Important Concepts for This Lecture

- Appropriateness, Goodness, Quality
- Top Down vs. Bottom Up Design
- Design flaws and anomalies
- General Guidelines
Database Design Issues

• Our current approach has been from a top down view
• Generally we ignore the attributes at first and start with defining entities and relationships
• Attributes are incorporated in the design as an after thought to support entities and relationships
• Attributes are sometime fabricated to correct data flaws (example: A Sequentially Generated Primary Key introduced only to support the uniqueness of the primary key)
• One indication of a poor design is major changes of the schema as new releases are implemented

Database Design Top Down Approach

• Requirements are gathered
• Entity and Attribute are grouped by common sense or intuition
• Relationships are based on how the entities were organized by the designer
• Candidate Keys are not formally defined, they are derived from superkeys
• Designs can vary depending upon interpretation of requirements and the experience of the designer/analyst
Database Design Flaws

- Entity and Attribute grouping by Common Sense allows for NO formal measures for appropriateness, goodness, and quality
- Conclusion: There is a strong need for a more formal approach to database design

Database Design Characteristics

- APPROPRIATENESS – The design fully satisfies the requirements that are stated by the user community and allow for accommodating future enhancements and modifications to the requirements.
• **GOODNESS**
  – The design presents *a clear semantic* meaning of entities and attributes
  – The design *supports effective and efficient* updating and storage of the base relations

• **QUALITY** – The design meets a set of *agreed upon* standards by a governing body and *accepted by* the designing community
Methodologies

• **Top-Down Design**
  – Starts with concepts about entities and relationships with constraints being applied. Attributes are then assigned as appropriate

• **Bottom-Up Design**
  – Starts with attributes and the associations between the attributes. Entities are build up formally from the attributes then relationships and constraints are applied

Informal Design Guidelines

Concepts

• **Semantic Clarity** - Insure the clarity of objects used in the database schema

• **Application Independent** - The design should be independent of current application constraints but should accommodate future application enhancements

• **Redundancy Reduction** - Minimize the storage space required to maintain the required information

• **Avoid Null Values** - Minimize or eliminate inconsistent or missing information

• **Avoid Spurious Tuples** - Insure correctness of queries results
Semantic Clarity

Attribute Overloading

- Each attribute should portray a clear, single, and certain meaning.
- An attribute’s meaning should not be overloaded.
- For example: having multiple meanings depending on another attribute(s).

CREATE TABLE customer  (
    customer_id       varchar2(11),
    customer_name     varchar2(20),
    street_address    varchar2(20),
    city              varchar2(20),
    state             varchar2(2),
    zipcode           varchar2(10),
    address_type      varchar2(3) CHECK (address-type IN ('B','S','D','BS','SD','BD','BSD')),
    CONSTRAINT customer_pk PRIMARY KEY (customer_id) );

NOTE: address_type indicates whether the address is a billing, service, or delivery address.
Semantic Clarity

In the following example the composite attribute *address* consisting of

- street_address
- city
- state
- zipcode

is overloaded and dependent on another attribute (*address_type*) for its meaning. It is confusing and inhibits understandable query development.

Semantic Clarity

Attributes

- **Avoid**
  - Composite attributes
  - Overloaded attributes
  - Multi-valued attributes
  - Repeating groups of attributes
  - Nested attributes
Semantic Clarity

Relation Overloading

• In grouping attributes together to form Entities and Relationships, we assume a clear, single, and certain meaning is associated with the attribute grouping
• Relations that contain multiple concepts are overloaded and cause updating issues

Semantic Clarity

Update Anomalies

• Overloading a relation with too much information creates update anomalies and possible data inconsistencies
  – Insertion anomalies
  – Deletion anomalies
  – Modification anomalies
Relation Overloading Example

