Midterm 2 Study guide

DEFINITIONS:

- candidate keys
  - A field within a database that can identify each unique record. It has no redundant columns and is in its most minimal representation of the tuples
  - A table can have multiple candidate keys
    - i.e. – a super key in its most minimized form
- composite keys
  - A key that consists of two or more attributes that uniquely identify any row in the table. The unique keys by themselves cannot define the table.
- Primary Keys
  - The most appropriate candidate key for the table. Not null and unique
- foreign keys
  - column representing a primary key form another table
- superkey
  - A collection of any number of columns which can uniquely identify each other column
- Tuple
  - another name for a record or row
- FUNCTIONAL DEPENDENCIES
  - In a table (w, x, y, z) a functional dependency could be
    - w,x → y, z
  - Meaning that w and x can map onto y and z columns
  - determinant
    - in the above example w and x are the determinants
  - Dependents
    - In the above, the dependents are y and z
- Partial dependency
  - Determinant is a proper subset of some candidate key
  - Dependent column is a non-key column
- Transitive dependency
  - Determinant is not a subset of some candidate key and is not a candidate key itself
  - Dependent column is a non key-column
- 1NF - First Normal Form –
  - the table has no multi value columns
- 2NF - Second Normal Form –
  - 1NF and the table has no partial dependencies
- 3NF – Third Normal Form -
  - 2NF and the table has no transitive dependencies
- BCNF – Boyce Codd normal form
  - 3NF and where X → Y, x must be a superkey
  - Or, non-key values cannot determine key values

- Armstrong's axiom
    - Reflexivity: if X ⊆ Y then Y → X
    - Augmentation: If X → Y, then XZ → YZ for any Z
    - Transitivity: If X →Y and Y → Z then X → Z
- Insertion anomaly
    - Inserting a row into table is not possible because the insertion lacks a specific column.
    - The information cannot be captured because of the missing column information
- Update Anomaly
    - Updating a row in a table causes multiple other rows to need to update as well, or the data become incorrect
- Deletion anomaly
    - Deleting a row from a table causes a loss of information.
    - For example, if I had a house table and I deleted a row with the only reference to a specific room, it would be as if that room no longer existed

Normailizing a Table:

Steps to normalize:

1. Identify all Functional Dependencies
2. Identify all Candidate Keys
3. Identify all Partial dependencies and transitive dependencies
4. Arrange P.D.s and T.D.s into statements that share dependencies
    i. Ie.
        1. X→y
        2. X→w
        3. X→Z
        4. Becomes – x → y,w,z
5. Decompose the table repeatedly with respect to each functional dependency statement
    a. Create two tables
        i. Right side contains all F.d we decompose from
        ii. Left contains all but the dependent column
        iii. Apply foreign key constraints to columns that are primary keys on the right side

SQL Statements

- LENGTH()
    - **Returns** an integer of the length of the source string
        - **LENGTH(sourceString)**
- UPPER() and LOWER()
    - **Returns** a new string with all the alphabetic characters replaced with either upper or lower case
        - **UPPER(sourceString)** or **LOWER(sourceString)**

- SUBSTR()
    - **Returns** a string starting at the position specified and of the length specified
        - **SUBSTR(sourceString, startPosition, [length])**
        - Note that the first character is indexed at 1 and that if the optional parameter of length is not used, the default is the final index location of the string
        - An out of bounds index returns null
- INSTR()
    - **Returns** a integer of the index of a string being searched for within another string
        - **INSTR(sourceString, searchString, [start], [-nth])**
            - Indexing begins at 1, the optional parameter of start can be included to specify where the first index number to be searched
            - The option -nth parameter can be included to find the nth occurrence of the search parameter
- REPLACE()
    - **Returns** the string but replaces an occurrence of a specified string with another
        - **REPLACE(sourceString, searchString, [replaceWith])**
            - The optional replace with parameter can be used to swap the found search string with itself. If not included, the default is to delete the found string
- CONCAT()
    - **Returns** a string of two string parameters concatenated together
        - **CONCAT(string1, string2)**
        - The short hand || can be used to mean the same thing
            - **string1 || string2**
        - chr(10) is the ascii number for newline used in Oracle DB's
- NVL()
    - **Returns** an expression of the specified datatype. Similar to a ternary operator
        - **NVL(exp1, exp2)**
            - exp2 is returned if exp1 is a null value, else exp1 is returned
                - exp2 is automatically cast to the same datatype as exp1
- TO_DATE()
    - **Returns** a date value from the source string in the format specified
        - **TO_DATE(sourceString, [formatString], [nls_language(not used)])**
            - Format can be 'dd mm yyyy' or dd/MONTH/ YY' ect.
            - Sysdate can be used to specify the current date else a string of numbers is used for source string ie – '20100531' for 31 MAY, 2010
- TO_CHAR()
    - **Returns** string from the source string in the specified format
        - **TO_CHAR(sourceString, [formatString], [nls_language(not used)])**
        - Can be formatted as:
            - '9,999.99' for a float number up to 4 digits
            - '$9,999.00' for a float number formatted as money with always to decimal points
            - 'dd/mm/yyyy' for dates or however else you want to format the date

    - A note on sysdate – You can add or subtract an integer to modify the date. 1 signifies a single day
        - Sysdate – 1 is the day yesterday
        - Sysdate – 1/24 is one hour ago
        - Sysdate + 7 is a week from now
        - Sysdate + 1/(24*60) is one minute from now
- DECODE()
    - Similar to an if-then statement
        - DECODE(sourceColumn, 'stringSearch','replaceWIth', [elseUseThis])
            - The source column specifies the column to decode
            - String search is the 'if'
            - Replace with is the 'then'
            - elseUseThis is the 'Else'
        - for example:
            - DECODE(city, 'Kelowna', '250-'||phone, phone)
            - Reads as:
                - In the City column if it matches 'Kelowna' then print phone with '250-' concatenated in front, else just print phone.