CSE 204
INTRODUCTION TO DATABASE SYSTEMS

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

COURSE SCHEDULE

- Room, Day, & Time:
  - AMFİ2 – Tuesday/Thursday15:30
- Instructor: Joseph LEDET
  - Office: Computer Engineering Building 2nd Floor
  - E-mail: josephledet@akdeniz.edu.tr
  - Office Hours: Tuesday 12:30 – 13:30, 14:30 – 15:30 or by appointment
- Assistant: Doruk BAŞARAN
  - Office: Computer Engineering Building 2nd Floor
  - E-mail: dorukbasaran@akdeniz.edu.tr
  - Office Hours: Thursday 10:00 – 12:00 or by appointment

COURSE OBJECTIVES

- This course introduces the fundamentals of database systems including relational data model, entity/relationship model, SQL, query optimization, integrity constraints, and normalization. Upon successful completion of this course, students will be able to:
  - Understand DBMS and RDBMS
  - Explain the basic components of an E-R Model
  - Normalize a database
  - Obtain data from a database using SELECT SQL queries
  - Alter data in a database using UPDATE, DELETE, INSERT SQL queries
  - Design, create, and modify structure of a database

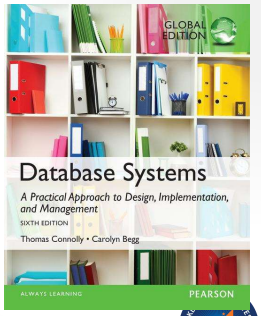
TEXTBOOK

- T. Connolly and C. Begg

Database Systems: A Practical Approach to Design, Implementation, and Management, Global Edition.

Pearson (Intl), 2014.

ISBN-13: 9781292061184



COURSE INFORMATION

- The lecture slides, announcements, and homework will be available through Microsoft Teams. Make sure that you have been added to the course

25B_CSE 204_Database Systems_Ö_1

- https://www.microsoft.com/en-us/microsoft-365/microsoft-teams/group-chat-software/

- It is **your** responsibility to check frequently for updates concerning this course.
- You can install the mobile app (Android or IOS) to your phone so that you can get notifications whenever an announcement is made.

GRADING

Category	Weight
Assignments & Quizzes	20%
Midterm	20%
Project & Presentation	20%
Final	40%

ATTENDANCE

- Attendance and class discussions are essential for the learning process
- If you miss a class it is your responsibility to find out what was discussed in the class
  - Ask and answer questions

HOMEWORK & PROJECTS

- All homework assignments are due by the date and time set by instructor as the deadline. You will lose 10% for each day they are late and homework submissions will not be accepted after two days.
- A semester project will be given. Details of this project will be discussed by the third week of class.

EXAMINATIONS

- There will be no make-up for any midterm unless a medical excuse is provided.
- Final exam will be comprehensive and you will be responsible for all the topics covered during the semester.
  - There will be a final re-take exam after the letter grades are posted. This exam will be a replacement exam for your final exam score.

ACADEMIC INTEGRITY

- If you get caught cheating, maximum possible action will be taken, including reporting the matter to the appropriate university authorities.
- Please cooperate by doing your own work and not seeking inappropriate help from your classmates.
- It is encouraged to discuss course topics and assignments amongst yourselves, as long as that discussion does not lead to an exchange of solutions.

COURSE OUTLINE

Week	Tue	Thu	Chapter	Tentative Topics
1	Feb 11	Feb 13	1, 4	Introduction, Relational Model
2	Feb 18	Feb 20	6	SQL
3	Feb 25	Feb 27	6, 7	Advanced SQL
4	Mar 04	Mar 06	10	Database System Development
5	Mar 11	Mar 13	11	Database Analysis
6	Mar 18	Mar 20	12, 13	Entity Relationship Model
7	Mar 25	Mar 27	13, 14	E-R Model, Normalization
8	Apr 01	Apr 03		Ramazan Bayramı & Normalization
9	Apr 08	Apr 10		Tentative Midterm Week
10	Apr 15	Apr 17	14, 15	Normalization
11	Apr 22	Apr 24	16	Conceptual Database Design
12	Apr 29	May 01	17, 18	Logical & Physical Database Design Emek ve Dayanışma Günü
13	May 06	May 08		Group Presentations
14	May 13	May 15		Group Presentations

COMMUNICATION

- If you need to email me, please
  - Use my university email address (josephledet@akdeniz.edu.tr)
  - Begin the subject line with "CSE 204" and include a brief (maximum of a few words) description of your comment or question
    - CSE 204: Group Project
    - CSE 204: Assignment 2
    - CSE 204: Quiz 01
  - I will do everything within my power to respond within 24 hours (business day)
  - Make sure your question has not already been addressed in the course material (slides, announcements, etc.)

COMMUNICATION

- If you have concerns about grading, contact me within a reasonable amount of time
- If you are asking to have a grade increased, have a valid reason why
  - The solution does work, but maybe the grader did not understand completely
  - The grader did not see a part of the solution on another page
  - NOTE: I really, really, really want/need to pass this class is NOT a valid reason

GROUP PROJECT

- If you have a group (of 5) and an idea, have the group leader email me with the following
  - Names of all group members (each member must be cc'ed in the email)
  - Group project idea
  - Include a bit about why you want to do this project
- Details to follow in later lectures

IMPORTANT NOTES

- Keep up with the course.
- When in doubt…ASK!
- Attempting and failing is better than not attempting at all.
  - Partial credit is your friend!
- Practice!
- If you have a class conflict or if you want to use a previous year's quiz grades or project grades, we must be notified by email no later than the end of the third class week (28-Feb-2025)

THIS WEEK

- Install a tool for SQL or MySQL/PHP
  - https://en.wikipedia.org/wiki/List_of_Apache%E2%80%93MySQL%E2%80%93PHP_packages
- Suggestions
  - XAMPP
  - ORACLE
  - Windows
    - Microsoft Access
    - WAMP
  - Mac OSX
    - MAMP
  - Linux
    - LAMP

CSE 204 - INTRO TO DATABASE SYSTEMS INTRODUCTION

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

QUOTE OF THE DAY

- Data is not information
- Information is not knowledge
- Knowledge is not understanding
- Understanding is not wisdom.
  - Clifford Stoll

OBJECTIVES

- Some common uses of database systems.
- Characteristics of file-based systems.
- Problems with file-based approach.
- Meaning of the term database.
- Meaning of the term Database Management System (DBMS).

OBJECTIVES

- Typical functions of a DBMS.
- Major components of the DBMS environment.
- Personnel involved in the DBMS environment.
- History of the development of DBMSs.
- Advantages and disadvantages of DBMSs.

EXAMPLES OF DATABASE APPLICATIONS

- Purchases from the supermarket
- Purchases using your credit card
- Booking a holiday at the travel agents
- Using the local library
- Taking out insurance
- Renting a video
- Using the Internet
- Studying at university

FILE-BASED SYSTEMS

- Collection of application programs that perform services for the end users (e.g. reports).
- Each program defines and manages its own data.

FILE-BASED PROCESSING

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo
PA14	16 Holhead Rd	Aberdeen	AB7 5SU	House	6	650	CO46
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93

PrivateOwner

ownerNo	fName	lName	address	telNo
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Client

clientNo	fName	lName	address	telNo	prefType	maxRent
CR76	John	Kay	56 High St, London SW1 4EH	0207-774-5632	Flat	425
CR56	Aline	Stewart	64 Fern Dr, Glasgow G42 0BL	0141-848-1825	Flat	350
CR74	Mike	Ritchie	18 Tain St, PA1G 1YQ	01475-392178	House	750
CR62	Mary	Tregear	5 Tarbot Rd, Aberdeen AB9 3ST	01224-196720	Flat	600

FILE-BASED PROCESSING

Lease

leaseNo	propertyNo	clientNo	rent	payment Method	deposit	paid	rentStart	rentFinish	duration
10024	PA14	CR62	650	Visa	1300	Y	1-Jun-13	31-May-14	12
10075	PL94	CR76	400	Cash	800	N	1-Aug-13	31-Jan-14	6
10012	PG21	CR74	600	Cheque	1200	Y	1-Jul-13	30-Jun-14	12

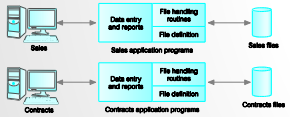
PropertyForRent

propertyNo	street	city	postcode	rent
PA14	16 Holhead	Aberdeen	AB7 5SU	650
PL94	6 Argyll St	London	NW2	400
PG21	18 Dale Rd	Glasgow	G12	600

Client

clientNo	fName	lName	address	telNo
CR76	John	Kay	56 High St, London SW1 4EH	0171-774-5632
CR74	Mike	Ritchie	18 Tain St, PA1G 1YQ	01475-392178
CR62	Mary	Tregear	5 Tarbot Rd, Aberdeen AB9 3ST	01224-196720

FILE-BASED PROCESSING



Sales Files
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)

Contracts Files
Lease (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish, duration)
PropertyForRent (propertyNo, street, city, postcode, rent)
Client (clientNo, fName, lName, address, telNo)

Sales Files
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)

Contracts Files
Lease (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish, duration)
PropertyForRent (propertyNo, street, city, postcode, rent)
Client (clientNo, fName, lName, address, telNo)

LIMITATIONS OF FILE-BASED APPROACH

- Separation and isolation of data
  - Each program maintains its own set of data.
  - Users of one program may be unaware of potentially useful data held by other programs.
- Duplication of data
  - Same data is held by different programs.
  - Wasted space and potentially different values and/or different formats for the same item.

LIMITATIONS OF FILE-BASED APPROACH

- Data dependence
  - File structure is defined in the program code.
- Incompatible file formats
  - Programs are written in different languages, and so cannot easily access each other's files.
- Fixed Queries/Proliferation of application programs
  - Programs are written to satisfy particular functions.
  - Any new requirement needs a new program.

DATABASE APPROACH

- Arose because:
  - Definition of data was embedded in application programs, rather than being stored separately and independently.
  - No control over access and manipulation of data beyond that imposed by application programs.
- Result:
  - the database and Database Management System (DBMS).

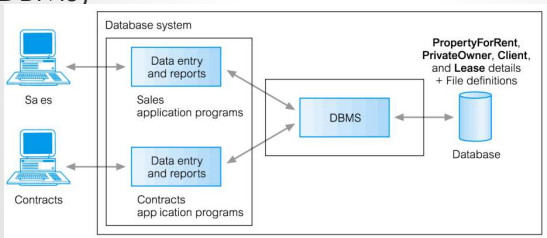
DATABASE

- Shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.
- System catalog (metadata) provides description of data to enable program–data independence.
- Logically related data comprises entities, attributes, and relationships of an organization's information.

DATABASE MANAGEMENT SYSTEM (DBMS)

- A software system that enables users to define, create, maintain, and control access to the database.
- (Database) application program: a computer program that interacts with database by issuing an appropriate request (SQL statement) to the DBMS.

DATABASE MANAGEMENT SYSTEM (DBMS)



PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fName, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentF nish)

DATABASE APPROACH

- Data definition language (DDL).
  - Permits specification of data types, structures and any data constraints.
  - All specifications are stored in the database.
- Data manipulation language (DML).
  - General enquiry facility (query language) of the data.

DATABASE APPROACH

- Controlled access to database may include:
  - a security system
  - an integrity system
  - a concurrency control system
  - a recovery control system
  - a user-accessible catalog.

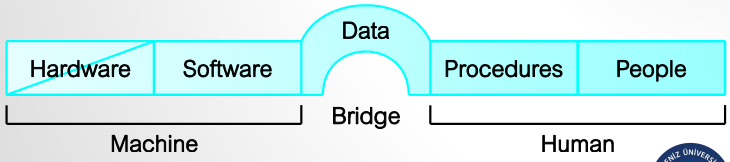
VIEWS

- Allows each user to have his or her own view of the database.
- A view is essentially some subset of the database.

VIEWS - BENEFITS

- Reduce complexity
- Provide a level of security
- Provide a mechanism to customize the appearance of the database
- Present a consistent, unchanging picture of the structure of the database, even if the underlying database is changed

COMPONENTS OF DBMS ENVIRONMENT



COMPONENTS OF DBMS ENVIRONMENT

- Hardware
  - Can range from a PC to a network of computers.
- Software
  - DBMS, operating system, network software (if necessary) and also the application programs.
- Data
  - Used by the organization and a description of this data called the schema.
- Procedures
  - Instructions and rules that should be applied to the design and use of the database and DBMS.
- People

ROLES IN THE DATABASE ENVIRONMENT

- Data Administrator (DA)
- Database Administrator (DBA)
- Database Designers (Logical and Physical)
- Application Programmers
- End Users (naive and sophisticated)

HISTORY OF DATABASE SYSTEMS

- First-generation
  - Hierarchical and Network
- Second generation
  - Relational
- Third generation
  - Object-Relational
  - Object-Oriented

ADVANTAGES OF DBMSS

- Control of data redundancy
- Data consistency
- More information from the same amount of data
- Sharing of data
- Improved data integrity
- Improved security
- Enforcement of standards
- Economy of scale

ADVANTAGES OF DBMSS

- Balance conflicting requirements
- Improved data accessibility and responsiveness
- Increased productivity
- Improved maintenance through data independence
- Increased concurrency
- Improved backup and recovery services

DISADVANTAGES OF DBMSS

- Complexity
- Size
- Cost of DBMS
- Additional hardware costs
- Cost of conversion
- Performance
- Higher impact of a failure

ASSIGNMENT 1

- Using a language of your choice we will manipulate a file that holds information for courses. The file has the following format separated by commas
  - Course Department
  - Course Code (number)
  - Course Title
  - Course AKTS
- The operations you must implement are as follows:
  - Get a list of courses in the file (just department and code concatenated)
  - Add a new course (takes the parameters above in order)
  - Removes a course (only takes the department and course code as parameters)
  - Changes the AKTS for a course (takes department, course code, and new AKTS as parameters)
  - Changes the course code (takes department, current course code, and new course code as parameters)

CSE 204 - INTRO TO DATABASE SYSTEMS RELATIONAL MODEL

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- Terminology of relational model.
- How tables are used to represent data.
- Connection between mathematical relations and relations in the relational model.
- Properties of database relations.
- How to identify CK, PK, and FKs.
- Meaning of entity integrity and referential integrity.
- Purpose and advantages of views.

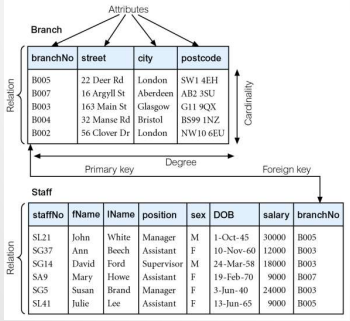
RELATIONAL MODEL TERMINOLOGY

- A relation is a table with columns and rows.
  - Only applies to logical structure of the database, not the physical structure.
- Attribute is a named column of a relation.
- Domain is the set of allowable values for one or more attributes.

RELATIONAL MODEL TERMINOLOGY

- Tuple is a row of a relation.
- Degree is the number of attributes in a relation.
- Cardinality is the number of tuples in a relation.
- Relational Database is a collection of normalized relations with distinct relation names.