**EMPLOYEE_DEPARTMENT**

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>NOT NULL</td>
<td>CHAR(11)</td>
</tr>
<tr>
<td>FNAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(15)</td>
</tr>
<tr>
<td>MINIT</td>
<td></td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>LNAME</td>
<td>NOT NULL</td>
<td>VARCHAR2(15)</td>
</tr>
<tr>
<td>BDATE</td>
<td></td>
<td>DATE</td>
</tr>
<tr>
<td>ADDRESS</td>
<td></td>
<td>VARCHAR2(20)</td>
</tr>
<tr>
<td>CITY</td>
<td></td>
<td>VARCHAR2(15)</td>
</tr>
<tr>
<td>STATE</td>
<td></td>
<td>VARCHAR2(2)</td>
</tr>
<tr>
<td>ZIPCODE</td>
<td></td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>SEX</td>
<td></td>
<td>CHAR(1)</td>
</tr>
<tr>
<td>SALARY</td>
<td></td>
<td>NUMBER(10,2)</td>
</tr>
<tr>
<td>SUPERSSN</td>
<td></td>
<td>CHAR(9)</td>
</tr>
<tr>
<td>DNUMBER</td>
<td>NOT NULL</td>
<td>NUMBER(9)</td>
</tr>
<tr>
<td>DNAME</td>
<td></td>
<td>VARCHAR2(15)</td>
</tr>
<tr>
<td>MGRSSN</td>
<td>CHAR(9)</td>
<td></td>
</tr>
<tr>
<td>MGRSTARTDATE</td>
<td>DATE</td>
<td></td>
</tr>
</tbody>
</table>

Dr. James Dullea, Database Design Considerations  Slide 18 of 40

Update Anomalies
(Assume the following tuples)

<table>
<thead>
<tr>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ZIPCODE</th>
<th>SALARY</th>
<th>DNUMBER</th>
<th>DNAME</th>
<th>MGRSSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>BORG</td>
<td>888665555</td>
<td>10-NOV-27</td>
<td>48525</td>
<td>55000</td>
<td>1</td>
<td>Headquarters</td>
<td>888665555</td>
</tr>
<tr>
<td>WALLACE</td>
<td>987654321</td>
<td>28-JUN-31</td>
<td>78965</td>
<td>43000</td>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
</tr>
<tr>
<td>SMITH</td>
<td>333445555</td>
<td>08-OCT-45</td>
<td>39090</td>
<td>40000</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
<tr>
<td>SLEYLA</td>
<td>123456789</td>
<td>09-JAN-55</td>
<td>45001</td>
<td>30000</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
<tr>
<td>ENGLISH</td>
<td>999887777</td>
<td>19-JUL-58</td>
<td>74110</td>
<td>25000</td>
<td>4</td>
<td>Administration</td>
<td>987654321</td>
</tr>
<tr>
<td>JABBAR</td>
<td>453453453</td>
<td>31-JUL-62</td>
<td>45654</td>
<td>25000</td>
<td>5</td>
<td>Research</td>
<td>333445555</td>
</tr>
</tbody>
</table>

Dr. James Dullea, Database Design Considerations  Slide 18 of 40
Update Issues

- What are the candidate keys?
- What would you make the primary key?
- How would one create a new department?
- If all the employees transferred out of the department, how do we prevent losing the department information?
- If there were 200,000 employees, how costly would it be to change the name of a department?
- What would happen if those inputting data used different but similar dept names?
  – 'Computing technology' vs 'computing technologies' vs 'computing technology'

Application Independence

- The database schema should be independent of the applications accessing the data.
- Applications should be modified to support the real meaning of the data when necessary.
- Temporary accommodations may be necessary but should be implemented as close to the application level as possible.
Application Independence

Example of a design problem

```sql
CREATE TABLE event (
    vehicle_number    varchar2(16),
    event_date_time   date,
    fault             char(4),
    found_code        char(1),
    CONSTRAINT event_pk PRIMARY KEY(vehicle_number, event_date_time, fault));
```

NOTE: found_code code from 1 to 5 that indicates how the fault was found (example 1 = inspection). An added requirement to capture a severity code from 0 to 5 is suggested by the user. The designer suggests to change the found_code to char(2) because the current screen design can only capture one number, and use codes such as 13 would equal "inspection, medium severity". Is this acceptable? What would you suggest, if you were the designer.

