS(crs code, class section, class_time, class_room)
- Attribute Types (Chen)
  - Composite attribute - an attribute with sub attributes
  - Simple attribute - an attribute broken down as small as we want to store it
- Attribute Values
  - Single-valued attribute - an attribute that can store only one value
  - Multivalued attribute - an attribute that may have one or more values
- Multivalued Attributes
  - Must be avoided in implementation by one of two courses of action:
    - Keep the original entity, but split the multivalued attribute into several single-valued attributes. This is usually easier in the short term, but can lead to major structural problems in the table later
    - Create a new entity composed of the components of the multivalued attribute
      - The better solution
      - Crow's Foot cannot implement this idea
- Derived Attribute - indicated using a dotted line (Chen)
  - An attribute that can be calculated from other attributes
- Relationships
  - Cardinality and Connectivity
    - Connectivity - the kind of relationship (Crow's foot symbol and a single line symbol)
    - Cardinality - the minimum and maximum numbers, example: (1,1)(1,4)
      - Should always come from a business rule
  - Relationship Participation (the inner vertical line symbol) - mandatory or optional
    - Crow's Foot Symbols: - NEVER USE (M:N) "Many to Many"
      - (0,N) - "Many" side is optional
      - (1,N) - "Many" side is mandatory
      - (1,1) - "1" side is mandatory
      - (0,1) - "1" side is optional
- Existence Dependencies - for an entity to exist another entity must exist first
  - Existence dependence - can only exist with at least one other entity
    - Participation is mandatory
  - Existence independence - can exist without any other entity
    - Participation is optional
- Relationship Strength
  - Weak (non-identifying) relationships - dotted or dashed line
  - Strong (identifying) relationships - solid line

        - Often causes composite keys and become undesirable
    - Entity names should always be capitalized
    - Attributes should be clear as to what they are (based on the business rules)
- Relationship degrees (Chapter 4)
    - Unary - a relationship that relates to itself
    - Binary - relationships between two entities
    - Ternary - relationships to three or more entities

Chapter 5

October 4, 2019 2:17 PM

- Advanced ERD's
  - EERM/EERD
    - Extended (Enhanced) Entity Relationship Model/Diagram - special tool not to be used every day
    - Allow depictions of specialization
    - Hierarchies (supertype/subtype)
      - Inheritance
      - Subtype Discriminator - an attribute of the parent that tells us what child is connected to it
        - Must be an attribute from the parent entity
      - Disjoint and overlapping constraints - can use only: "d" and "o"
        - "d" - Disjoint means that the parent can only have one child
          - Must have one discriminator
        - "o" - Overlapping means that a parent can have more than one child
          - Uses boolean and must have more than one discriminator
      - Completeness constraint - basically the same thing as the relationships between different entities
        - Partial completeness - single line; subtypes are not required
        - Fully complete - double line; there must be at least one subtype somewhere
- Entity Cluster
  - "Virtual" entity type used to represent multiple entities and relationships in an ERD
  - Simply implements a higher level of abstraction in the diagram for readability
  - As general rule, attributes are not displayed when entity clusters are used
  - Views
- Selecting Primary Keys
  - Some real world object have a "natural" unique identifier
  - If no "natural key" is available, a combination of desirable characteristics and experience should be used to choose the key
  - PK is used for *identification*, not *description*
- Desirable PK Characteristics
  - Non-intelligent - doesn't describe the entity in any way; no embedded meaning
  - Unique values - Primary keys cannot be repeated
  - No change over time - maintainability is preferred; could cause inconsistencies with change
  - Single-attributes - much easier to find one value rather than two
  - Numeric - easier to handle on the processor
  - Security compliant - make sure that attributes are appropriately locked down (can be intelligent and security compliant)
- *Surrogate keys* meet all of the above requirements
- Design Cases
  - 1:1 Relationships
    - one side is required one side is optional, the optional side always gets the foreign key
    - If both sides are optional then choose the side that would have the fewest records or fewest NULLS
    - If both sides are required consider redoing the diagram