INSTANCES OF BRANCH AND STAFF RELATIONS



EXAMPLES OF ATTRIBUTE DOMAINS

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

ALTERNATIVE TERMINOLOGY FOR RELATIONAL MODEL

Formal terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

MATHEMATICAL DEFINITION OF RELATION

- Consider two sets, D_1 & D_2, where $D_1 = \{2, 4\}$ and $D_2 = \{1, 3, 5\}$.
- Cartesian product, $D_1 \times D_2$, is set of all ordered pairs, where first element is member of D_1 and second element is member of D_2.

  $D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$
- Alternative way is to find all combinations of elements with first from D_1 and second from D_2.

MATHEMATICAL DEFINITION OF RELATION

- Any subset of Cartesian product is a relation; e.g.

  $R = \{(2, 1), (4, 1)\}$
- May specify which pairs are in relation using some condition for selection; e.g.
  - second element is 1:

    $R = \{(x, y) \mid x \in D_1, y \in D_2, \text{and } y = 1\}$
  - first element is always twice the second:

    $S = \{(x, y) \mid x \in D_1, y \in D_2, \text{and } x = 2y\}$

MATHEMATICAL DEFINITION OF RELATION

- Consider three sets D_1, D_2, D_3 with Cartesian Product

  $D_1 \times D_2 \times D_3$;

e.g.

$D_1 = \{1, 3\}$ $D_2 = \{2, 4\}$ $D_3 = \{5, 6\}$

$D_1 \times D_2 \times D_3 = \{(1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6)\}$

- Any subset of these ordered triples is a relation.

MATHEMATICAL DEFINITION OF RELATION

- Cartesian product of *n* sets $(D_1, D_2, \ldots, D_n)$ is:
- $D_1 \times D_2 \times \ldots \times D_n = \{(d_1, d_2, \ldots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \ldots, d_n \in D_n\}$

  usually written as:

  $$\prod_{i=1}^{n} D_i$$
- Any set of n-tuples from this Cartesian product is a relation on the n sets.

DATABASE RELATIONS

- Relation schema
  - Named relation defined by a set of attribute and domain name pairs.
- Relational database schema
  - Set of relation schemas, each with a distinct name.

PROPERTIES OF RELATIONS

- Relation name is distinct from all other relation names in relational schema.
- Each cell of relation contains exactly one atomic (single) value.
- Each attribute has a distinct name.
- Values of an attribute are all from the same domain.

PROPERTIES OF RELATIONS

- Each tuple is distinct; there are no duplicate tuples.
- Order of attributes has no significance.
- Order of tuples has no significance, theoretically.

RELATIONAL KEYS

- Superkey
  - An attribute, or set of attributes, that uniquely identifies a tuple within a relation.
- Candidate Key
  - Superkey (K) such that no proper subset is a superkey within the relation.
  - In each tuple of R, values of K uniquely identify that tuple (uniqueness).
  - No proper subset of K has the uniqueness property (irreducibility).

RELATIONAL KEYS

- Primary Key
  - Candidate key selected to identify tuples uniquely within relation.
- Alternate Keys
  - Candidate keys that are not selected to be primary key.
- Foreign Key
  - Attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation.

INTEGRITY CONSTRAINTS

- Null
  - Represents value for an attribute that is currently unknown or not applicable for tuple.
  - Deals with incomplete or exceptional data.
  - Represents the absence of a value and is not the same as zero or spaces, which are values.

INTEGRITY CONSTRAINTS

- Entity Integrity
  - In a base relation, no attribute of a primary key can be null.
- Referential Integrity
  - If foreign key exists in a relation, either foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null.

INTEGRITY CONSTRAINTS

- General Constraints
  - Additional rules specified by users or database administrators that define or constrain some aspect of the enterprise.

VIEWS

- Base Relation
  - Named relation corresponding to an entity in conceptual schema, whose tuples are physically stored in database.
- View
  - Dynamic result of one or more relational operations operating on base relations to produce another relation.

VIEWS

- A virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.
- Contents of a view are defined as a query on one or more base relations.
- Views are dynamic, meaning that changes made to base relations that affect view attributes are immediately reflected in the view.

PURPOSE OF VIEWS

- Provides powerful and flexible security mechanism by hiding parts of database from certain users.
- Permits users to access data in a customized way, so that same data can be seen by different users in different ways, at same time.
- Can simplify complex operations on base relations.

UPDATING VIEWS

- All updates to a base relation should be immediately reflected in all views that reference that base relation.
- If view is updated, underlying base relation should reflect change.

UPDATING VIEWS

- There are restrictions on types of modifications that can be made through views:
- Updates are allowed if query involves a single base relation and contains a candidate key of base relation.
- Updates are not allowed involving multiple base relations.
- Updates are not allowed involving aggregation or grouping operations.

UPDATING VIEWS

- Classes of views are defined as:
  - theoretically not updateable;
  - theoretically updateable;
  - partially updateable.

CSE 204 - INTRO TO DATABASE SYSTEMS SQL

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- Purpose and importance of SQL.
- How to retrieve data from database using SELECT and:
  - Use compound WHERE conditions.
  - Sort query results using ORDER BY.
  - Use aggregate functions.
  - Group data using GROUP BY and HAVING.
  - Use subqueries.
  - Join tables together.
  - Perform set operations (UNION, INTERSECT, EXCEPT).
- How to update database using INSERT, UPDATE, and DELETE.

OBJECTIVES OF SQL

- Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.
- Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.
- It must be portable.

OBJECTIVES OF SQL

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- Until SQL:1999, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.

OBJECTIVES OF SQL

- SQL is relatively easy to learn:
  - it is non-procedural - you specify what information you require, rather than how to get it;
  - it is essentially free-format.
  - Consists of standard English words:

```
1. CREATE TABLE Staff(staffNo VARCHAR(5),
lName VARCHAR(15),
salary DECIMAL(7,2));
2. INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
3. SELECT staffNo, lName, salary
   FROM Staff
   WHERE salary > 10000;
```


OBJECTIVES OF SQL

- Can be used by range of users including DBAs, management, application developers, and other types of end users.
- An ISO standard now exists for SQL, making it both the formal and de facto standard language for relational databases.

HISTORY OF SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced 'see-quel', though official pronunciation is 'S-Q-L'.
- IBM subsequently produced a prototype DBMS called System R, based on SEQUEL/2.
- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.

HISTORY OF SQL

- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- In 1987, ANSI and ISO published an initial standard for SQL.
- In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released.
- In summer 2008, SQL:2008 was released.
- In late 2011, SQL:2011 was released.

IMPORTANCE OF SQL

- SQL has become part of application architectures such as IBM's Systems Application Architecture.
- It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
- SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
  - ISO's Information Resource Directory System (IRDS) Standard
  - Remote Data Access (RDA) Standard.

WRITING SQL COMMANDS

- SQL statement consists of reserved words and user-defined words.

❑Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.

❑User-defined words are made up by user and represent names of various database objects such as relations, columns, views.

WRITING SQL COMMANDS

- Most components of an SQL statement are case insensitive, except for literal character data.
- More readable with indentation and lineation:
  - Each clause should begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause.

WRITING SQL COMMANDS

- Use extended form of BNF notation:
  - ❑ Upper-case letters represent reserved words.
  - ❑ Lower-case letters represent user-defined words.
  - ❑ | indicates a choice among alternatives.
  - ❑ Curly braces indicate a required element.
  - ❑ Square brackets indicate an optional element.
  - ❑ … indicates optional repetition (0 or more).

LITERALS

- Literals are constants used in SQL statements.
- All non-numeric literals must be enclosed in single quotes (e.g. 'London').
- All numeric literals must not be enclosed in quotes (e.g. 650.00).

SELECT STATEMENT

```
SELECT [DISTINCT | ALL]
     {* | [columnExpression [AS newName]] [,...] }
FROM      TableName [alias] [, ...]
[WHERE    condition]
[GROUP BY columnList]  [HAVING     condition]
[ORDER BY columnList]
```


SELECT STATEMENT

- SELECT Specifies which columns are to appear in output.
- FROM Specifies table(s) to be used.
- WHERE Filters rows.
- GROUP BY Forms groups of rows with same column value.
- HAVING Filters groups subject to some condition.
- ORDER BY Specifies the order of the output.
- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.

EXAMPLE – ALL COLUMNS, ALL ROWS

- List full details of all staff.

```
SELECT staffNo, fName, lName, address,
          position, sex, DOB, salary, branchNo
     FROM Staff;
```

- Can use * as an abbreviation for 'all columns':

```
SELECT *
FROM Staff;
```


EXAMPLE – ALL COLUMNS, ALL ROWS

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

EXAMPLE 2 – SPECIFIC COLUMNS, ALL ROWS

- Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff;
```


EXAMPLE 2 – SPECIFIC COLUMNS, ALL ROWS

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

EXAMPLE 3 – USE OF DISTINCT

- List the property numbers of all properties that have been viewed.

```
SELECT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG4
PA14
PG36

EXAMPLE 3 – USE OF DISTINCT

- Use DISTINCT to eliminate duplicates:

```
SELECT DISTINCT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG36

EXAMPLE 4 – CALCULATED FIELDS

- Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

```
SELECT staffNo, fName, lName, salary/12
FROM Staff;
```

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

EXAMPLE 4 – CALCULATED FIELDS

- To name column, use AS clause:

```
SELECT staffNo, fName, lName, salary/12 AS monthlySalary
FROM Staff;
```


EXAMPLE 5 – COMPARISON SEARCH

- List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

EXAMPLE 6 – COMPOUND COMPARISON SEARCH

- List addresses of all branch offices in London or Glasgow.

```
SELECT *
FROM Branch
WHERE city = 'London' OR city = 'Glasgow';
```

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

EXAMPLE 7 – RANGE SEARCH

- List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- BETWEEN test includes the endpoints of range.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

EXAMPLE 7 – RANGE SEARCH

- Also a negated version NOT BETWEEN.
- BETWEEN does not add much to SQL's expressive power. Could also write:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary>=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

EXAMPLE 8 – SET MEMBERSHIP

- List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

EXAMPLE 8 – SET MEMBERSHIP

- There is a negated version (NOT IN).
- IN does not add much to SQL's expressive power. Could have expressed this as:

```
SELECT staffNo, fName, lName, position
   FROM Staff
WHERE position='Manager' OR
      position='Supervisor';
```

- IN is more efficient when set contains many values.

EXAMPLE 9 – PATTERN MATCHING

- Find all owners with the string 'Glasgow' in their address.

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE '%Glasgow%';
```

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

EXAMPLE 9 – PATTERN MATCHING

- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
- LIKE '%Glasgow%' means a sequence of characters of any length containing 'Glasgow'.

EXAMPLE 10 – NULL SEARCH CONDITION

- List details of all viewings on property PG4 where a comment has not been supplied.
- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword IS NULL:

```
SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo = 'PG4' AND
                    comment IS NULL;
```

EXAMPLE 10 – NULL SEARCH CONDITION

clientNo	viewDate
CR56	26-May-04

- Negated version (IS NOT NULL) can test for non-null values.

EXAMPLE 11 – ORDERING (SINGLE COLUMN)

- List salaries for all staff, arranged in descending order of salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff
ORDER BY salary DESC;
```

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

EXAMPLE 12 – ORDERING (MULTIPLE COLUMN)

- Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type;
```

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

EXAMPLE 12 – ORDERING (MULTIPLE COLUMN)

- Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify minor order:

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type, rent DESC;
```

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

SELECT STATEMENT AGGREGATES

- ISO standard defines five aggregate functions:
  - COUNT returns number of values in specified column.
  - SUM returns sum of values in specified column.
  - AVG returns average of values in specified column.
  - MIN returns smallest value in specified column.
  - MAX returns largest value in specified column.

SELECT STATEMENT AGGREGATES

- Each operates on a single column of a table and returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.
- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.

SELECT STATEMENT AGGREGATES

- Aggregate functions can be used only in SELECT list and in HAVING clause.
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

EXAMPLE 13 – USE OF COUNT(*)

- How many properties cost more than 350 per month to rent?

```
SELECT COUNT(*) AS myCount
    FROM PropertyForRent
    WHERE rent > 350;
```

myCount
5

EXAMPLE 14 – USE OF COUNT(DISTINCT)

- How many different properties viewed in May '13?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-13'
          AND '31-May-13';
```

myCount
2

EXAMPLE 16 – USE OF MIN, MAX, AVG

- Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin,
       MAX(salary) AS myMax,
       AVG(salary) AS myAvg
FROM Staff;
```

myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT STATEMENT GROUPING

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be single-valued per group, and SELECT clause may only contain:
  - column names
  - aggregate functions
  - constants
  - expression involving combinations of the above.

SELECT STATEMENT GROUPING

- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

EXAMPLE 17 – USE OF GROUP BY

- Find number of staff in each branch and their total salaries.

```
SELECT branchNo,
                COUNT(staffNo) AS myCount,
       SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

RESTRICTED GROUPINGS – HAVING CLAUSE

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

EXAMPLE 18 – USE OF HAVING

- For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,
      COUNT(staffNo) AS myCount,
  SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

SUBQUERIES

- Some SQL statements can have a SELECT embedded within them.
- A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a subquery or nested query.
- Subselects may also appear in INSERT, UPDATE, and DELETE statements.

EXAMPLE 19 – SUBQUERY WITH EQUALITY

- List staff who work in branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo =
          (SELECT branchNo
           FROM Branch
           WHERE street = '163 Main St');
```

EXAMPLE 19 – SUBQUERY WITH EQUALITY

- Inner SELECT finds branch number for branch at '163 Main St' ('B003').
- Outer SELECT then retrieves details of all staff who work at this branch.
- Outer SELECT then becomes:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```


EXAMPLE 19 – SUBQUERY WITH EQUALITY

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

EXAMPLE 20 – SUBQUERY WITH AGGREGATE

- List all staff whose salary is greater than the average salary, and show by how much.

```
SELECT staffNo, fName, lName, position,
  salary – (SELECT AVG(salary) FROM Staff) As
salDiff
FROM Staff
WHERE salary >
(SELECT AVG(salary)
 FROM Staff);
```


EXAMPLE 20 – SUBQUERY WITH AGGREGATE

- Cannot write 'WHERE salary > AVG(salary)'
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, fName, lName, position,
     salary – 17000 As salDiff
FROM Staff
WHERE salary > 17000;
```

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

SUBQUERY RULES

- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).
- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.
- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an alias.
- When subquery is an operand in a comparison, subquery must appear on right-hand side.
- A subquery may not be used as an operand in an expression.

EXAMPLE 21 – NESTED SUBQUERY: USE OF IN

- List properties handled by staff at '163 Main St'.

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN
(SELECT staffNo
 FROM Staff
 WHERE branchNo =
(SELECT branchNo
 FROM Branch
 WHERE street = '163 Main St'));
```

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

ANY AND ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by all values produced by subquery.
- With ANY, condition will be true if it is satisfied by any values produced by subquery.
- If subquery is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

EXAMPLE 22 – USE OF ANY/SOME

- Find staff whose salary is larger than salary of at least one member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME
          (SELECT salary
           FROM Staff
           WHERE branchNo = 'B003');
```


EXAMPLE 22 – USE OF ANY/SOME

- Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

EXAMPLE 23 – USE OF ALL

- Find staff whose salary is larger than salary of every member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL
      (SELECT salary
       FROM Staff
       WHERE branchNo = 'B003');
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

MULTI-TABLE QUERIES

- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).

MULTI-TABLE QUERIES

- Also possible to use an alias for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.