Redundant Information

- Attribute Grouping has a significant effect on storage, data consistency, updating and query efficiencies.
- Populating an attribute with information that can be derived from another attribute through a simple join to a much smaller table is a waste space
  - Example: a company with 100,000 employees and 10 departments where the department number and name is needed.
Redundant Information

EMPLOYEE(ssn, lname, fname, dept_number, dept_name)

CREATE TABLE employee
  ssn      varchar2(11),
  lname    varchar2(15),
  fname    varchar2(20),
  dept_number number(2),
  dept_name varchar2(20),
CONSTRAINT employee_pk PRIMARY KEY (ssn));

Redundant Information

If dept_number uniquely determines dept_name, the following problems arise.
Assume: 100,000 employees and 10 departments

Storage: Approximately 2 megabytes of storage is wasted
Data Inconsistencies: Each insert of an employee requires the department name to be enter. Typing errors at data entry could be introduced into the database
Update inefficiencies: The change of the name of a department requires accessing each employee tuple
Query Inconsistencies: In the WHERE clause should we use dept_name or dept_number.
A Better Design

EMPLOYEE(ssn, lname, fname, dept_number)
DEPARTMENT(dept_number, dept_name)

CREATE TABLE department
    dept_number number(2),
    dept_name varchar2(20),
CONSTRAINT department_pk PRIMARY KEY (dept_number));

CREATE TABLE employee
    ssn varchar2(11),
    lname varchar2(15),
    fname varchar2(20),
    dept_number number(2),
CONSTRAINT employee_pk PRIMARY KEY (ssn),
CONSTRAINT employee_fk1 FOREIGN KEY (dept_number)
    REFERENCES department(dept_number));

Null Values in Tuples

- A "FAT RELATION" is one where the designer overloads a relation with attributes that do not apply to all tuples.
- Result of NULL Values
  - Possible wasted space
  - Uncertainty in aggregate operations
  - Multiple interpretations of missing values
Null Values in Tuples
Possible wasted space

- In most implementations space is fully allocated for:
  - CHAR
  - DATE
  - NUMBER
  - BOOLEAN
- Depending on the implementation, space may not be fully allocated for:
  - VARCHAR2

Null Values in Tuples
Uncertainty in aggregate operations

- AVG and SUM do not include 'NULL VALUES' and thus the true value of the aggregate is uncertain
- COUNT(*) as compared to COUNT(attribute_name) can yield different results if NULLS are present
- MAX and MIN do not consider NULL VALUES
Null Values in Tuples
Multiple Interpretations

- A null value in an attribute instance can be interpreted as (not inclusive):
  - does not apply to this tuple
  - value is unknown for this tuple
  - value is known but not available (or recorded)
  - had a hidden meaning at data entry time
- Don’t design with NULLS in mind if they can be avoided
- Don’t design with NULLS having an intended meaning.

Spurious Tuples

- Spurious tuples are tuples that may appear after joining relations (through what appears to be a natural join) but are invalid.
- These tuples do not exist in the original conceptual relations and hence represent loss of consistent data, additional information that is not true, or loss of information.
Spurious Tuples

• Example
  – Employee_Location Table
    • EName
    • PLocation
  – Employee_Project Table
    • SSN
    • Pnumber
    • Hours
    • PName
    • PLocation

• Example
  If one was to join the Employee_Location Table with the Employee_Project Table, what would the WHERE clause in the SQL statement look like?
Spurious Tuples

• Example

• If one was to join the Employee_Location Table with the Employee_Project Table, what would the WHERE clause in the SQL statement look like?

• Is

\[
\text{WHERE Employee\_Location.PLocation = Employee\_Project.PLocation}
\]
correct?

Spurious Tuples

• The NATURAL JOIN of the two tables is on PLocation (the only common attribute in both tables)

• Joining on PLocation gives invalid results

• See Figure 10.6 in Elmasri 4rd Edition page 303 (asterisks are missing)

• See Figure 14.6 in Elmasri 3rd Edition page 475
Guideline 1
Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationships types into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, the meaning tends to be clear. Otherwise, it tends to be a mixture of multiple entities and relationships and hence semantically unclear.

Guideline 2
Design the base relation schema so that no insertion, deletion, or modification anomalies occur in the relations. If any anomalies are present, note them clearly so that the programs that update the database will operate correctly.
Guideline 3

As far as possible, avoid placing attributes in a base relation whose values may be null. If nulls are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Guideline 4

Design relation schema so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are generated.
Guideline 5

Design relation schema from the underlying data model and the business rules derived from the requirements analysis. Avoid designing the database schema from a single or small group of current reports. Consider information requirements at different levels of management.

Design Assignment

Assignment 08-1A
Design Considerations
(and Reverse Engineering)