Chapter 6

October 14, 2019 2:01 PM

- Normalization - the only way to do normalization is to have data
  - Normalization is a process for assigning attributes to entities
    - Reduces data redundancies
    - Helps eliminate data anomalies
    - Produces controlled redundancies to link tables
    - p. 203, 204 - how the data looks
  - Repeating group - a group of things that we assume are repeating in the data
- 1NF - First Normal Form
  - Repeating groups must be eliminated (must have no assumptions) - must be the first step
  - Proper primary key developed (can only do this if there are no assumptions)
    - Uniquely identifies attribute values (rows)
  - Dependencies can be identified
    - Desirable dependencies based on primary key
    - Less desirable dependencies
      - Partial - refers to part of the Primary Key
        - Needs to have a composite primary key
      - Transitive - one nonprime attribute depends on another nonprime attribute
- 1NF Summary
  - All key attributes defined
  - No repeating groups in table
  - All attributes dependent on primary key
- 2NF - Second Normal Form - focused solely on partial dependencies
  - Start with 1NF format (short-hand form):

    ID, C# -> ~~Name~~, ~~C Name~~, Grade, Text
  - Write each key component on separate line
  - Write original primary key on last line
  - Each component is new table
  - Write dependent attributes after each key
  - 2NF format:

    ID -> Name

    C# -> C Name

    ID, C# -> Grade, Text
- 2NF
  - In 1NF
  - no partial dependencies
    - No attribute dependent on a portion of primary key
  - Still possible to exhibit transitive dependency
    - Attributes may be functionally dependent on non-key attributes
- 3NF
  - Create separate table(s) to eliminate transitive functional dependencies
    - Leaves foreign key(s)
  - 3NF format:

    ID -> Name

    C# -> C Name

    ID, C# -> Grade

    Grade -> Text
- 3NF

- In 2NF
- Contains no transitive dependencies

★❖ Boyce-Codd Normal Form (BCNF) - used to help identify the best primary key
- Every determinant in the table is a primary key
- In 3NF with the best primary key

❖ 4NF - has no multiple sets of multi-valued dependencies
- Requires 3NF
- No transitive or partial dependencies

❖ 4NF: ID, Parking #, Room #

ID, Parking #
ID, Room #

★❖ Denormalization
- Normalization is one of many database design goals
- Normalized table requirements
  - Additional processing
  - Loss of system speed
- Normalization purity is difficult to sustain due to conflict in
  - Design efficiency
  - Information requirements
  - Processing
- Un-normalized Table Defects
  - Data updates less efficient
  - Indexing more cumbersome
  - No simple strategies for creating views

