EXAM II Review

CIS 205

Database Management Systems

Relational Databases

- A relational database is a collection of tables/relations
- Each entity is stored in its own table
- Attributes of an entity become the fields or columns in the table
- Relationships are implemented through common columns in two or more tables

Relation

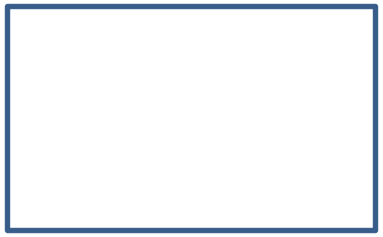
- **Relation:** two-dimensional table in which:
  - Entries are single-valued
  - Each column has a distinct name (called the attribute name)
  - All values in a column are values of the same attribute
  - Order of columns is immaterial
  - Each row is distinct
  - Order of rows is immaterial

Relational Tables

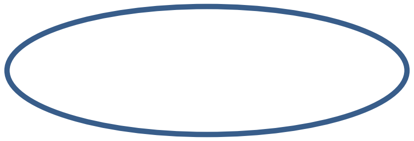
- Explain what is a Primary Key
- **Primary key:** column or collection of columns of a table (relation) that uniquely identifies a given row in that table
- **Foreign Key:** a primary key in another table. Links to primary key item

- Create an Entity Attribute Relationship (EAR) Diagram

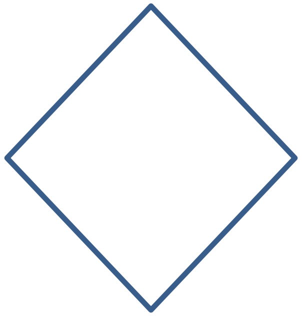
- Entity



- Attribute



- Relationship



- **What is a Query?**

- **Query: SELECT, UPDATE, DELETE:** a question represented in a way the DBMS can recognize and process
- **Query-By-Example (QBE) MS ACCESS**
  – Visual approach to writing queries
  – Users ask their questions using an on-screen grid
  – Data appears on the screen in tabular form

- **Select Query:** a query that retrieves and aggregates data satisfying the criteria entered in the query
- **Update query:** a query that changes data
  - Makes a specified change to all records satisfying the criteria in the query
- **Delete query:** permanently deletes all records satisfying the criteria entered in the query

Some Query Terms

- **Simple Queries** – Single table queries
- **Criteria**: conditions that data must satisfy
- **Criterion**: single condition that data must satisfy

Functions

- Built-in **functions**
  - Called **aggregate functions**

- Count
- Sum
- Avg (average)
- Max (largest value)
- Min (smallest value)

- **Comparison Operators**
  - Also called a **relational operator, there are six**

    = (equal to)

    > (greater than)

    < (less than)

    >= (greater than or equal to)

    <= (less than or equal to)

    NOT (not equal to)

Boolean Operators

- **Compound criteria**, or **compound conditions**
  - **AND criterion:** both criteria must be true for the compound criterion to be true
  - **OR criterion:** either criteria must be true for the compound criterion to be true

Sorting

- **Sorting:** listing records in query results in an ordered way
- **Sort key:** field on which records are sorted
- **Major sort key**
  - Also called the **primary sort key**
  - First sort field, when sorting records by more than one field
- **Minor sort key**
  - Also called the **secondary sort key**
  - Second sort field, when sorting records by more than one field

- Explain Functional Dependency

- **Functional Dependency:** An association between a primary key and attribute or set of attributes; functional dependency exists if at every moment in time, for one value of the key, one and only one value of the attribute or set of attributes exists.

- Explain difference between a relation and a relationship

A **Relation** is a Table

A **Relationship** is an association between Tables

- Explain Data Redundancy
- Data redundancy is a data organization issue that allows the unnecessary duplication of data

Duplicate Data in the system

Causes integrity problems, which version of the data is correct?

Causes performance problems, slows down the system

- **Data Dictionary – Meta Data**
- A software tool for recording the definition of data, the relationship of one category of data to another, the attributes and keys of groups of data, and so forth.