EXAMPLE 24 – SIMPLE JOIN

- List names of all clients who have viewed a property along with any comment supplied.

```
SELECT c.clientNo, fName, lName,
            propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```


EXAMPLE 24 – SIMPLE JOIN

- Only those rows from both tables that have identical values in the clientNo columns (c.clientNo = v.clientNo) are included in result.
- Equivalent to equi-join in relational algebra.

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

ALTERNATIVE JOIN CONSTRUCTS

- SQL provides alternative ways to specify joins:

```
FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
FROM Client JOIN Viewing USING clientNo
FROM Client NATURAL JOIN Viewing
```

- In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.

EXAMPLE 25 – SORTING A JOIN

- For each branch, list numbers and names of staff who manage properties, and properties they manage.

```
SELECT s.branchNo, s.staffNo, fName, lName,
            propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```


EXAMPLE 25 – SORTING A JOIN

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

EXAMPLE 26 – THREE TABLE JOIN

- For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,
       propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND
      s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

EXAMPLE 26 – THREE TABLE JOIN

branchNo	city	staffNo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

- Alternative formulation for FROM and WHERE:
- FROM (Branch b JOIN Staff s USING branchNo) AS
- bs JOIN PropertyForRent p USING staffNo

EXAMPLE 27 – MULTIPLE GROUPING COLUMNS

- Find number of properties handled by each staff member.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```


EXAMPLE 27 – MULTIPLE GROUPING COLUMNS

branchNo	staffNo	myCount
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

COMPUTING A JOIN

- Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in FROM clause.
2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort result table as required.
6. SQL provides special format of SELECT for Cartesian product:

```
SELECT    [DISTINCT | ALL]    {* | columnList}
FROM Table1 CROSS JOIN Table2
```


OUTER JOIN

- If one row of a joined table is unmatched, row is omitted from result table.
- Outer join operations retain rows that do not satisfy the join condition.
- Consider following tables:

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

OUTER JOIN

- The (inner) join of these two tables:

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

OUTER JOIN

- Result table has two rows where cities are same.
- There are no rows corresponding to branches in Bristol and Aberdeen.
- To include unmatched rows in result table, use an Outer join.

EXAMPLE 28 – LEFT OUTER JOIN

- List branches and properties that are in same city along with any unmatched branches.

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN
     PropertyForRent1 p ON b.bCity = p.pCity;
```

- Includes those rows of first (left) table unmatched with rows from second (right) table.
- Columns from second table are filled with NULLs.

EXAMPLE 28 – LEFT OUTER JOIN

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

EXAMPLE 29 – RIGHT OUTER JOIN

- List branches and properties in same city and any unmatched properties.

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN
      PropertyForRent1 p ON b.bCity = p.pCity;
```

- Right Outer join includes those rows of second (right) table that are unmatched with rows from first (left) table.
- Columns from first table are filled with NULLs.

EXAMPLE 29 – RIGHT OUTER JOIN

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

EXAMPLE 30 – FULL OUTER JOIN

- List branches and properties in same city and any unmatched branches or properties.

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN
      PropertyForRent1 p ON b.bCity = p.pCity;
```

- Includes rows that are unmatched in both tables.
- Unmatched columns are filled with NULLs.

EXAMPLE 30 – FULL OUTER JOIN

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

EXISTS AND NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries.
- Produce a simple true/false result.
- True if and only if there exists at least one row in result table returned by subquery.
- False if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.
- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
- Common for subqueries following (NOT) EXISTS to be of form:
- `(SELECT * ...)`

EXAMPLE 31 – QUERY USING EXISTS

- Find all staff who work in a London branch.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
      (SELECT *
       FROM Branch b
       WHERE s.branchNo = b.branchNo AND
             city = 'London');
```

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

EXAMPLE 31 – QUERY USING EXISTS

- Note, search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff.
- If omitted, would get all staff records listed out because subquery:

```
SELECT * FROM Branch WHERE city='London'
```

- would always be true and query would be:

```
SELECT staffNo, fName, lName, position FROM Staff
WHERE true;
```


EXAMPLE 31 – QUERY USING EXISTS

- Could also write this query using join construct:

```
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
                   city = 'London';
```

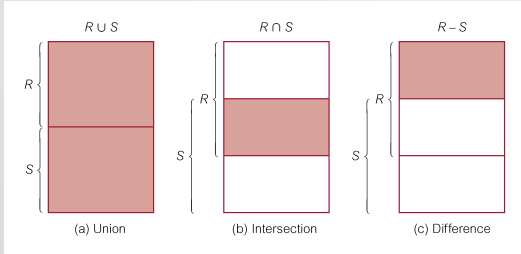

UNION, INTERSECT, AND DIFFERENCE (EXCEPT)

- Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into a single result table.
- Union of two tables, A and B, is table containing all rows in either A or B or both.
- Intersection is table containing all rows common to both A and B.
- Difference is table containing all rows in A but not in B.
- Two tables must be union compatible.

UNION, INTERSECT, AND DIFFERENCE (EXCEPT)

- Format of set operator clause in each case is:
- `op [ALL] [CORRESPONDING [BY {column1 [, ...]}]]`
- If CORRESPONDING BY specified, set operation performed on the named column(s).
- If CORRESPONDING specified but not BY clause, operation performed on common columns.
- If ALL specified, result can include duplicate rows.

UNION, INTERSECT, AND DIFFERENCE (EXCEPT)



EXAMPLE 32 – USE OF UNION

- List all cities where there is either a branch office or a property.

```
(SELECT city
FROM Branch
WHERE city IS NOT NULL) UNION
(SELECT city
FROM PropertyForRent
WHERE city IS NOT NULL);
```


EXAMPLE 32 – USE OF UNION

- Or

```
(SELECT *
FROM Branch
WHERE city IS NOT NULL)
UNION CORRESPONDING BY city
(SELECT *
FROM PropertyForRent
WHERE city IS NOT NULL);
```


EXAMPLE 32 – USE OF UNION

- Produces result tables from both queries and merges both tables together.

city
London
Glasgow
Aberdeen
Bristol

EXAMPLE 33 – USE OF INTERSECT

- List all cities where there is both a branch office and a property.

```
(SELECT city FROM Branch)
INTERSECT
(SELECT city FROM PropertyForRent);
```


EXAMPLE 33 – USE OF INTERSECT

- Or

```
(SELECT * FROM Branch)
INTERSECT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

city
Aberdeen
Glasgow
London

EXAMPLE 33 – USE OF INTERSECT

- Could rewrite this query without INTERSECT operator:

```
SELECT b.city
FROM Branch b PropertyForRent p
WHERE b.city = p.city;
```

- Or:

```
SELECT DISTINCT city FROM Branch b
WHERE EXISTS
(SELECT * FROM PropertyForRent p
WHERE p.city = b.city);
```


EXAMPLE 34 – USE OF EXCEPT

- List of all cities where there is a branch office but no properties.

```
(SELECT city FROM Branch)
EXCEPT
(SELECT city FROM PropertyForRent);
```

city
Bristol

- Or

```
(SELECT * FROM Branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

EXAMPLE 34 – USE OF EXCEPT

- Could rewrite this query without EXCEPT:

```
SELECT DISTINCT city FROM Branch
WHERE city NOT IN
(SELECT city FROM PropertyForRent);
```

- Or

```
SELECT DISTINCT city FROM Branch b
WHERE NOT EXISTS
(SELECT * FROM PropertyForRent p
WHERE p.city = b.city);
```


INSERT

```
INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
- dataValueList must match columnList as follows:
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in dataValueList must be compatible with data type of corresponding column.

INSERT

```
INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.

EXAMPLE 35 – INSERT INTO … VALUES

- Insert a new row into Staff table supplying data for all column

```
INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M',
Date'1957-05-25', 8300, 'B003');
```


EXAMPLE 36 – INSERT USING DEFAULTS

- Insert a new row into Staff table supplying data for all mandatory columns.

```
INSERT INTO Staff (staffNo, fName, lName,
                   position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones',
                   'Assistant', 8100, 'B003');
```

- Or

```
INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL,
                    NULL, 8100, 'B003');
```


INSERT…SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO TableName [ (columnList) ]
SELECT ...
```


EXAMPLE 37 – INSERT … SELECT

- Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:
- `StaffPropCount(staffNo, fName, lName, propCnt)`
- Populate StaffPropCount using Staff and PropertyForRent tables.

EXAMPLE 37 – INSERT … SELECT

```
INSERT INTO StaffPropCount
(SELECT s.staffNo, fName, lName, COUNT(*)
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.staffNo, fName, lName)
UNION
(SELECT staffNo, fName, lName, 0
FROM Staff
WHERE staffNo NOT IN
(SELECT DISTINCT staffNo
     FROM PropertyForRent));
```


EXAMPLE 37 – INSERT … SELECT

staffNo	fName	lName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

- If second part of UNION is omitted, excludes those staff who currently do not manage any properties.

UPDATE

```
UPDATE TableName
SET columnName1 = dataValue1
        [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

- TableName can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

UPDATE

- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - if specified, only those rows that satisfy searchCondition are updated.
- New dataValue(s) must be compatible with data type for corresponding column.

EXAMPLE 38/39 – UPDATE

- Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary * 1.03;
```

- Give all Managers a 5% pay increase.

```
UPDATE Staff
SET salary = salary * 1.05
WHERE position = 'Manager';
```


EXAMPLE 40 – UPDATE

- Promote David Ford (staffNo='SG14') to Manager and change his salary to £18,000.

```
UPDATE Staff
SET position = 'Manager', salary = 18000
WHERE staffNo = 'SG14';
```


DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- TableName can be name of a base table or an updatable view.
- searchCondition is optional; if omitted, all rows are deleted from table. This does not delete table. If search_condition is specified, only those rows that satisfy condition are deleted.

EXAMPLE 41/42 – DELETE

- Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
WHERE propertyNo = 'PG4';
```

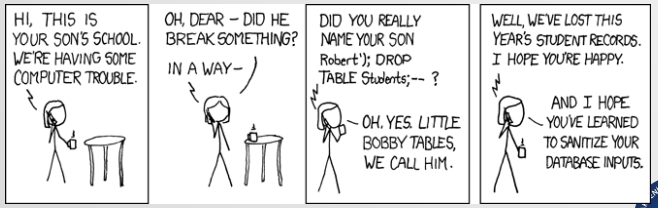
- Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```


CSE 204 - INTRO TO DATABASE SYSTEMS ADVANCED SQL

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

QUOTE OF THE DAY



OUTLINE

- Data types supported by SQL standard.
- Purpose of integrity enhancement feature of SQL.
- How to define integrity constraints using SQL.
- How to use the integrity enhancement feature in the CREATE and ALTER TABLE statements.
- Views
  - Purpose of views.
  - How to create and delete views using SQL.
  - How the DBMS performs operations on views.
  - Under what conditions views are updatable.
  - Advantages and disadvantages of views.
- How the ISO transaction model works.
- How to use the GRANT and REVOKE statements as a level of security.

ISO SQL DATA TYPES

DATA TYPE	DECLARATIONS				
boolean	BOOLEAN				
character	CHAR	VARCHAR			
bit†	BIT	BIT VARYING			
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT	BIGINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION		
datetime	DATE	TIME	TIMESTAMP		
interval	INTERVAL				
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT		

†BIT and BIT VARYING have been removed from the SQL:2003 standard.

INTEGRITY ENHANCEMENT FEATURE

- Consider five types of integrity constraints:
  - required data
  - domain constraints
  - entity integrity
  - referential integrity
  - general constraints.

INTEGRITY ENHANCEMENT FEATURE

- Required Data

```
position  VARCHAR(10)    NOT NULL
```

- Domain Constraints
- CHECK

```
sex  CHAR NOT NULL
          CHECK (sex IN ('M', 'F'))
```


INTEGRITY ENHANCEMENT FEATURE

- CREATE DOMAIN

```
CREATE DOMAIN DomainName [AS] dataType
      [DEFAULT defaultOption]
      [CHECK (searchCondition)]
```

- For example:

```
CREATE DOMAIN SexType AS CHAR
     CHECK (VALUE IN ('M', 'F'));
sex  SexType   NOT NULL
```


INTEGRITY ENHANCEMENT FEATURE

- searchCondition can involve a table lookup:

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo
                 FROM Branch));
```

- Domains can be removed using DROP DOMAIN:

```
DROP DOMAIN DomainName
     [RESTRICT | CASCADE]
```


IEF - ENTITY INTEGRITY

- Primary key of a table must contain a unique, non-null value for each row.
- ISO standard supports FOREIGN KEY clause in CREATE and ALTER TABLE statements:

```
PRIMARY KEY(staffNo)
PRIMARY KEY(clientNo, propertyNo)
```

- Can only have one PRIMARY KEY clause per table. Can still ensure uniqueness for alternate keys using UNIQUE:

```
UNIQUE(telNo)
```


IEF - REFERENTIAL INTEGRITY

- FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK.
- Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.
- ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

```
FOREIGN KEY(branchNo) REFERENCES Branch
```


IEF - REFERENTIAL INTEGRITY

- Any INSERT/UPDATE attempting to create FK value in child table without matching CK value in parent is rejected.
- Action taken attempting to update/delete a CK value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

```
CASCADE          -  SET NULL
SET DEFAULT      -  NO ACTION
```


IEF - REFERENTIAL INTEGRITY

- **CASCADE**: Delete row from parent and delete matching rows in child, and so on in cascading manner.
- **SET NULL**: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.
- **SET DEFAULT**: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
- **NO ACTION**: Reject delete from parent. Default.

```
FOREIGN KEY (staffNo) REFERENCES Staff                ON
DELETE SET NULL

FOREIGN KEY (ownerNo) REFERENCES Owner      ON   UPDATE
CASCADE
```

IEF - GENERAL CONSTRAINTS

- Could use CHECK/UNIQUE in CREATE and ALTER TABLE.
- Similar to the CHECK clause, also have:

```
CREATE ASSERTION AssertionName
CHECK (searchCondition)

CREATE ASSERTION StaffNotHandlingTooMuch
CHECK (NOT EXISTS    (SELECT staffNo
FROM PropertyForRent
GROUP BY staffNo
HAVING COUNT(*) > 100))
```

DATA DEFINITION

- SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.
- Main SQL DDL statements are:

```
CREATE SCHEMA   DROP SCHEMA
CREATE/ALTER DOMAIN DROP DOMAIN
CREATE/ALTER TABLE   DROP TABLE
CREATE VIEW     DROP VIEW
```

- Many DBMSs also provide:

```
CREATE INDEX    DROP INDEX
```

DATA DEFINITION

- Relations and other database objects exist in an environment.
- Each environment contains one or more catalogs, and each catalog consists of set of schemas.
- Schema is named collection of related database objects.
- Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.

CREATE SCHEMA

```
CREATE SCHEMA [Name |
          AUTHORIZATION CreatorId ]
DROP SCHEMA Name [RESTRICT | CASCADE ]
```

- With RESTRICT (default), schema must be empty or operation fails.
- With CASCADE, operation cascades to drop all objects associated with schema in order defined above. If any of these operations fail, DROP SCHEMA fails.

CREATE TABLE

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] […,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,…]}
 {[CHECK (searchCondition)] [,…] })
```

CREATE TABLE

- Creates a table with one or more columns of the specified dataType.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action.