Chapter 7

October 23, 2019 2:18 PM

- Microsoft SQL Server Tools 18
  - Microsoft SQL Server Management Studio
    - Server name: cssqlserver
      - New Query
        - Change "master" to CS3032_137147
- Commands:
  - CREATE - generates a new item (creates metadata)
  - TABLE - starts an entirely new set of records
  - NUMERIC - used to store numbers
  - VARCHAR - do not use CHAR
  - DATE - requires single quotes around the date ('10-Oct-19')
  - PRIMARY KEY - the most important constraint (automatically adds NOT NULL and UNIQUE)
  - FOREIGN KEY - another constraint
  - DROP - used to delete entire tables
  - REFERENCES - references to a specific table
  - ON UPDATE CASCADE - if the key changes then it will change all values connect to the key as well
  - ON DELETE CASCADE - never use this
  - UNIQUE - makes all the values in a column unique
  - NOT NULL - prevents an item from being left out (must have a value)
    - Do not add this constraint unless you are told to do so
  - CHECK - adds business rules to a column
  - DEFAULT - default to a value if no other value is assigned
  - INSERT INTO - adds data to an already existing table
  - VALUES - used to insert data into a table
  - SELECT - in SQL, it allows you to implement all the operators (requires a projection)
    - Put an asterisk after the SELECT * to show all values from a table
  - FROM - displays all the values from a table
  - WHERE - allows the restriction of rows (acts like an "if" statement)
  - DELETE FROM - very dangerous and instantaneous deletion (should always have WHERE)
  - UPDATE - used to modify data
  - SET - used to assign a value
  - DISTINCT - don't show every message; show the unique messages (gives distinct rows; cannot be the primary key)
    - Use this after the SELECT
  - AS - allows you to rename a column
  - GROUP BY - shows the subtotals of the specified group
    - Happens after the WHERE constraint
  - HAVING - works just like WHERE but works for COUNT(*); should only have aggregate functions
  - ORDER BY - allows the data to be sorted (defaults to ascending order) [always the very last step]
- Example: (casing does not matter and variables are created opposite from normal programming)

```
CREATE TABLE test(
        test_id       NUMERIC (8, 0) PRIMARY KEY,
        test_name VARCHAR (20),
        test_date   DATE
)
CREATE TABLE test_taken(
        s_number   NUMERIC (8, 0),
        test_code   NUMERIC (8, 0),
        PRIMARY KEY (s_number, test_code),
        FOREIGN KEY (s_number) REFERENCES student ON UPDATE CASCADE,
        FOREIGN KEY (test_code) REFERENCES test       ON UPDATE CASCADE
)
INSERT INTO test (test_id, test_name, test_date) - use this syntax instead of the book
            VALUES(1,         'Fred',         '13-Dec-19')

SELECT *
```

```
        FROM test
       WHERE test_id = 4

DELETE
        FROM test
       WHERE test_id = 2

UPDATE test
      SET test_name = 'Bobby', test_date = '12-25-19'
 WHERE test_id = 3 AND test_name = 'Joe'

SELECT EMP_AREACODE, EMP_MANAGER,  COUNT(*) AS total
        FROM EMPLOYEE
       WHERE EMP_LNAME LIKE '%a%'
       GROUP BY EMP_AREACODE, EMP_MANAGER
      HAVING COUNT(*) > 1
       ORDER BY EMP_AREACODE, COUNT(*) DESC

SELECT V_CODE, SUM(P_QOH)
        FROM PRODUCT
       WHERE P_DISCOUNT > 0
       GROUP BY V_CODE
      HAVING SUM(P_QOH) > 25
       ORDER BY SUM(P_QOH) DESC

SELECT VENDOR.V_NAME, AVG(P_QOH)
        FROM PRODUCT, VENDOR          - does a cartesian product
       WHERE P_DISCOUNT > 0 AND PRODUCT.V_CODE = VENDOR.V_CODE
       GROUP BY VENDOR.V_NAME

SELECT P_DESCRIPT
       FROM PRODUCT
      WHERE P_PRICE = (SELECT MAX(P_PRICE)          - a subquery can only return one row and one column - p.296
                       FROM PRODUCT
                      WHERE P_DISCOUNT = 0)

SELECT *                                            - Traditional Standard
       FROM PRODUCT P, VENDOR V
      WHERE P.V_CODE = V.V_CODE

SELECT V.V_NAME, P.V_CODE, COUNT(*)                 - ANSI standard join
       FROM PRODUCT P
       JOIN VENDOR V ON P.V_CODE = V.V_CODE
       GROUP BY V.V_NAME, P.V_CODE
       HAVING COUNT(*) > 2
```

- Not equal to: !=
  or
  <>

❖ Logical operators:
- AND
- OR
- NOT - generally use != instead

❖ 5 new operators:
- ★ IS NULL - used to check for NULL
  - IS NOT NULL
- ★ BETWEEN - used to check between an inclusive range "between 1 and 10" (lower number first)
- ★ IN - used to check values that are equal to the IN values "IN (1,2,5,7,8,21,37,95)" (avoid duplicates)
  - Never put NULL in the list and the data types must match
  - A subquery allows for one column and multiple rows
  - p. 282

- ★ LIKE - two key symbols to use:
  LIKE '%saw' - replace with 0 or more characters before the keyword
  Or
  LIKE '%saw%' - replace with 0 or more characters before and after the keyword
  LIKE '_' - replace with one and only one character
- ★ EXISTS - performs a subquery to check to see if something exists