What is Cardinality

- Is the Relationships between data tables explains how each table links to another.
- Explain the different types of cardinality and the circumstances to use them
- 1 – 1 - What do you do
- 1 – M - What do you do
- M – M - What do you do

- **One to One Cardinality**

Most of the time you want to merge the two tables together

Other times you may make the most frequently used table the primary key table and the less frequently used table the foreign key table

- **Many to Many Cardinality**

Create a Mapping/Translation Table, place the primary keys of the associated entities tables into the mapping table as foreign keys. These become a composite key to uniquely identify a record/instance in the relationship file

- **One to Many Cardinality**

Do nothing, this is what you want. Ensure that the primary key of the one table is embedded in the many table as a foreign key attribute

- **Explain Referential Integrity**

Must insert or create Primary Key item before inserting Foreign Key items

Must delete or drop Foreign Key items before deleting or dropping Primary Key Items

- **Explain the three tier system**

Client System – Web Browser

Middleware – JSP, ASP, PHP

Database Server – MySQL. Oracle, DB2, MS Server

Usually on three or more computer systems. All three may reside on one computer system during application development and testing.

- Explain the enterprise model and the file system model

Enterprise – centrally managed data, one set of data to many applications

File System – duplicate sets of data, one application to one set of data, a problem is incompatibility across applications and data redundancy

- **Be able to Create a Data Dictionary**

Field Names

Field Descriptions

Field Data Types

Field Constraints

Identify Primary and Foreign Keys

Categorize by Table

- **Be able to Translate Relational Schema into a Stable Translation**
- **In this type of Translation:**

All Entities Become Tables

All Relationship Become Tables

Bring Primary Keys of Entities into associated Relationship Tables and make them foreign keys in the Relationship Table

- Be able to Normalize a Relational Schema

Check for 1st Normal Form

repeating groups, composite attributes, multi-valued attributes

Check for 2nd Normal Form

Identify PK, All non Key Attributes are determine by PK

Check for 3rd Normal Form – no Transitive Dependencies A -> B ->C , A cannot -> C

Explain Projection and Selection

Projection filters columns

- Takes a vertical subset of a table
- Causes only certain columns to be included in the new table

Selection filters rows

- Takes a horizontal subset of a table
- Retrieves certain rows from an existing table (based on criteria) and saves them as a new table
- Includes the word *WHERE* followed by a condition

- Explain Union, Intersection, Difference

- **Union** merges two tables with identical data types sequentially across
- **Intersect** identifies like items in both tables based on attributes
- **Difference** identifies what is in one table but not duplicated in another table

Normal Set Operations

- **Union** of tables A and B
  - Table containing all rows that are in either table A or table B or in both table A and table B
- **Intersection** of tables A and B
  - Table containing all rows that are common in both table A and table B
- **Difference** of tables A and B
  - Referred to as A minus B
  - Set of all rows that are in table A but that are not in table B

Joins

- Allows extraction of data from more than one table
- **Join column:** column on which two tables are joined

- *American National Standards Institute – ANSI*

- ANSI SQL/99 features include ANSI compliant joins. There are several advantages in using this new syntax, one of which is the separation of the join condition from the WHERE clause.

Joins

- **Explain Natural Join, Left Outer Join, Inner join Right Outer Join and Full Outer Join**
- Examine these two tables

Employee Table	
LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
John	NULL

Department Table	
DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

- **Natural Join** - The resulting joined table contains only one column for each pair of equally-named columns. The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate.
- No more than two tables can be joined using this method. So, it is best to avoid natural joins when possible.

DepartmentID	Employee.LastName	Department.Department Name
34	Smith	Clerical
33	Jones	Engineering
34	Robinson	Clerical
33	Steinberg	Engineering
31	Rafferty	Sales

Inner join creates a new result table by combining column values of two tables The query compares each row of A with each row of B to find all pairs of rows which satisfy the join-predicate

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Robinson	34	Clerical	34
Jones	33	Engineering	33
Smith	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

- **Left outer join -** The result of a *left outer join* (or simply **left join**) for table A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B).