EXAMPLE 1 - CREATE TABLE

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
CHECK (VALUE IN (SELECT ownerNo FROM PrivateOwner));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
CHECK (VALUE IN (SELECT staffNo FROM Staff));
CREATE DOMAIN PNumber AS VARCHAR(5);
CREATE DOMAIN PRooms AS SMALLINT;
CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PRent AS DECIMAL(6,2)
CHECK(VALUE BETWEEN 0 AND 9999.99);
```

EXAMPLE 1 - CREATE TABLE

```
CREATE TABLE PropertyForRent (
propertyNo     PNumber    NOT NULL, ….
rooms          PRooms     NOT NULL  DEFAULT 4,
rent          PRent  NOT NULL,      DEFAULT 600,
ownerNo    OwnerNumber    NOT NULL,
staffNo    StaffNumber
Constraint StaffNotHandlingTooMuch ….
branchNo  BranchNumber    NOT NULL,
PRIMARY KEY (propertyNo),
FOREIGN KEY (staffNo) REFERENCES Staff
              ON DELETE SET NULL ON UPDATE CASCADE ….);
```

ALTER TABLE

- Add a new column to a table.
- Drop a column from a table.
- Add a new table constraint.
- Drop a table constraint.
- Set a default for a column.
- Drop a default for a column.

EXAMPLE 2 - ALTER TABLE

- Change Staff table by removing default of 'Assistant' for position column and setting default for sex column to female ('F').

```
ALTER TABLE Staff
    ALTER position DROP DEFAULT;
ALTER TABLE Staff
    ALTER sex SET DEFAULT 'F';
```


EXAMPLE 2 - ALTER TABLE

- Remove constraint from PropertyForRent that staff are not allowed to handle more than 100 properties at a time. Add new column to Client table.

```
ALTER TABLE PropertyForRent
    DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
    ADD prefNoRooms PRooms;
```


DROP TABLE

```
DROP TABLE TableName [RESTRICT | CASCADE]
    e.g. DROP TABLE PropertyForRent;
```

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

VIEWS

- View
  - Dynamic result of one or more relational operations operating on base relations to produce another relation.
- Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.

VIEWS

- Contents of a view are defined as a query on one or more base relations.
- With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

SQL FOR CREATING VIEWS

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
AS subselect
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- Can assign a name to each column in view.
- If list of column names is specified, it must have same number of items as number of columns produced by subselect.
- If omitted, each column takes name of corresponding column in subselect.

SQL FOR CREATING VIEWS

- List must be specified if there is any ambiguity in a column name.
- The subselect is known as the defining query.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

EXAMPLE 3 - CREATE HORIZONTAL VIEW

- Create view so that manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
AS    SELECT *
      FROM Staff
      WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

EXAMPLE 4 - CREATE VERTICAL VIEW

- Create view of staff details at branch B003 excluding salaries.

```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
      FROM Staff
      WHERE branchNo = 'B003';
```

staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

EXAMPLE 5 - GROUPED AND JOINED VIEWS

- Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
      FROM Staff s, PropertyForRent p
      WHERE s.staffNo = p.staffNo
      GROUP BY s.branchNo, s.staffNo;
```

branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

DROPPING VIEWS

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- Causes definition of view to be deleted from database.
- For example:

```
DROP VIEW Manager3Staff;
```


DROPPING VIEWS

- With CASCADE, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- With RESTRICT (default), if any other objects depend for their existence on continued existence of view being dropped, command is rejected.

VIEW RESOLUTION

- Count number of properties managed by each member at branch B003.

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```


VIEW RESOLUTION

- View column names in SELECT list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

- View names in FROM are replaced with corresponding FROM lists of defining query:

```
FROM Staff s, PropertyForRent p
```


VIEW RESOLUTION

- WHERE from user query is combined with WHERE of defining query using AND:

```
WHERE s.staffNo = p.staffNo AND branchNo = 'B003'
```

- GROUP BY and HAVING clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

- ORDER BY copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```


VIEW RESOLUTION

- Final merged query is now executed to produce the result:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
                  branchNo = 'B003'
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```


VIEW RESTRICTIONS

SQL imposes several restrictions on creation and use of views.

- If column in view is based on an aggregate function:
  - Column may appear only in SELECT and ORDER BY clauses of queries that access view.
  - Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.

VIEW RESTRICTIONS

- For example, following query would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

- Similarly, following query would also fail:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

VIEW RESTRICTIONS

- Grouped view may never be joined with a base table or a view.
  - For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

VIEW UPDATABILITY

- All updates to base table reflected in all views that encompass base table.
- Similarly, may expect that if view is updated then base table(s) will reflect change.
- However, consider again view StaffPropCnt.
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

- Have to insert 2 records into PropertyForRent showing which properties SG5 manages. However, do not know which properties they are; i.e. do not know primary keys!

VIEW UPDATABILITY

- If change definition of view and replace count with actual property numbers:

```
CREATE VIEW StaffPropList (branchNo,
                           staffNo, propertyNo)
AS SELECT s.branchNo, s.staffNo, p.propertyNo
          FROM Staff s, PropertyForRent p
          WHERE s.staffNo = p.staffNo;
```


VIEW UPDATABILITY

- Now try to insert the record:

```
INSERT INTO StaffPropList
VALUES ('B003', 'SG5', 'PG19');
```

- Still problem, because in PropertyForRent all columns except postcode/staffNo are not allowed nulls.
- However, have no way of giving remaining non-null columns values.

VIEW UPDATABILITY

- ISO specifies that a view is updatable if and only if:
  - DISTINCT is not specified.
  - Every element in SELECT list of defining query is a column name and no column appears more than once.
  - FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.
  - No nested SELECT referencing outer table.
  - No GROUP BY or HAVING clause.
  - Also, every row added through view must not violate integrity constraints of base table.

UPDATABLE VIEW

- For view to be updatable, DBMS must be able to trace any row or column back to its row or column in the source table.

WITH CHECK OPTION

- Rows exist in a view because they satisfy WHERE condition of defining query.
- If a row changes and no longer satisfies condition, it disappears from the view.
- New rows appear within view when insert/update on view cause them to satisfy WHERE condition.
- Rows that enter or leave a view are called migrating rows.
- WITH CHECK OPTION prohibits a row migrating out of the view.

WITH CHECK OPTION

- LOCAL/CASCADED apply to view hierarchies.
- With LOCAL, any row insert/update on view and any view directly or indirectly defined on this view must not cause row to disappear from view unless row also disappears from derived view/table.
- With CASCADED (default), any row insert/ update on this view and on any view directly or indirectly defined on this view must not cause row to disappear from the view.

EXAMPLE 6 - WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
AS    SELECT *
FROM Staff
WHERE branchNo = 'B003'
WITH CHECK OPTION;
```

- Cannot update branch number of row B003 to B002 as this would cause row to migrate from view.
- Also cannot insert a row into view with a branch number that does not equal B003.

EXAMPLE 6 - WITH CHECK OPTION

- Now consider the following:

```
CREATE VIEW LowSalary
AS   SELECT * FROM Staff WHERE salary > 9000;
CREATE VIEW HighSalary
AS   SELECT * FROM LowSalary
WHERE salary > 10000
WITH LOCAL CHECK OPTION;
CREATE VIEW Manager3Staff
AS   SELECT * FROM HighSalary
WHERE branchNo = 'B003';
```

EXAMPLE 6 - WITH CHECK OPTION

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

- This update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary.
- However, if update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary.

EXAMPLE 6 - WITH CHECK OPTION

- If HighSalary had specified WITH CASCADED CHECK OPTION, setting salary to 9500 or 8000 would be rejected because row would disappear from HighSalary.
- To prevent anomalies like this, each view should be created using WITH CASCADED CHECK OPTION.

ADVANTAGES OF VIEWS

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

DISADVANTAGES OF VIEWS

- Update restriction
- Structure restriction
- Performance

VIEW MATERIALIZATION

- View resolution mechanism may be slow, particularly if view is accessed frequently.
- View materialization stores view as temporary table when view is first queried.
- Thereafter, queries based on materialized view can be faster than recomputing view each time.
- Difficulty is maintaining the currency of view while base tables(s) are being updated.

VIEW MAINTENANCE

- View maintenance aims to apply only those changes necessary to keep view current.
- Consider following view:

```
CREATE VIEW StaffPropRent(staffNo)
AS   SELECT DISTINCT staffNo
          FROM PropertyForRent
          WHERE branchNo = 'B003' AND
               rent > 400;
```

staffNo
SG37
SG14

VIEW MATERIALIZATION

- If insert row into PropertyForRent with rent <= 400 then view would be unchanged.
- If insert row for property PG24 at branch B003 with staffNo = SG19 and rent = 550, then row would appear in materialized view.
- If insert row for property PG54 at branch B003 with staffNo = SG37 and rent = 450, then no new row would need to be added to materialized view.
- If delete property PG24, row should be deleted from materialized view.
- If delete property PG54, then row for PG37 should not be deleted (because of existing property PG21).

TRANSACTIONS

- SQL defines transaction model based on COMMIT and ROLLBACK.
- Transaction is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.
- An SQL transaction automatically begins with a transaction-initiating SQL statement (e.g., SELECT, INSERT).
- Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.

TRANSACTIONS

- Transaction can complete in one of four ways:
  - COMMIT ends transaction successfully, making changes permanent.
  - ROLLBACK aborts transaction, backing out any changes made by transaction.
  - For programmatic SQL, successful program termination ends final transaction successfully, even if COMMIT has not been executed.
  - For programmatic SQL, abnormal program end aborts transaction.

TRANSACTIONS

- New transaction starts with next transaction-initiating statement.
- SQL transactions cannot be nested.
- SET TRANSACTION configures transaction:

```
SET TRANSACTION
[READ ONLY | READ WRITE] |
[ISOLATION LEVEL READ UNCOMMITTED |
READ COMMITTED|REPEATABLE READ |SERIALIZABLE ]
```


IMMEDIATE AND DEFERRED INTEGRITY CONSTRAINTS

- Do not always want constraints to be checked immediately, but instead at transaction commit.
- Constraint may be defined as INITIALLY IMMEDIATE or INITIALLY DEFERRED, indicating mode the constraint assumes at start of each transaction.
- In former case, also possible to specify whether mode can be changed subsequently using qualifier [NOT] DEFERRABLE.
- Default mode is INITIALLY IMMEDIATE.

IMMEDIATE AND DEFERRED INTEGRITY CONSTRAINTS

- SET CONSTRAINTS statement used to set mode for specified constraints for current transaction:

```
SET CONSTRAINTS
{ALL | constraintName [, . . . ]}
     {DEFERRED ¦ IMMEDIATE}
```


ACCESS CONTROL - AUTHORIZATION IDENTIFIERS AND OWNERSHIP

- Authorization identifier is normal SQL identifier used to establish identity of a user. Usually has an associated password.
- Used to determine which objects user may reference and what operations may be performed on those objects.
- Each object created in SQL has an owner, as defined in AUTHORIZATION clause of schema to which object belongs.
- Owner is only person who may know about it.

PRIVILEGES

- Actions user permitted to carry out on given base table or view:

SELECT	Retrieve data from a table.
INSERT	Insert new rows into a table.
UPDATE	Modify rows of data in a table.
DELETE	Delete rows of data from a table.
REFERENCES	Reference columns of named table in integrity constraints.
USAGE	Use domains, collations, character sets, and translations.

PRIVILEGES

- Can restrict INSERT/UPDATE/REFERENCES to named columns.
- Owner of table must grant other users the necessary privileges using GRANT statement.
- To create view, user must have SELECT privilege on all tables that make up view and REFERENCES privilege on the named columns.

GRANT

```
GRANT     {PrivilegeList | ALL PRIVILEGES}
ON    ObjectName
TO    {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

- PrivilegeList consists of one or more of above privileges separated by commas.
- ALL PRIVILEGES grants all privileges to a user.

GRANT

- PUBLIC allows access to be granted to all present and future authorized users.
- ObjectName can be a base table, view, domain, character set, collation or translation.
- WITH GRANT OPTION allows privileges to be passed on.

EXAMPLE 7 & 8 - GRANT

- Give Manager full privileges to Staff table.

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

- Give users Personnel and Director SELECT and UPDATE on column salary of Staff.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```


EXAMPLE 9 - GRANT SPECIFIC PRIVILEGES TO PUBLIC

- Give all users SELECT on Branch table.

```
GRANT SELECT
ON Branch
TO PUBLIC;
```


REVOKE

- REVOKE takes away privileges granted with GRANT.

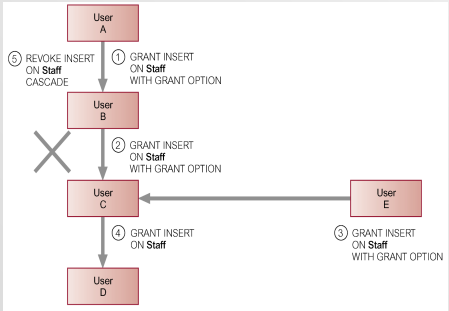
```
REVOKE [GRANT OPTION FOR]
      {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
        [RESTRICT | CASCADE]
```

- ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges.

REVOKE

- GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.
- REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.
- Privileges granted to this user by other users are not affected.

REVOKE



EXAMPLE 10 & 11 – REVOKE SPECIFIC PRIVILEGES

- Revoke privilege SELECT on Branch table from all users.

```
REVOKE SELECT
ON Branch
FROM PUBLIC;
```

- Revoke all privileges given to Director on Staff table.

```
REVOKE ALL PRIVILEGES
ON Staff
FROM Director;
```


CSE 204 - INTRO TO DATABASE SYSTEMS DATABASE DEVELOPMENT

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- Main components of an information system.
- Main stages of database system development lifecycle.
- Main phases of database design: conceptual, logical, and physical design.
- Benefits of CASE tools.
- How to evaluate and select a DBMS.
- Distinction between data administration and database administration.
- Purpose and tasks associated with data administration and database administration.

SOFTWARE DEPRESSION

- Last few decades have seen proliferation of software applications, many requiring constant maintenance involving:
  - correcting faults,
  - implementing new user requirements,
  - modifying software to run on new or upgraded platforms.
- Effort spent on maintenance began to absorb resources at an alarming rate.
- As a result, many major software projects were
  - late,
  - over budget,
  - unreliable,
  - difficult to maintain,
  - performed poorly.
- In late 1960s, led to 'software crisis', now refer to as the 'software depression'.

SOFTWARE DEPRESSION

- Major reasons for failure of software projects includes:
  - lack of a complete requirements specification;
  - lack of appropriate development methodology;
  - poor decomposition of design into manageable components.
- Structured approach to development was proposed called Information Systems Lifecycle (ISLC).

INFORMATION SYSTEM

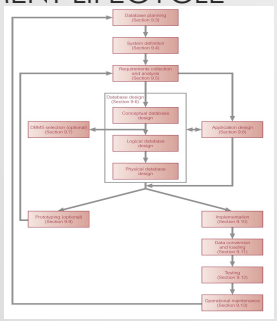
Resources that enable collection, management, control, and dissemination of information throughout an organization.

- Database is fundamental component of IS, and its development/usage should be viewed from perspective of the wider requirements of the organization.