❖ Aggregate function - takes data and summarizes it
- ★ COUNT(*) - counts the number of rows that are returned (asterisk will always count the primary key)
  - Cannot count the NULL's
- ★ SUM - can only sum numeric data types
- ★ AVG - gives the average value of a certain column; takes the SUM and divides by the COUNT
  - Can only average numeric data types
- ★ MIN - only works with numeric or date data types
- ★ MAX - only works with numeric or date data types

❖ p. 311, 312 - do practice functions (DB: CS303)
❖ JOIN - p. 300-304
❖ Test over chapter 6: Normalization, 7: SQL - all-written

Chapter 8

November 13, 2019 2:00 PM

- Natural JOIN

```
SELECT *
FROM PRODUCT, VENDOR
WHERE product.V_CODE = vendor.V_CODE
```

- ANSI style JOIN ON

```
SELECT *
FROM PRODUCT JOIN VENDOR ON product.V_CODE = vendor.V_CODE
or
SELECT*
FROM PRODUCT JOIN VENDOR USING (V_CODE)
```

- ANSI style Outer joins

```
SELECT *
FROM PRODUCT LEFT JOIN VENDOR ON product.V_CODE = vendor.V_CODE
and
SELECT *
FROM PRODUCT RIGHT JOIN VENDOR ON product.V_CODE = vendor.V_CODE
and
SELECT *
FROM PRODUCT FULL JOIN VENDOR ON product.V_CODE = vendor.V_CODE
```

- Subqueries

```
SELECT *
FROM PRODUCT
WHERE V_CODE IN (SELECT v_code
                 FROM vendor)
OR V_CODE IS NULL
or
SELECT *
FROM PRODUCT LEFT JOIN VENDOR ON product.V_CODE = vendor.V_CODE
WHERE VENDOR.V_CODE IS NULL
```

- Set Operators
  - In a UNION both tables must have the same number of columns and the same data types
  - UNION automatically removes duplicates

```
SELECT V_CONTACT
FROM VENDOR

UNION

SELECT EMP_LNAME
FROM EMPLOYEE
```

  - This will show all values including duplicates

```
SELECT V_CONTACT
FROM VENDOR
```

UNION ALL

SELECT EMP_LNAME
FROM EMPLOYEE

- Pseudo columns

SELECT V_CONTACT, 1
FROM VENDOR

UNION

SELECT EMP_LNAME, 2
FROM EMPLOYEE
or
SELECT V_CONTACT as last_name, 'V' as source_table
FROM VENDOR

UNION

SELECT EMP_LNAME, 'E'
FROM EMPLOYEE

- Pseudo tables

SELECT '-- Select a value' as last_name, '-1' FROM dual -(only include "FROM dual" when using Oracle)

UNION

SELECT EMP_LNAME, 'E'
FROM EMPLOYEE

- Other set operators

SELECT V_CONTACT
FROM VENDOR

INTERSECT - (intersection)

SELECT EMP_LNAME
FROM EMPLOYEE
and
SELECT V_CONTACT
FROM VENDOR

EXCEPT - (Oracles uses "MINUS" instead)

SELECT EMP_LNAME
FROM EMPLOYEE

- JOIN intersection this only works if the value is a primary/foreign key connection

SELECT DISTINCT V.V_CODE
FROM VENDOR V
JOIN PRODUCT P ON V.V_CODE = P.V_CODE

- Subquery intersection

```
SELECT V_CODE
FROM VENDOR
WHERE V_CODE IN (SELECT DISTINCT V_CODE
                 FROM PRODUCT
                 WHERE V_CODE IS NOT NULL)
or
SELECT DISTINCT V_CODE
FROM PRODUCT
WHERE V_CODE IN (SELECT V_CODE
                 FROM VENDOR)
```