Employee.LastName	**Employee.DepartmentID**	**Department.DepartmentName**	**Department.DepartmentID**
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
John	`NULL`	`NULL`	`NULL`
Steinberg	33	Engineering	33

- **Right outer joins -** A right outer join returns all the values from the right table and matched values from the left table (NULL in case of no matching join predicate).

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

- **Full outer join -** A **full outer join** combines the results of both left and right outer joins. The joined table will contain all records from both tables, and fill in NULLs for missing matches on either side.

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
John	NULL	NULL	NULL
Steinberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	*Marketing*	*35*

- SELECT command selects only certain rows
- PROJECT command selects only certain columns
- JOIN command combines data from two or more tables based on common columns
- Normal set of operations: union, intersection, and difference
- Product of two tables results from concatenating every row in the first with every row in the second

NESTING OF QUERIES

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*
  - Queries can be specified in an alternative form using nesting
- *Query: Retrieve the name and address of all employees who work for the 'Research' department.*

```
SELECT FNAME, LNAME, ADDRESS
  FROM EMPLOYEE JOIN DEPARTMENT ON DNO = DNUMBER
WHERE DNAME=‘Research‘;
```

Nested query alternative

Outer Query

```
SELECT FNAME, LNAME, ADDRESS
 FROM EMPLOYEE
WHERE DNO IN  (SELECT DNUMBER
                FROM DEPARTMENT
               WHERE DNAME=‘Research’);
```

Nested Query

NESTING OF QUERIES (contd.)

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- A reference to an *unqualified attribute* refers to the relation declared in the *innermost nested query*
- In this example, the nested query is *not correlated* with the outer query

CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query*, the two queries are said to be *correlated*
  - The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) the outer query
- *Query: Retrieve the name of each employee who has a dependent with the same first name as the employee.*

```
SELECT E.FNAME, E.LNAME                    Alias
 FROM EMPLOYEE AS E
WHERE E.SSN IN
        (SELECT ESSN                          Correlated Nested Query
          FROM  DEPENDENT
        WHERE   ESSN=E.SSN AND E.FNAME=DEPENDENT_NAME);
```

CORRELATED NESTED QUERIES (contd.)

- In the previous query, the nested query has a different result in the outer query
- A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can ***always*** be expressed as a single block query. For example, the previous query may be written as:

```
SELECT  E.FNAME, E.LNAME
FROM    EMPLOYEE E JOIN DEPENDENT D
        ON E.SSN=D.ESSN
WHERE E.FNAME=D.DEPENDENT_NAME;
```

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not
  - We can formulate the previous query in an alternative form that uses EXISTS as:

THE EXISTS FUNCTION (contd.)

- *Query: Retrieve the name of each employee who has a dependent with the same first name as the employee.*
-
```
SELECT  FNAME, LNAME
  FROM EMPLOYEE
WHERE EXISTS (SELECT * FROM DEPENDENT
                        WHERE SSN=ESSN
                          AND FNAME=DEPENDENT_NAME);
```

THE EXISTS FUNCTION (contd.)

- *Query: Retrieve the names of employees who have no dependents:*

```
SELECT   FNAME, LNAME
 FROM    EMPLOYEE
WHERE    NOT EXISTS  (SELECT    *
                       FROM     DEPENDENT
                      WHERE     SSN=ESSN);
```

- The correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If *none exist*, the EMPLOYEE tuple is selected
  - EXISTS is necessary for the expressive power of SQL

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is **NULL** (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so *equality comparison is not appropriate*.
- *Query: Retrieve the names of all employees who do not have supervisors:*

```
SELECT   FNAME, LNAME
  FROM   EMPLOYEE
WHERE    SUPERVISOR_ID  IS  NULL;
```

  – Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

SUBSTRING COMPARISON

- The **LIKE** comparison operator is used to compare partial strings
- Two reserved characters are used: '**%**' (or '*****' in some implementations) replaces an arbitrary number of characters, and '**_**' replaces a single arbitrary character

SUBSTRING COMPARISON (contd.)