DATABASE SYSTEM DEVELOPMENT LIFECYCLE

- Database planning
- System definition
- Requirements collection and analysis
- Database design
- DBMS selection (optional)
- Application design
- Prototyping (optional)
- Implementation
- Data conversion and loading
- Testing
- Operational maintenance

STAGES OF THE DATABASE SYSTEM DEVELOPMENT LIFECYCLE



DATABASE PLANNING

- Management activities that allow stages of database system development lifecycle to be realized as efficiently and effectively as possible.
- Must be integrated with overall IS strategy of the organization.

DATABASE PLANNING – MISSION STATEMENT

- *Mission statement* for the database project defines major aims of database application.
- Those driving database project normally define the mission statement.
- Mission statement helps clarify purpose of the database project and provides clearer path towards the efficient and effective creation of required database system.

DATABASE PLANNING – MISSION OBJECTIVES

- Once mission statement is defined, *mission objectives* are defined.
- Each objective should identify a particular task that the database must support.
- May be accompanied by some additional information that specifies the work to be done, the resources with which to do it, and the money to pay for it all.

DATABASE PLANNING

- Database planning should also include development of standards that govern:
  - how data will be collected,
  - how the format should be specified,
  - what necessary documentation will be needed,
  - how design and implementation should proceed.

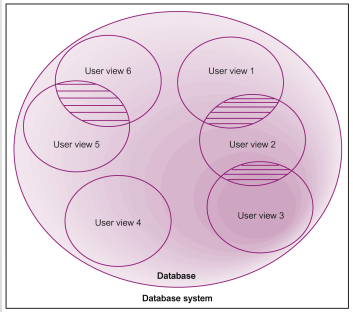
SYSTEM DEFINITION

- Describes scope and boundaries of database system and the major user views.
- User view defines what is required of a database system from perspective of:
  - a particular job role (such as Manager or Supervisor) or
  - enterprise application area (such as marketing, personnel, or stock control).

SYSTEM DEFINITION

- Database application may have one or more user views.
- Identifying user views helps ensure that no major users of the database are forgotten when developing requirements for new system.
- User views also help in development of complex database system allowing requirements to be broken down into manageable pieces.

REPRESENTATION OF A DATABASE SYSTEM WITH MULTIPLE USER VIEWS



REQUIREMENTS COLLECTION AND ANALYSIS

- Process of collecting and analyzing information about the part of organization to be supported by the database system, and using this information to identify users' requirements of new system.
- Information is gathered for each major user view including:
  - a description of data used or generated;
  - details of how data is to be used/generated;
  - any additional requirements for new database system.
- Information is analyzed to identify requirements to be included in new database system. Described in the requirements specification.

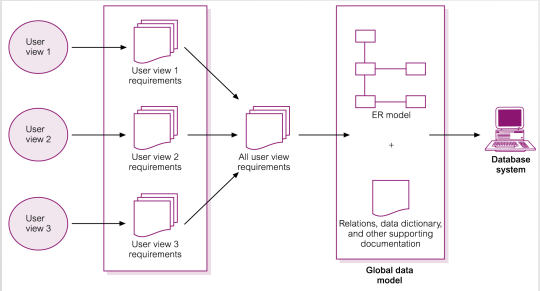
REQUIREMENTS COLLECTION AND ANALYSIS

- Another important activity is deciding how to manage the requirements for a database system with multiple user views.
- Three main approaches:
  - centralized approach;
  - view integration approach;
  - combination of both approaches.

REQUIREMENTS COLLECTION AND ANALYSIS

- Centralized approach
  - Requirements for each user view are merged into a single set of requirements.
  - A data model is created representing all user views during the database design stage.

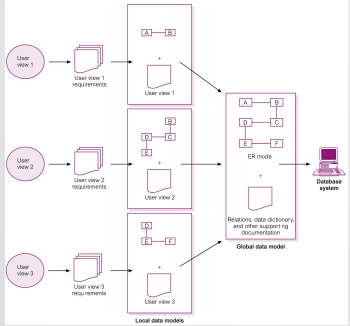
CENTRALIZED APPROACH TO MANAGING MULTIPLE USER VIEWS



REQUIREMENTS COLLECTION AND ANALYSIS

- View integration approach
  - Requirements for each user view remain as separate lists.
  - Data models representing each user view are created and then merged later during the database design stage.
- Data model representing single user view (or a subset of all user views) is called a local data model.
- Each model includes diagrams and documentation describing requirements for one or more but not all user views of database.
- Local data models are then merged at a later stage during database design to produce a global data model, which represents all user views for the database.

VIEW INTEGRATION APPROACH TO MANAGING MULTIPLE USER VIEWS



DATABASE DESIGN

- Process of creating a design for a database that will support the enterprise's mission statement and mission objectives for the required database system.
- Main approaches include:
  - Top-down
  - Bottom-up
  - Inside-out
  - Mixed

DATABASE DESIGN

- Main purposes of data modeling include:
  - to assist in understanding the meaning (semantics) of the data;
  - to facilitate communication about the information requirements.
- Building data model requires answering questions about entities, relationships, and attributes.
- A data model ensures we understand:
  - each user's perspective of the data;
  - nature of the data itself, independent of its physical representations;
  - use of data across user views.

CRITERIA TO PRODUCE AN OPTIMAL DATA MODEL

Structural validity	Consistency with the way the enterprise defines and organizes information.
Simplicity	Ease of understanding by IS professionals and non-technical users.
Expressibility	Ability to distinguish between different data, relationships between data, and constraints.
Nonredundancy	Exclusion of extraneous information; in particular, the representation of any one piece of information exactly once.
Shareability	Not specific to any particular application or technology and thereby usable by many.
Extensibility	Ability to evolve to support new requirements with minimal effect on existing users.
Integrity	Consistency with the way the enterprise uses and manages information.
Diagrammatic representation	Ability to represent a model using an easily understood diagrammatic notation.

DATABASE DESIGN

- Three phases of database design:
  - Conceptual database design
  - Logical database design
  - Physical database design.

CONCEPTUAL DATABASE DESIGN

- Process of constructing a model of the data used in an enterprise, independent of all physical considerations.
- Data model is built using the information in users' requirements specification.
- Conceptual data model is source of information for logical design phase.

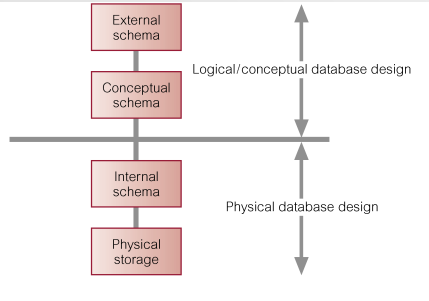
LOGICAL DATABASE DESIGN

- Process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.
- Conceptual data model is refined and mapped on to a logical data model.

PHYSICAL DATABASE DESIGN

- Process of producing a description of the database implementation on secondary storage.
- Describes base relations, file organizations, and indexes used to achieve efficient access to data. Also describes any associated integrity constraints and security measures.
- Tailored to a specific DBMS system.

THREE-LEVEL ANSI-SPARC ARCHITECTURE AND PHASES OF DATABASE DESIGN



DBMS SELECTION

- Selection of an appropriate DBMS to support the database system.
- Undertaken at any time prior to logical design provided sufficient information is available regarding system requirements.
- Main steps to selecting a DBMS:
  - define Terms of Reference of study;
  - shortlist two or three products;
  - evaluate products;
  - recommend selection and produce report.

DBMS EVALUATION FEATURES

DATA DEFINITION	PHYSICAL DEFINITION
Primary key enforcement	File structures available
Foreign key specification	File structure maintenance
Data types available	Ease of reorganization
Data type extensibility	Indexing
Domain specification	Variable length fields/records
Ease of restructuring	Data compression
Integrity controls	Encryption routines
View mechanism	Memory requirements
Data dictionary	Storage requirements
Data independence	
Underlying data model	
Schema evolution	
ACCESSIBILITY	**TRANSACTION HANDLING**
Query language: SQL2/SQL:2011/ODMG compliant	Backup and recovery routines Checkpointing facility
Interfacing to 3GLs	Logging facility
Multi-user	Granularity of concurrency
Security	Deadlock resolution strategy
• Access controls	Advanced transaction models
• Authorization mechanism	Parallel query processing

DBMS EVALUATION FEATURES

UTILITIES	DEVELOPMENT
Performance measuring	4GL/5GL tools
Tuning	CASE tools
Load/unload facilities	Windows capabilities
User usage monitoring	Stored procedures, triggers, and rules
Database administration support	Web development tools
OTHER FEATURES	
Upgradability	Interoperability with other DBMSs and other systems
Vendor stability	Web integration
User base	Replication utilities
Training and user support	Distributed capabilities
Documentation	Portability
Operating system required	Hardware required
Cost	Network support
Online help	Object-oriented capabilities
Standards used	Architecture (2- or 3-tier client/server)
Version management	Performance
Extensible query optimization	Transaction throughput
Scalability	Maximum number of concurrent users
Support for reporting and analytical tools	XML and Web services support

EXAMPLE - EVALUATION OF DBMS PRODUCT

DBMS: Sample product
Vendor: Sample vendor

Physical Definition Group

Features	Comments	Rating	Weighting	Score
File structures available	Choice of 4	8	0.15	1.2
File structure maintenance	NOT self-regulating	6	0.2	1.2
Ease of reorganization		4	0.25	1.0
Indexing		6	0.15	0.9
Variable length fields/records		6	0.15	0.9
Data compression	Specify with file structure	7	0.05	0.35
Encryption routines	Choice of 2	4	0.05	0.2
Memory requirements		0	0.00	0
Storage requirements		0	0.00	0
Totals		41	1.0	**5.75**
Physical definition group		5.75	0.25	**1.44**

APPLICATION DESIGN

- Design of user interface and application programs that use and process the database.
- Database design and application design are parallel activities.
- Includes two important activities:
  - transaction design;
  - user interface design.

APPLICATION DESIGN - TRANSACTIONS

- An action, or series of actions, carried out by a single user or application program, which accesses or changes content of the database.
- Should define and document the high-level characteristics of the transactions required.

APPLICATION DESIGN - TRANSACTIONS

- Important characteristics of transactions:
  - data to be used by the transaction;
  - functional characteristics of the transaction;
  - output of the transaction;
  - importance to the users;
  - expected rate of usage.
- Three main types of transactions: retrieval, update, and mixed.

PROTOTYPING

- Building working model of a database system.
- Purpose
  - to identify features of a system that work well, or are inadequate;
  - to suggest improvements or even new features;
  - to clarify the users' requirements;
  - to evaluate feasibility of a particular system design.

IMPLEMENTATION

- Physical realization of the database and application designs.
  - Use DDL to create database schemas and empty database files.
  - Use DDL to create any specified user views.
  - Use 3GL or 4GL to create the application programs. This will include the database transactions implemented using the DML, possibly embedded in a host programming language.

DATA CONVERSION AND LOADING

- Transferring any existing data into new database and converting any existing applications to run on new database.
- Only required when new database system is replacing an old system.
  - DBMS normally has utility that loads existing files into new database.
- May be possible to convert and use application programs from old system for use by new system.

TESTING

- Process of running the database system with intent of finding errors.
- Use carefully planned test strategies and realistic data.
- Testing cannot show absence of faults; it can show only that software faults are present.
- Demonstrates that database and application programs appear to be working according to requirements.

TESTING

- Should also test usability of system.
- Evaluation conducted against a usability specification.
- Examples of criteria include:
  - Learnability;
  - Performance;
  - Robustness;
  - Recoverability;
  - Adaptability.

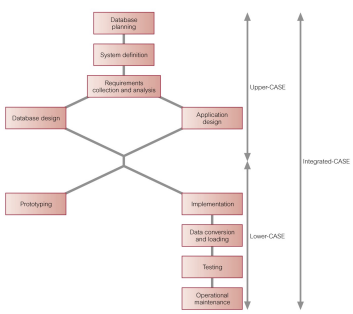
OPERATIONAL MAINTENANCE

- Process of monitoring and maintaining database system following installation.
- Monitoring performance of system.
- if performance falls, may require tuning or reorganization of the database.
- Maintaining and upgrading database application (when required).
- Incorporating new requirements into database application.

CASE TOOLS

- Support provided by CASE tools include:
  - data dictionary to store information about database system's data;
  - design tools to support data analysis;
  - tools to permit development of corporate data model, and conceptual and logical data models;
  - tools to enable prototyping of applications.
- Provide following benefits:
  - Standards;
  - Integration;
  - Support for standard methods;
  - Consistency;
  - Automation .

CASE TOOLS AND DATABASE SYSTEM DEVELOPMENT LIFECYCLE



DATA ADMINISTRATION AND DATABASE ADMINISTRATION

- The Data Administrator (DA) and Database Administrator (DBA) are responsible for managing and controlling the corporate data and corporate database, respectively.
- DA is more concerned with early stages of database system development lifecycle and DBA is more concerned with later stages.

DATA ADMINISTRATION

- Management of data resource including:
  - database planning,
  - development and maintenance of standards, policies and procedures, and conceptual and logical database design.

DATABASE ADMINISTRATION

- Management of physical realization of a database system including:
  - physical database design and implementation,
  - setting security and integrity controls,
  - monitoring system performance, and reorganizing the database.

CSE 204 - INTRO TO DATABASE SYSTEMS DATABASE ANALYSIS

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

QUOTE OF THE DAY

- The analysis of concepts is for the understanding nothing more than what the magnifying glass is for sight.
  - Moses Mendelssohn

OUTLINE

- When fact-finding techniques are used in the database application lifecycle.
- The types of facts collected in each stage of the database application lifecycle.
- The types of documentation produced in each stage of the database application lifecycle.
- The most commonly used fact-finding techniques.
- How to use each fact-finding technique and the advantages and disadvantages of each.
- About a property rental company called DreamHome.
- How to apply fact-finding techniques to the early stages of the database application lifecycle.

FACT-FINDING TECHNIQUES

- It is critical to capture the necessary facts to build the required database application.
- These facts are captured using fact-finding techniques.
- The formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences.

WHEN ARE FACT-FINDING TECHNIQUES USED?

- Fact-finding used throughout the database application lifecycle. Crucial to the early stages including database planning, system definition, and requirements collection and analysis stages.
- Enables developer to learn about the terminology, problems, opportunities, constraints, requirements, and priorities of the organization and the users of the system.

Stage of database system development lifecycle	Examples of data captured	Examples of documentation produced
Database planning	Aims and objectives of database project	Mission statement and objectives of database system
System definition	Description of major user views (includes job roles or business application areas)	Definition of scope and boundary of database application; definition of user views to be supported
Requirements collection and analysis	Requirements for user views; systems specifications, including performance and security requirements	Users' and system requirements specifications
Database design	Users' responses to checking the logical database design; functionality provided by target DBMS	Conceptual/logical database design (includes ER model(s), data dictionary, and relational schema); physical database design
Application design	Users' responses to checking interface design	Application design (includes description of programs and user interface)
DBMS selection	Functionality provided by target DBMS	DBMS evaluation and recommendations
Prototyping	Users' responses to prototype	Modified users' requirements and systems specifications
Implementation	Functionality provided by target DBMS	
Data conversion and loading	Format of current data; data import capabilities of target DBMS	
Testing	Test results	Testing strategies used; analysis of test results
Operational maintenance	Performance testing results; new or changing user and system requirements	User manual; analysis of performance results; modified users' requirements and systems specifications

EXAMPLES OF DATA CAPTURED AND DOCUMENTATION PRODUCED DURING THE DATABASE APPLICATION LIFECYCLE

FACT-FINDING TECHNIQUES

- A database developer normally uses several fact-finding techniques during a single database project including:
  - examining documentation
  - interviewing
  - observing the organization in operation
  - research
  - questionnaires

EXAMINING DOCUMENTATION

- Can be useful
  - to gain some insight as to how the need for a database arose.
  - to identify the part of the organization associated with the problem.
  - To understand the current system.