- More intersections (no primary or foreign keys)

```
SELECT DISTINCT V_CONTACT
FROM VENDOR
WHERE V_CONTACT IN (SELECT DISTINCT EMP_LNAME
                    FROM EMPLOYEE
                    WHERE EMP_LNAME IS NOT NULL)
```

- More difference operator

```
SELECT V_CODE
FROM VENDOR

EXCEPT

SELECT V_CODE
FROM PRODUCT
```

- Subquery difference

```
SELECT DISTINCT V_CODE
FROM PRODUCT
WHERE V_CODE NOT IN (SELECT V_CODE
                     FROM VENDOR)
OR V_CODE IS NULL
or
SELECT V_CODE
FROM VENDOR
WHERE V_CODE NOT IN (SELECT DISTINCT V_CODE
                     FROM PRODUCT
                     WHERE V_CODE IS NOT NULL)
```

- OUTER JOIN difference (order matters)

```
SELECT V.V_CODE
FROM VENDOR V
LEFT JOIN PRODUCT P ON V.V_CODE = P.V_CODE
WHERE P.P_CODE IS NULL                    - this should always be the primary key
or
SELECT DISTINCT P.V_CODE
FROM PRODUCT P
LEFT JOIN VENDOR V ON V.V_CODE = P.V_CODE
WHERE V.V_CODE IS NULL
```

- P.418, 419 put in the three tables and practice

Chapter 8-2

November 20, 2019 2:16 PM

- Subqueries - must return one and row one column

```
SELECT *
FROM product
WHERE p_price > (SELECT AVG(p_price) FROM product)

SELECT p_descript, p_price, (SELECT AVG(p_price) FROM product)
FROM product
```

- IN - allows subqueries to use more than one row and more than one column

```
SELECT v_contact
FROM vendor
WHERE v_contact IN (SELECT emp_lname FROM employee)
```

- Proper intersection

```
SELECT DISTINCT v_contact
FROM vendor
WHERE v_contact IN (SELECT DISTINCT emp_lname FROM employee
                    WHERE emp_lname IS NOT NULL)
```

- Multiple rows and columns can be used in the FROM before a subquery

```
SELECT *
FROM (SELECT p_code, p_descript, v_name, v_contact
      FROM product p JOIN vendor v ON p.v_code = v.v_code) a
JOIN LINE l ON a.p_code = l.p_code
```

- Virtual table - allows you to take a query and save it as a table (CREATE, DROP)

```
CREATE VIEW product_vendor AS
SELECT p_code, p_descript, v_name, v_contact
FROM product p JOIN vendor v ON p.v_code = v.v_code

DROP VIEW product_vendor
```

- Use the saved query (not actually a table)

```
SELECT *
FROM product_vendor
```

- Views allow us to add security to our data

```
CREATE VIEW product_secure AS
SELECT p_code AS product_pk, p_descript AS name, p_qoh AS QOH, p_discount AS dis, v_code AS vendor_key
FROM product
```

- Views allow for data migration
- Materialized view (created by Oracle) - when a view is stored and periodically updated (data is out of date if data is constantly changing)
  - p.349-356
- Correlated subquery - is like a nested for loop for subqueries

Chapter 8-4

November 25, 2019 2:01 PM

- The GETDATE() function
  SELECT GETDATE()

- How to display the date in Oracle
  SELECT sysdate FROM dual

- to_char() allows you to convert the date into a string
  SELECT to_char(sysdate, 'month day YYYY') FROM dual

- to_char() syntax:
  MONTH: name of the month
  MON: three-letter month name
  MM: two-digit month name
  D: number for day of week
  DD: number or day of month
  DAY: name of day of week
  YYYY: four-digit year value
  YY: two-digit year value

- to_date()
  SELECT to_date('101112', 'DDMMYY') FROM dual

- add_months() to add or subtract months
  SELECT add_months(sysdate, -1) FROM dual

- If you want to add days:
  SELECT sysdate + 90 FROM dual

- last_day() displays the last day of that month
  SELECT last_day(sysdate) FROM dual