- *Query: Retrieve all employees whose address is in Houston, Texas. Here, the value of the ADDRESS attribute must contain the substring 'Houston,TX' in it.*

```
SELECT      FNAME, LNAME
 FROM       EMPLOYEE
 WHERE      ADDRESS LIKE '%Houston,TX%'
```

SUBSTRING COMPARISON (contd.)

- *Query: Retrieve all employees who were born during the 1950s.*
  - Here, '5' must be the 8th character of the string (according to our format for date), so the BDATE value is '_______5_', with each underscore as a place holder for a single arbitrary character.

```
SELECT    FNAME, LNAME
  FROM    EMPLOYEE
WHERE     BDATE LIKE '_______5_’
```

- The LIKE operator allows us to get around the fact that each value is considered atomic and indivisible
  - Hence, in SQL, character string attribute values are not atomic

ARITHMETIC OPERATIONS

- The standard arithmetic operators **'+', '-'. '*', and '/'** (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result
- *Query: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.*

```
SELECT    FNAME, LNAME, 1.1*SALARY
FROM      EMPLOYEE
          JOIN WORKS_ON ON SSN=ESSN
          JOIN PROJECT ON PNO=PNUMBER
WHERE     PNAME='ProductX’;
```

AGGREGATE FUNCTIONS

- Include **COUNT, SUM, MAX, MIN, and AVG**
- *Query: Find the maximum salary, the minimum salary, and the average salary among all employees:*

```
SELECT MAX(SALARY),
       MIN(SALARY),
       AVG(SALARY)
  FROM EMPLOYEE;
```

- Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTIONS (contd.)

- *Query: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department:*

```
SELECT  MAX(SALARY),
        MIN(SALARY),
        AVG(SALARY)
  FROM  EMPLOYEE JOIN DEPARTMENT
          ON DNO=DNUMBER
  WHERE DNAME='Research‘;
```

AGGREGATE FUNCTIONS (contd.)

- *Queries: (1) Retrieve the total number of employees in the company, and (2) the number of employees in the 'Research' department.*

```
(1): SELECT  COUNT (*)
      FROM  EMPLOYEE;


(2): SELECT  COUNT (*)
       FROM EMPLOYEE JOIN DEPARTMENT
                 ON DNO=DNUMBER
     WHERE  DNAME='Research';
```

GROUPING

- In many cases, we want to apply the aggregate functions to *subgroups of tuples* in a relation
- Each subgroup of tuples consists of the set of tuples that have the *same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

GROUPING (contd.)

- *Query: For each department, retrieve the department number, the number of employees in the department, and their average salary.*

```
SELECT   DNO, COUNT (*), AVG (SALARY)
 FROM    EMPLOYEE
GROUP BY DNO;
```

  - In this query the EMPLOYEE tuples are divided into groups-
    - Each group having the same value for the grouping attribute DNO
  - The COUNT and AVG functions are applied to each such group of tuples separately
  - The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples
  - A join condition can be used in conjunction with grouping

GROUPING (contd.)

- *Query: For each project, retrieve the project number, project name, and the number of employees who work on that project.*

```
SELECT    PNUMBER, PNAME, COUNT (*)
FROM      PROJECT JOIN WORKS_ON
              ON PNUMBER=PNO
GROUP BY PNUMBER, PNAME;
```

  - In this case, the grouping and functions are applied after the joining of the two relations

THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*
- The **HAVING**-clause is used for specifying a selection condition on groups (rather than on individual tuples)

THE HAVING-CLAUSE (contd.)

- *Query: For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on that project.*

```
SELECT      PNUMBER, PNAME,
            COUNT(*)
FROM        PROJECT JOIN WORKS_ON
              ON PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING      COUNT (*) > 2;
```

Summary of SQL Queries

- *A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory. The clauses are specified in the following order:*

```
SELECT          <attribute list>
FROM            <table list>
[WHERE          <condition>]
[GROUP BY       <grouping attribute(s)>]
[HAVING         <group condition>]
[ORDER BY       <attribute list>]
```

Summary of SQL Queries (contd.)

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
  - A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

References

- http://en.wikipedia.org/wiki/Relational_algebra
- http://en.wikipedia.org/wiki/Join_(SQL)
- http://www.orafaq.com/wiki/Natural_join