EXAMPLES OF TYPES OF DOCUMENTATION THAT SHOULD BE EXAMINED

Purpose of documentation	Examples of useful sources
Describes problem and need for database	Internal memos, e-mails, and minutes of meetings Employee/customer complaints, and documents that describe the problem Performance reviews/reports
Describes the part of the enterprise affected by problem	Organizational chart, mission statement, and strategic plan of the enterprise Objectives for the part of the enterprise being studied Task/job descriptions Samples of completed manual forms and reports Samples of completed computerized forms and reports
Describes current system	Various types of flowcharts and diagrams Data dictionary Database system design Program documentation User/training manuals

INTERVIEWING

- Most commonly used, and normally most useful, fact-finding technique. Enables collection of information from individuals face-to-face.
- Objectives include finding out facts, verifying facts, clarifying facts, generating enthusiasm, getting the end-user involved, identifying requirements, and gathering ideas and opinions.

ADVANTAGES AND DISADVANTAGES OF INTERVIEWING

Advantages	Disadvantages
Allows interviewee to respond freely and openly to questions	Very time-consuming and costly, and therefore may be impractical
Allows interviewee to feel part of project	Success is dependent on communication skills of interviewer
Allows interviewer to follow up on interesting comments made by interviewee	Success can be dependent on willingness of interviewees to participate in interviews
Allows interviewer to adapt or re-word questions during interview	
Allows interviewer to observe interviewee's body language	

INTERVIEWING

- There are two types of interviews unstructured and structured.
- Open-ended questions allow the interviewee to respond in any way that seems appropriate.
- Closed-ended questions restrict answers to either specific choices or short, direct responses.

OBSERVING THE ORGANIZATION IN OPERATION

- An effective technique for understanding a system.
- Possible to either participate in, or watch, a person perform activities to learn about the system.
- Useful when validity of data collected is in question or when the complexity of certain aspects of the system prevents a clear explanation by the end-users.

ADVANTAGES AND DISADVANTAGES OF USING OBSERVATION

Advantages	Disadvantages
Allows the validity of facts and data to be checked	People may knowingly or unknowingly perform differently when being observed
Observer can see exactly what is being done	May miss observing tasks involving different levels of difficulty or volume normally experienced during that time period
Observer can also obtain data describing the physical environment of the task	Some tasks may not always be performed in the manner in which they are observed
Relatively inexpensive	May be impractical
Observer can do work measurements	

RESEARCH

- Useful to research the application and problem.
- Use computer trade journals, reference books, and the Internet (including user groups and bulletin boards).
- Provide information on how others have solved similar problems, plus whether or not software packages exist to solve or even partially solve the problem.

ADVANTAGES AND DISADVANTAGES OF USING RESEARCH

Advantages	Disadvantages
Can save time if solution already exists	Requires access to appropriate sources of information
Researcher can see how others have solved similar problems or met similar requirements	May ultimately not help in solving problem because problem is not documented elsewhere
Keeps researcher up to date with current developments	

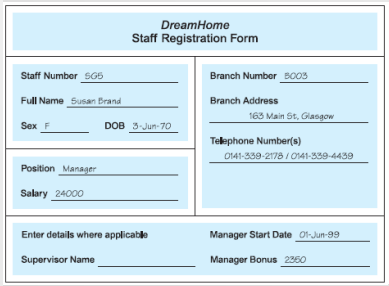
QUESTIONNAIRES

- Conduct surveys through questionnaires, which are special-purpose documents that allow facts to be gathered from a large number of people while maintaining some control over their responses.
- There are two types of questions, namely free-format and fixed-format.

ADVANTAGES AND DISADVANTAGES OF USING QUESTIONNAIRES

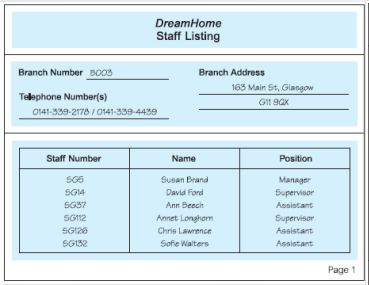
Advantages	Disadvantages
People can complete and return questionnaires at their convenience	Number of respondents can be low, possibly only 5% to 10%
Relatively inexpensive way to gather data from a large number of people	Questionnaires may be returned incomplete
People more likely to provide the real facts as responses can be kept confidential	May not provide an opportunity to adapt or re-word questions that have been misinterpreted
Responses can be tabulated and analyzed quickly	Cannot observe and analyze the respondent's body language

USING FACT-FINDING TECHNIQUES – A WORKED EXAMPLE



DreamHome
Staff Registration Form

Staff Number SG5
Full Name Susan Brand
Sex F DOB 3-Jun-70
Position Manager
Salary 24000
Branch Number B003
Branch Address
163 Main St, Glasgow
Telephone Number(s)
0141-339-2178 / 0141-339-4439
Enter details where applicable
Supervisor Name
Manager Start Date 01-Jun-99
Manager Bonus 2360



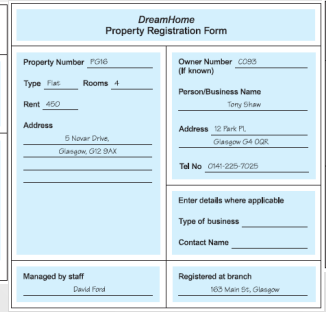
DreamHome
Staff Listing

Branch Number B003
Telephone Number(s)
0141-339-2178 / 0141-339-4439
Branch Address
163 Main St, Glasgow
G11 9QX

Staff Number	Name	Position
SG5	Susan Brand	Manager
SG14	David Ford	Supervisor
SG37	Ann Beech	Assistant
SG112	Annet Longhorn	Supervisor
SG128	Chris Lawrence	Assistant
SG132	Sofie Walters	Assistant

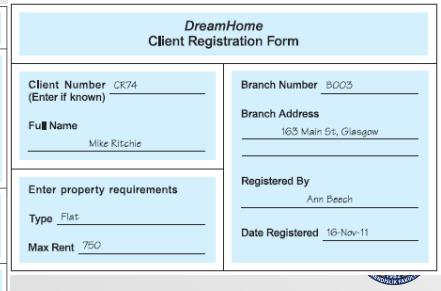
Page 1

USING FACT-FINDING TECHNIQUES – A WORKED EXAMPLE



DreamHome
Property Registration Form

Property Number PG16
Type Flat Rooms 4
Rent 450
Address
6 Novar Drive,
Glasgow, G12 9AX
Owner Number (If known) CO93
Person/Business Name
Tony Shaw
Address 12 Park Pl,
Glasgow G4 0QR
Tel No 0141-225-7025
Enter details where applicable
Type of business
Contact Name
Managed by staff
David Ford
Registered at branch
163 Main St, Glasgow



DreamHome
Client Registration Form

Client Number (Enter if known) CR74
Full Name
Mike Ritchie
Enter property requirements
Type Flat
Max Rent 750
Branch Number B003
Branch Address
163 Main St, Glasgow
Registered By
Ann Beech
Date Registered 16-Nov-11

USING FACT-FINDING TECHNIQUES – A WORKED EXAMPLE

DreamHome
Property Listing for Week beginning 01/06/13

If you are interested in viewing or renting any of the properties in this list, please contact our branch office as soon as possible.

Branch Address
163 Main St, Glasgow
G11 9QX

Telephone Number(s)
0141-339-2178 / 0141-339-4439

Property No	Address	Type	Rooms	Rent
PG4	6 Lawrence St, Glasgow	Flat	3	350
PG36	2 Manor Rd, Glasgow	Flat	3	375
PG21	18 Dale Road, Glasgow	House	5	600
PG16	5 Novar Drive, Glasgow	Flat	4	450
PG77	100A Apple Lane, Glasgow	House	6	560
PG81	781 Greentree Dr, Glasgow	Flat	4	440

Page 1

DreamHome
Property Viewing Report

Property Numner PG4
Property Address 6 Lawrence St, Glasgow
Type Flat
Rent 350

Client No	Name	Date	Comments
CR76	John Kay	20/04/13	Too remote.
CR56	Aline Stewart	26/05/13	
CR74	Mike Ritchie	11/11/13	
CR62	Mary Tregear	11/11/13	OK, but needs redecoration throughout.

Page 1

21

USING FACT-FINDING TECHNIQUES – A WORKED EXAMPLE

DreamHome Lease
Number 00345810

Client Number CR74 (Enter if known)
Full Name Mike Ritchie (Please print)
Client Signature

Property Number PG16
Property Address 5 Novar Dr, Glasgow

Enter payment details
Monthly Rent 450
Payment Method Cheque
Deposit Paid (Y or N) Yes

Rent Start 01/06/12
Rent Finish 31/05/13
Duration 1 year

22

MISSION STATEMENT FOR *DREAMHOME* DATABASE SYSTEM

"The purpose of the *DreamHome* database system is to maintain the data that is used and generated to support the property rentals business for our clients and property owners and to facilitate the cooperation and sharing of information between branches."

23

MISSION OBJECTIVES FOR *DREAMHOME* DATABASE SYSTEM

To maintain (enter, update, and delete) data on branches.
To maintain (enter, update, and delete) data on staff.
To maintain (enter, update, and delete) data on properties for rent.
To maintain (enter, update, and delete) data on property owners.
To maintain (enter, update, and delete) data on clients.
To maintain (enter, update, and delete) data on property viewings.
To maintain (enter, update, and delete) data on leases.
To maintain (enter, update, and delete) data on newspaper adverts.

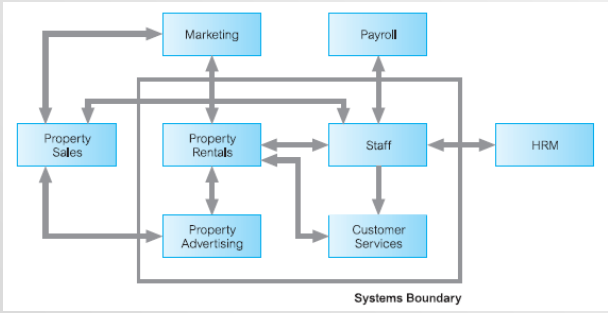
To perform searches on branches.
To perform searches on staff.
To perform searches on properties for rent.
To perform searches on property owners.
To perform searches on clients.
To perform searches on property viewings.
To perform searches on leases.
To perform searches on newspaper adverts.

To track the status of property for rent.
To track the status of clients wishing to rent.
To track the status of leases.

To report on branches.
To report on staff.
To report on properties for rent.
To report on property owners.
To report on clients.
To report on property viewings.
To report on leases.
To report on newspaper adverts.

24

SYSTEM BOUNDARY FOR *DREAMHOME* DATABASE SYSTEM



MAJOR USER VIEWS FOR *DREAMHOME* DATABASE SYSTEM

Data	Access Type	Director	Manager	Supervisor	Assistant
All Branches	Maintain				
	Query	X	X		
	Report	X	X		
Single Branch	Maintain		X		
	Query		X		
	Report		X		
All Staff	Maintain				
	Query	X	X		
	Report	X	X		
Branch Staff	Maintain		X		
	Query		X	X	
	Report		X	X	
All Property	Maintain				
	Query	X			
	Report	X	X		
Branch Property	Maintain		X	X	
	Query		X	X	X
	Report		X	X	X
All Owners	Maintain				
	Query	X			
	Report	X	X		
Branch Owners	Maintain		X	X	
	Query		X	X	X
	Report		X		
All Clients	Maintain				
	Query	X			
	Report	X	X		
Branch Clients	Maintain		X	X	
	Query		X	X	X
	Report		X		
All Viewings	Maintain				
	Query				
	Report				
Branch Viewings	Maintain			X	X
	Query			X	X
	Report			X	X
All Leases	Maintain				
	Query	X			
	Report	X	X		
Branch Leases	Maintain		X	X	
	Query		X	X	X
	Report		X	X	
All Newspapers	Maintain				
	Query	X			
	Report	X	X		
Branch Newspapers	Maintain		X		
	Query		X		
	Report		X		

CROSS-REFERENCE OF USER VIEWS WITH MAIN TYPES OF DATA USED BY EACH

	Director	Manager	Supervisor	Assistant
branch	X	X		
staff	X	X	X	
property for rent	X	X	X	X
owner	X	X	X	X
client	X	X	X	X
property viewing			X	X
lease	X	X	X	X
newspaper	X	X		

27

CSE 204 - INTRO TO DATABASE SYSTEMS E-R MODELING

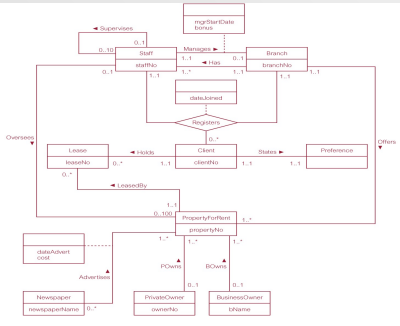
Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- How to use Entity–Relationship (ER) modeling in database design.
- Basic concepts associated with ER model.
- Diagrammatic technique for displaying ER model using Unified Modeling Language (UML).
- How to identify and resolve problems with ER models called connection traps.
- How to build an ER model from a requirements specification.

2

ER DIAGRAM OF BRANCH USER VIEWS OF *DREAMHOME*



CONCEPTS OF THE ER MODEL

- Entity types
- Relationship types
- Attributes

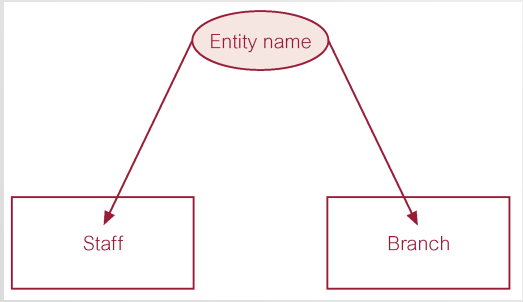
ENTITY TYPE

- Entity type
  - Group of objects with same properties, identified by enterprise as having an independent existence.
- Entity occurrence
  - Uniquely identifiable object of an entity type.

EXAMPLES OF ENTITY TYPES

Physical existence	
Staff	Part
Property	Supplier
Customer	Product
Conceptual existence	
Viewing	Sale
Inspection	Work experience

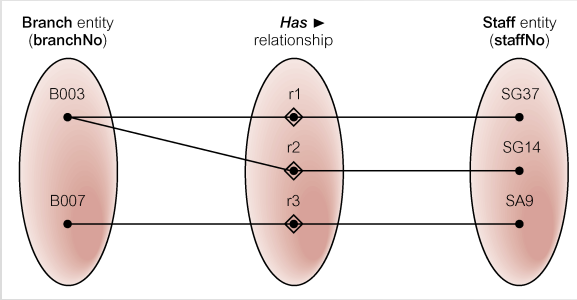
ER DIAGRAM OF STAFF AND BRANCH ENTITY TYPES



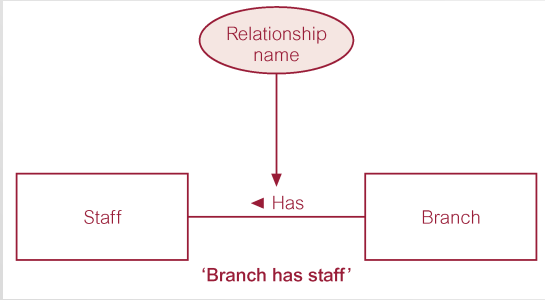
RELATIONSHIP TYPES

- Relationship type
  - Set of meaningful associations among entity types.
- Relationship occurrence
  - Uniquely identifiable association, which includes one occurrence from each participating entity type.