- MySQL is known for being open-source
- ABS(numeric_value) - absolute operator
- round() - rounds up to the selected decimal places
  SELECT round(3.456789, 2) FROM dual

  SELECT floor(3.456789, 2) FROM dual -MySQL/SQL Server

  SELECT ceiling(3.456789, 2) - MySQL

  SELECT ceil(3.456789, 2) FROM dual - Oracle

- String functions (specific according to DBMS)
  SELECT emp_fname + ' ' + emp_lname FROM employee - SQL server

  SELECT emp_fname || ' ' || emp_lname FROM employee - Oracle

  SELECT concat(emp_fname, emp_lname) FROM employee - MySQL

SELECT lower(last_name) FROM employee - MySQL/Oracle/SQL Server

SELECT substr(first_name, 1, 3) || last_name FROM student - Oracle

SELECT length(last_name) FROM student - Oracle

SELECT len(last_name) FROM student - SQL server

- Special functions for doing conditional value sets with data (decision)
- Decode() is the function that allows a switch statement

```
SELECT decode(grade_type_code, 'HM', 'Homework',
                               'QZ', 'Quiz'
                               'Something else')
FROM grade
```

- Does a NULL check - Oracle

```
SELECT decode(comments, NULL, 'No comment', comments)
FROM grade
```

- Oracle's shortcut - Oracle

```
SELECT decode(comments, NULL, 'No Comment', comments),
       NVL(comments, 'No Comment')
FROM grade
```

- Works on all DBMS:

```
SELECT decode(comments, NULL, 'No Comment', comments),
       NVL(comments, 'No Comment'),
       CASE WHEN comments IS NULL THEN 'No Comment'
       ELSE comments
       END
FROM grade
```

- Procedural Language SQL, three major types:
  - Trigger - event-driven code
    - Logging trigger - commonly used
  - Procedure - a void function that allow you to process things
  - Function - any other stored code that returns a value
- In order to use these commands you must have a cursor
- A cursor can retrieve data and execute SQL
- A sequence gives a number representing a key value
- Refer to the text for commands and DBMS implementations

Chapter 9

November 27, 2019 2:00 PM

- information systems provide for data collection, storage, transformation, and retrieval (data as a whole)
- ★ SDLC - System Development Life Cycle
  - 5 phases:
    - ★ Planning - asks three basic questions:
      - Should the existing system continue as is?
        - Can the current system be modified?
      - Should the existing system be replaced?
      - What are the resources required for complete replacement?
        - What is the day 1 cost?
        - What is the operational cost
    - ★ Analysis
      - What does the customer want?
      - What do the regulations require?
    - ★ Detailed system design
    - ★ Implementation
      - Where you actually implement and build everything that was planned, analyzed, and designed previously
    - ★ Maintenance
      - Three types of maintenance:
        - ★ Corrective maintenance - maintenance on something that is not working as intended
        - ★ Adaptive maintenance - maintenance that occurs when the business has changed
        - ★ Perfective maintenance - maintenance that improves on what is already working properly
- ★ Case tools - computer aided software engineering tools that put all interfaces and information together in one program
  - ★ Microsoft Access
- Design philosophies:
  - ★ Top-down design - takes generic ideas and becomes more specific as it's developed
  - ★ Bottom-up design - starts with lots of required detail and builds into larger categories

Chapter 10/Final

November 27, 2019 2:42 PM

Transaction Management and Concurrency Control

- A Database Request can be considered to be a single SQL statement
- A Transaction is a logically indivisible set of one or more database requests. it represent a real-world event such as the sale of a product
- A Consistent State is when all integrity constraints are satisfied
- Transaction properties:
  - Atomicity - must be a single, indivisible, logical unit of work.
  - Durability - once a transaction is committed, the consistent state that is reached cannot be lost
  - Serializability - if several transactions are executed at the same time (in a multi-user database) the resulting consistent state is the same as if the transactions had been executed one at a time
  - Isolation - all data from a transaction must come from a consistent state in the database