SEMANTIC NET OF *HAS* RELATIONSHIP TYPE



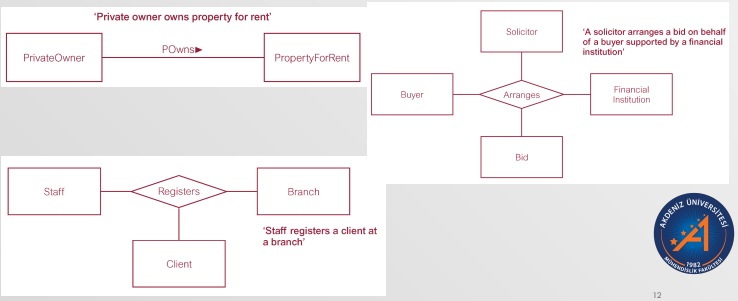
ER DIAGRAM OF BRANCH *HAS* STAFF RELATIONSHIP



RELATIONSHIP TYPES

- Degree of a Relationship
  - Number of participating entities in relationship.
- Relationship of degree :
  - two is binary
  - three is ternary
  - four is quaternary.

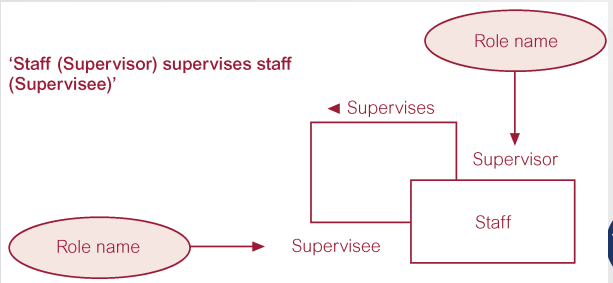
RELATIONSHIPS



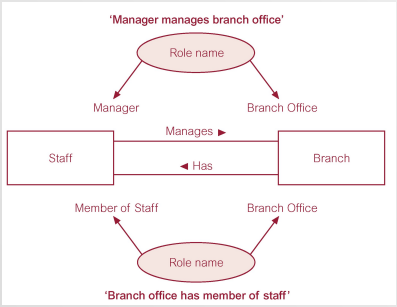
RELATIONSHIP TYPES

- Recursive Relationship
  - Relationship type where same entity type participates more than once in different roles.
- Relationships may be given role names to indicate purpose that each participating entity type plays in a relationship.

RECURSIVE RELATIONSHIP CALLED *SUPERVISES* WITH ROLE NAMES



ENTITIES ASSOCIATED THROUGH TWO DISTINCT RELATIONSHIPS WITH ROLE NAMES



ATTRIBUTES

- Attribute
  - Property of an entity or a relationship type.
- Attribute Domain
  - Set of allowable values for one or more attributes.
- Simple Attribute
  - Attribute composed of a single component with an independent existence.
- Composite Attribute
  - Attribute composed of multiple components, each with an independent existence.

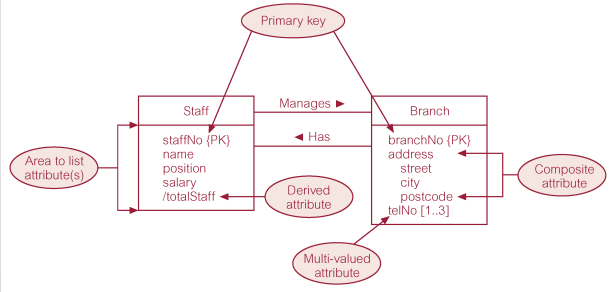
ATTRIBUTES

- Single-valued Attribute
  - Attribute that holds a single value for each occurrence of an entity type.
- Multi-valued Attribute
  - Attribute that holds multiple values for each occurrence of an entity type.
- Derived Attribute
  - Attribute that represents a value that is derivable from value of a related attribute, or set of attributes, not necessarily in the same entity type.

KEYS

- Candidate Key
  - Minimal set of attributes that uniquely identifies each occurrence of an entity type.
- Primary Key
  - Candidate key selected to uniquely identify each occurrence of an entity type.
- Composite Key
  - A candidate key that consists of two or more attributes.

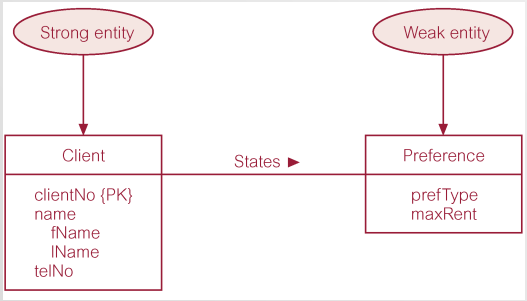
ER DIAGRAM OF STAFF AND BRANCH ENTITIES AND THEIR ATTRIBUTES



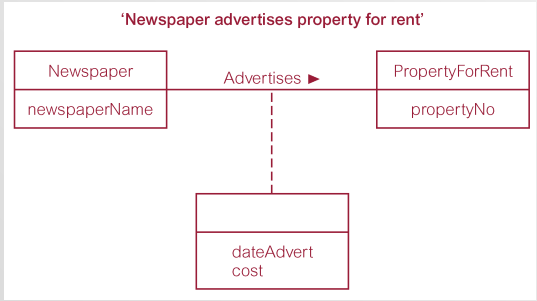
ENTITY TYPE

- Strong Entity Type
  - Entity type that is not existence-dependent on some other entity type.
- Weak Entity Type
  - Entity type that is existence-dependent on some other entity type.

STRONG ENTITY TYPE CALLED CLIENT AND WEAK ENTITY TYPE CALLED PREFERENCE



RELATIONSHIP CALLED *ADVERTISES* WITH ATTRIBUTES



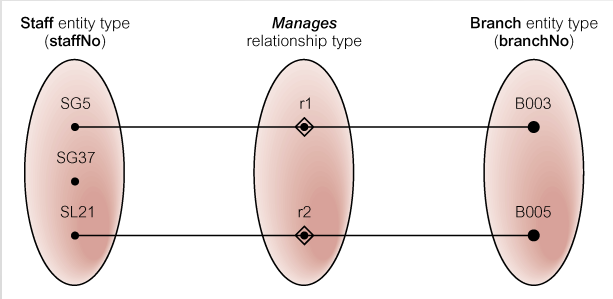
STRUCTURAL CONSTRAINTS

- Main type of constraint on relationships is called ***multiplicity***.
- Multiplicity - number (or range) of possible occurrences of an entity type that may relate to a single occurrence of an associated entity type through a particular relationship.
- Represents policies (called ***business rules***) established by user or company.

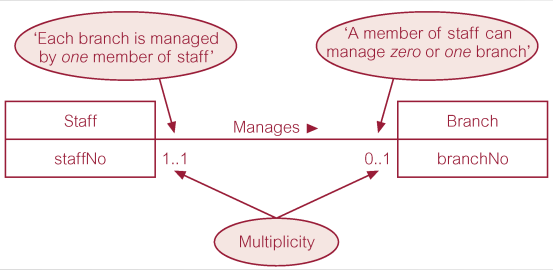
STRUCTURAL CONSTRAINTS

- The most common degree for relationships is binary.
- Binary relationships are generally referred to as being:
  - one-to-one (1:1)
  - one-to-many (1:*)
  - many-to-many (*:*)

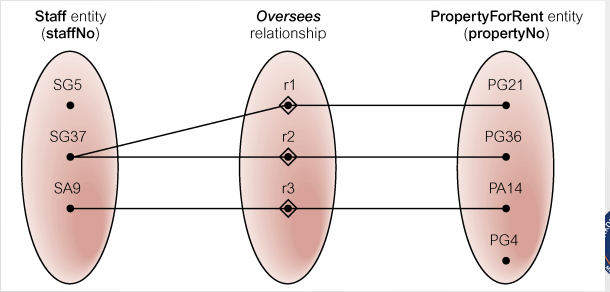
SEMANTIC NET OF STAFF *MANAGES* BRANCH RELATIONSHIP TYPE



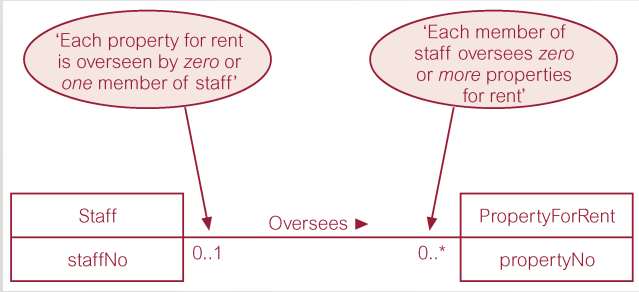
MULTIPLICITY OF STAFF MANAGES BRANCH (1:1) RELATIONSHIP



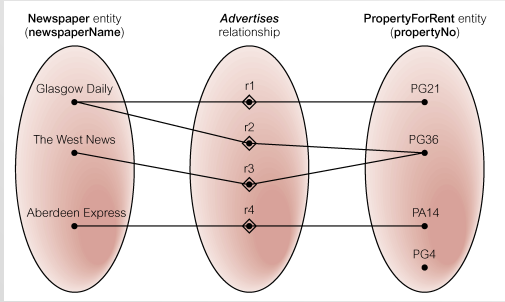
SEMANTIC NET OF STAFF *OVERSEES* PROPERTYFORRENT RELATIONSHIP TYPE



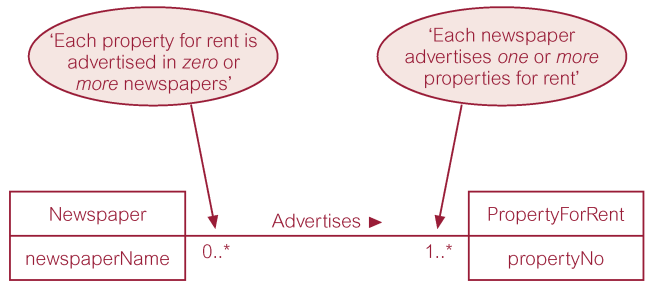
MULTIPLICITY OF STAFF *OVERSEES* PROPERTYFORRENT (1:*) RELATIONSHIP TYPE



SEMANTIC NET OF NEWSPAPER *ADVERTISES* PROPERTYFORRENT RELATIONSHIP TYPE



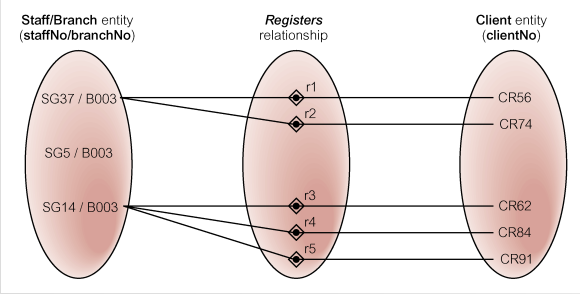
MULTIPLICITY OF NEWSPAPER *ADVERTISES* PROPERTYFORRENT (*:*) RELATIONSHIP



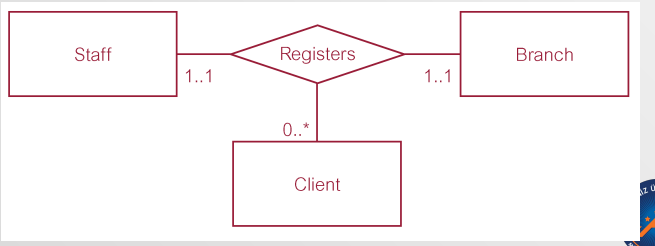
STRUCTURAL CONSTRAINTS

- Multiplicity for Complex Relationships
  - Number (or range) of possible occurrences of an entity type in an n-ary relationship when other (n-1) values are fixed.

SEMANTIC NET OF TERNARY *REGISTERS* RELATIONSHIP WITH VALUES FOR STAFF AND BRANCH ENTITIES FIXED



MULTIPLICITY OF TERNARY *REGISTERS* RELATIONSHIP



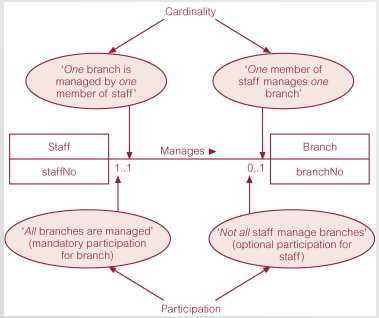
SUMMARY OF MULTIPLICITY CONSTRAINTS

Alternative ways to represent multiplicity constraints	Meaning
0..1	Zero or one entity occurrence
1..1 (or just 1)	Exactly one entity occurrence
0..* (or just *)	Zero or many entity occurrences
1..*	One or many entity occurrences
5..10	Minimum of 5 up to a maximum of 10 entity occurrences
0, 3, 6–8	Zero or three or six, seven, or eight entity occurrences

STRUCTURAL CONSTRAINTS

- Multiplicity is made up of two types of restrictions on relationships: cardinality and participation.
- Cardinality
  - Describes maximum number of possible relationship occurrences for an entity participating in a given relationship type.
- Participation
  - Determines whether all or only some entity occurrences participate in a relationship.

MULTIPLICITY AS CARDINALITY AND PARTICIPATION CONSTRAINTS

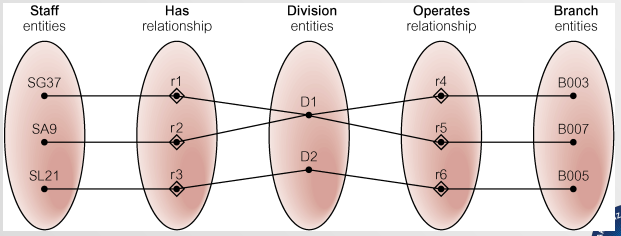


PROBLEMS WITH ER MODELS

- Problems may arise when designing a conceptual data model called *connection traps*.
- Often due to a misinterpretation of the meaning of certain relationships.
- Two main types of connection traps are called *fan traps* and *chasm traps*.
- Fan Trap
  - Where a model represents a relationship between entity types, but pathway between certain entity occurrences is ambiguous.
- Chasm Trap
  - Where a model suggests the existence of a relationship between entity types, but pathway does not exist between certain entity occurrences.