Transactions in SQL

- Transaction support in SQL is provided by the keywords:
  - COMMIT - the transaction was successful and is now permanent
  - ROLLBACK - the transaction was not successful and restore the previous state
  - BEGIN TRANSACTION (SQL Server)
- The database is not considered to be in a consistent state until immediately after a COMMIT or ROLLBACK has occurred
- All SQL states are implicitly committed unless inside of a BEGIN TRANSACTION (SQL Server)
- The Transaction Log is a set of one or more system tables that is maintained automatically by the DBMS
  - Helps to guarantee the Durability and Serializability

Concurrency Control

- Concurrency Control is the management of concurrent transaction executions (multiple transactions occurring at the same time)
  - Handles multiple transactions that occur at the same time
- The DBMS uses a special process called the scheduler to determine the serial order of transactions
- The scheduler is focused on three main types of data inconsistencies:
  - Lost Update - when two transactions update the same value at the same time using its original value to make their calculations
  - Uncommitted data - when one transaction is cancelled in the middle (and so all changes are undone) but meanwhile another transaction uses a value that had been updated by the first transaction
  - Inconsistent Retrievals - when a calculation or select is done over data that is in the process of being updated by another transaction. Some data is read before it is changed and some after the change
- The scheduler attempts to avoid these problems using three major techniques:
  - Locking - guarantees that a transaction has exclusive use of a data item
  - Time-stamping - gives an idea of when a transaction started
  - Optimistic - assumes that transaction conflicts will be rare
- The scheduler also ensures CPU efficiency by staying as busy as possible even during I/O operations
- Locks have two properties:
  - Granularity - level of the lock
    - Database level - entire database is locked (only one person can be looking at the data at one time)
    - Table level - all needed tables are locked
    - Page level - disk block locked (usually the best level)
    - Row level - only needed rows are locked
    - Field level - only needed fields are locked
  - Type - Binary or Shared
    - Binary locks - an object is either locked or not locked
    - Shared/Exclusive Lock - an object can be unlocked, shared (read), or exclusive (write)
- Two-phase locking:
  - Growing Phase - occurs as you are acquiring needed locks
  - Shrinking Phase - when you start to get rid of keys for others to use

- Deadlocks occur when two transaction are waiting for each other to release their locks (and they never do)
- Time-stamping can solve the issue of a deadlock by using the age of the transaction to say what goes first
- 2 major approaches to time-stamping:
  - Wait/Die - if the older table needs a lock from the younger table it will wait until the younger is done; if the younger needs a lock from the older then the younger will die (rollback) and will reschedule
  - Wound/Wait - if the older table needs a lock from the younger table it will wound (rollback) the younger table and the younger table is rescheduled; if the younger needs a lock from the older then the younger will wait for the older to finish
- Optimistic Concurrency Control assumes that transaction conflicts will be very rare
- Each transaction goes through 3 phases:
  - Read Phase
  - Validation Phase
  - Write Phase
- Transaction Recovery
- 4 important concepts to understand:
  - Write-Ahead-Log Protocol - the log is the most important thing, no the data (the log is immediately updated and permanent; written immediately to a disk)
  - Redundant Transaction Logs - the log is kept in multiple different locations
  - Database Buffers -
  - Database Checkpoint - where the buffer is permanently written to the hard disk
- 2 methods of transaction recovery:
  - Deferred update
  - Immediate update
- Flashback Queries show the database instance as it was previously

```
SELECT *
    FROM employee AS OF
    TIMESTAMP ('13-SEP-04 8:50:58',
                'DD-MON-YY HH24: MI SS')
```

- Snapshots in SQL Server are similar to Database Checkpoints

```
CREATE DATABASE mydb2
    ON ( NAME = mydb, FILENAME = 'C:\mydb.ss' )
    AS SNAPSHOT OF mydb;
SELECT *
    FROM [mydb2].dbo.Employees
```

- Similar to tests; Cumulative; Database terminology file terminology; data models (relational model); relational algebra; relationship diagrams; normalization; SQL data definition commands; data manipulation command; subqueries; transaction management; database design; otherwise look at the course outline