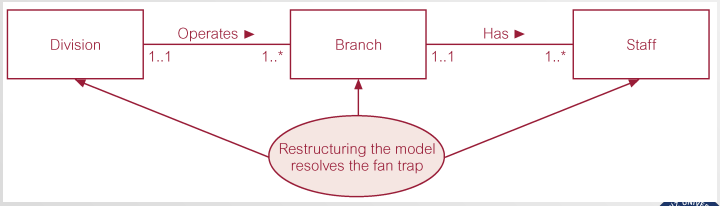
AN EXAMPLE OF A FAN TRAP

SEMANTIC NET OF ER MODEL WITH FAN TRAP

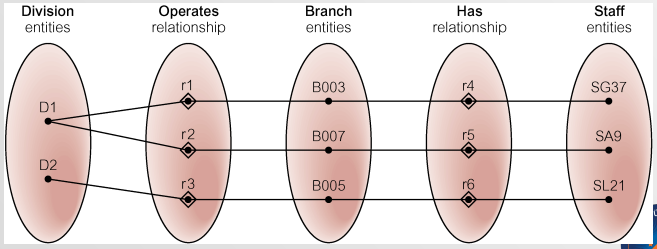


- At which branch office does staff number SG37 work?

RESTRUCTURING ER MODEL TO REMOVE FAN TRAP



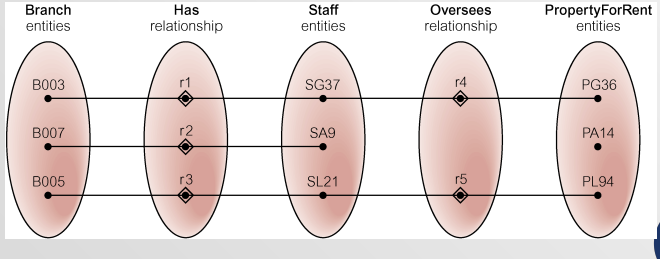
SEMANTIC NET OF RESTRUCTURED ER MODEL WITH FAN TRAP REMOVED



- SG37 works at branch B003.

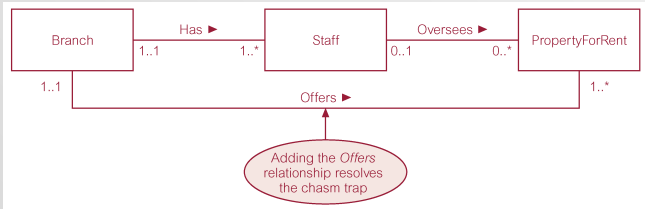
AN EXAMPLE OF A CHASM TRAP

SEMANTIC NET OF ER MODEL WITH CHASM TRAP

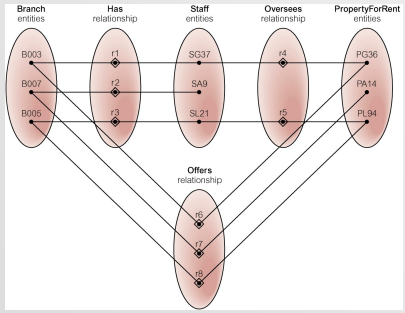


- At which branch office is property PA14 available?

ER MODEL RESTRUCTURED TO REMOVE CHASM TRAP



SEMANTIC NET OF RESTRUCTURED ER MODEL WITH CHASM TRAP REMOVED



CHAPTER 13 - OUTLINE

- Limitations of basic concepts of the ER model and requirements to represent more complex applications using additional data modeling concepts.
- Most useful additional data modeling concept of Enhanced ER (EER) model is called specialization/generalization.
- A diagrammatic technique for displaying specialization/generalization in an EER diagram using UML.

ENHANCED ENTITY-RELATIONSHIP MODEL

- Since 1980s there has been an increase in emergence of new database applications with more demanding requirements.
- Basic concepts of ER modeling are not sufficient to represent requirements of newer, more complex applications.
- Response is development of additional 'semantic' modeling concepts.
- Semantic concepts are incorporated into the original ER model and called the Enhanced Entity-Relationship (EER) model.
- Examples of additional concept of EER model is called specialization / generalization.

SPECIALIZATION / GENERALIZATION

- Superclass
  - An entity type that includes one or more distinct subgroupings of its occurrences.
- Subclass
  - A distinct subgrouping of occurrences of an entity type.
- Superclass/subclass relationship is one-to-one (1:1).
- Superclass may contain overlapping or distinct subclasses.
- Not all members of a superclass need be a member of a subclass.

SPECIALIZATION / GENERALIZATION

- Attribute Inheritance
  - An entity in a subclass represents same 'real world' object as in superclass, and may possess subclass-specific attributes, as well as those associated with the superclass.
- Specialization
  - Process of maximizing differences between members of an entity by identifying their distinguishing characteristics.
- Generalization
  - Process of minimizing differences between entities by identifying their common characteristics.

ALLSTAFF RELATION HOLDING DETAILS OF ALL STAFF

Attributes appropriate for all staff

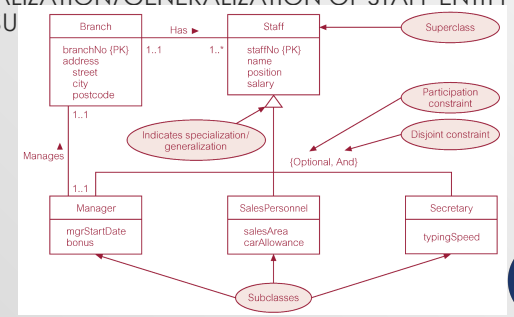
Attributes appropriate for branch Managers

Attributes appropriate for Sales Personnel

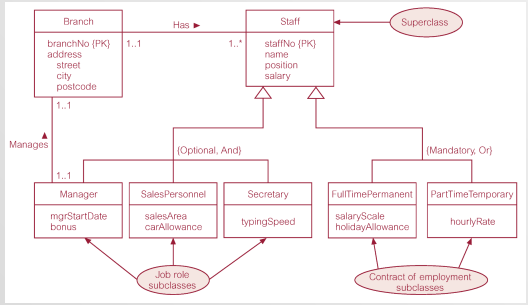
Attribute appropriate for Secretarial staff

staffNo	name	position	salary	mgrStartDate	bonus	sales Area	car Allowance	typing Speed
SL21	John White	Manager	30000	01/02/95	2000			
SG37	Ann Beech	Assistant	12000					
SG66	Mary Martinez	Sales Manager	27000			SA1A	5000	
SA9	Mary Howe	Assistant	9000					
SL89	Stuart Stern	Secretary	8500					100
SL31	Robert Chin	Snr Sales Asst	17000			SA2B	3700	
SG5	Susan Brand	Manager	24000	01/06/91	2350			

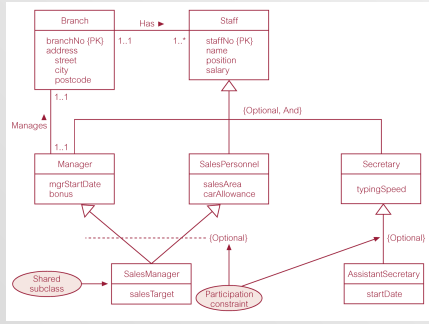
SPECIALIZATION/GENERALIZATION OF STAFF ENTITY INTO SU



SPECIALIZATION/GENERALIZATION OF STAFF ENTITY INTO JOB ROLES AND CONTRACTS OF EMPLOYMENT



EER DIAGRAM WITH SHARED SUBCLASS AND SUBCLASS WITH ITS OWN SUBCLASS



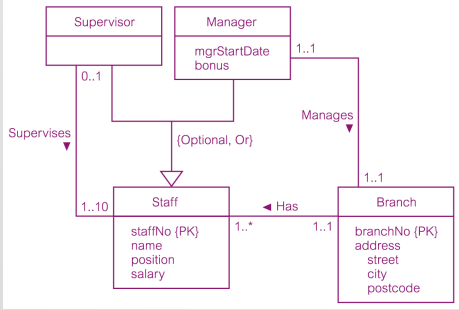
CONSTRAINTS ON SPECIALIZATION / GENERALIZATION

- Two constraints that may apply to a specialization/generalization:
  - participation constraints
  - disjoint constraints.
- Participation constraint
  - Determines whether every member in superclass must participate as a member of a subclass.
  - May be mandatory or optional.

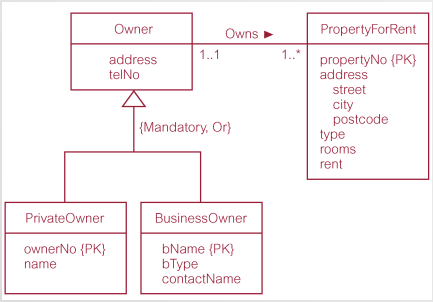
CONSTRAINTS ON SPECIALIZATION / GENERALIZATION

- Disjoint constraint
  - Describes relationship between members of the subclasses and indicates whether member of a superclass can be a member of one, or more than one, subclass.
  - May be disjoint or nondisjoint.
- There are four categories of constraints of specialization and generalization:
  - mandatory and disjoint
  - optional and disjoint
  - mandatory and nondisjoint
  - optional and nondisjoint.

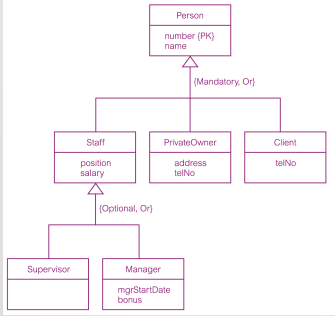
DREAMHOME WORKED EXAMPLE - STAFF SUPERCLASS WITH SUPERVISOR AND MANAGER SUBCLASSES



DREAMHOME WORKED EXAMPLE - OWNER SUPERCLASS WITH PRIVATEOWNER AND BUSINESSOWNER SUBCLASSES



DREAMHOME WORKED EXAMPLE - OWNER SUPERCLASS WITH PRIVATEOWNER AND BUSINESSOWNER SUBCLASSES



CSE 204 - INTRO TO DATABASE SYSTEMS NORMALIZATION

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- The purpose of normalization.
- How normalization can be used when designing a relational database.
- The potential problems associated with redundant data in base relations.
- The concept of functional dependency, which describes the relationship between attributes.
- The characteristics of functional dependencies used in normalization.
- How to identify functional dependencies for a given relation.

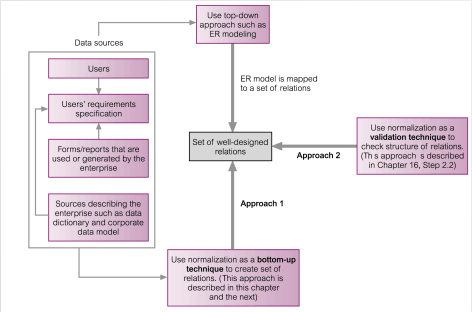
OUTLINE

- How functional dependencies identify the primary key for a relation.
- How to undertake the process of normalization.
- How normalization uses functional dependencies to group attributes into relations that are in a known normal form.
- How to identify the most commonly used normal forms, namely First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF).
- The problems associated with relations that break the rules of 1NF, 2NF, or 3NF.
- How to represent attributes shown on a form as 3NF relations using normalization.

PURPOSE OF NORMALIZATION

- Normalization is a technique for producing a set of suitable relations that support the data requirements of an enterprise.
- Characteristics of a suitable set of relations include:
  - the minimal number of attributes necessary to support the data requirements of the enterprise;
  - attributes with a close logical relationship are found in the same relation;
  - minimal redundancy with each attribute represented only once with the important exception of attributes that form all or part of foreign keys.
- The benefits of using a database that has a suitable set of relations is that the database will be:
  - easier for the user to access and maintain the data;
  - take up minimal storage space on the computer.

HOW NORMALIZATION SUPPORTS DATABASE DESIGN



DATA REDUNDANCY AND UPDATE ANOMALIES

- Major aim of relational database design is to group attributes into relations to minimize data redundancy.
- Potential benefits for implemented database include:
  - Updates to the data stored in the database are achieved with a minimal number of operations thus reducing the opportunities for data inconsistencies.
  - Reduction in the file storage space required by the base relations thus minimizing costs.
- Problems associated with data redundancy are illustrated by comparing the Staff and Branch relations with the StaffBranch relation.

DATA REDUNDANCY AND UPDATE ANOMALIES

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

DATA REDUNDANCY AND UPDATE ANOMALIES

- StaffBranch relation has redundant data; the details of a branch are repeated for every member of staff.
- In contrast, the branch information appears only once for each branch in the Branch relation and only the branch number (branchNo) is repeated in the Staff relation, to represent where each member of staff is located.
- Relations that contain redundant information may potentially suffer from update anomalies.
- Types of update anomalies include
  - Insertion
  - Deletion
  - Modification

LOSSLESS-JOIN AND DEPENDENCY PRESERVATION PROPERTIES

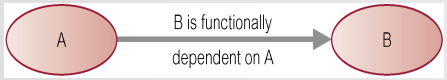
- Two important properties of decomposition.
  - Lossless-join property enables us to find any instance of the original relation from corresponding instances in the smaller relations.
  - Dependency preservation property enables us to enforce a constraint on the original relation by enforcing some constraint on each of the smaller relations.

FUNCTIONAL DEPENDENCIES

- Important concept associated with normalization.
- Functional dependency describes relationship between attributes.
- For example, if A and B are attributes of relation R, B is functionally dependent on A (denoted A -> B), if each value of A in R is associated with exactly one value of B in R.

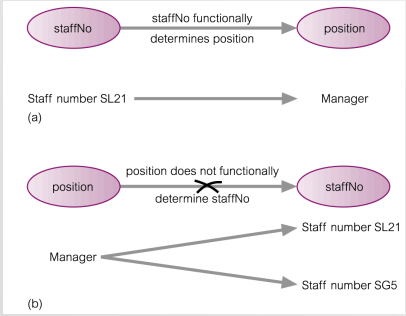
CHARACTERISTICS OF FUNCTIONAL DEPENDENCIES

- Property of the meaning or semantics of the attributes in a relation.
- Diagrammatic representation.



- The determinant of a functional dependency refers to the attribute or group of attributes on the left-hand side of the arrow.

AN EXAMPLE FUNCTIONAL DEPENDENCY



EXAMPLE FUNCTIONAL DEPENDENCY THAT HOLDS FOR ALL TIME

- Consider the values shown in staffNo and sName attributes of the Staff relation (slide 7).
- Based on sample data, the following functional dependencies appear to hold.
  - staffNo → sName
  - sName → staffNo
- However, the only functional dependency that remains true for all possible values for the staffNo and sName attributes of the Staff relation is:
  - staffNo → sName

CHARACTERISTICS OF FUNCTIONAL DEPENDENCIES

- Determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side.
- This requirement is called full functional dependency.
- Full functional dependency indicates that if A and B are attributes of a relation, B is fully functionally dependent on A, if B is functionally dependent on A, but not on any proper subset of A.

EXAMPLE FULL FUNCTIONAL DEPENDENCY

- Exists in the Staff relation (slide 7).
  - staffNo, sName → branchNo
- True - each value of (staffNo, sName) is associated with a single value of branchNo.
- However, branchNo is also functionally dependent on a subset of (staffNo, sName), namely staffNo. Example above is a partial dependency.

CHARACTERISTICS OF FUNCTIONAL DEPENDENCIES

- Main characteristics of functional dependencies used in normalization:
  - There is a one-to-one relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency.
  - Holds for all time.
  - The determinant has the minimal number of attributes necessary to maintain the dependency with the attribute(s) on the right hand-side.

TRANSITIVE DEPENDENCIES

- Important to recognize a transitive dependency because its existence in a relation can potentially cause update anomalies.
- Transitive dependency describes a condition where A, B, and C are attributes of a relation such that if A → B and B → C, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C).

EXAMPLE TRANSITIVE DEPENDENCY

- Consider functional dependencies in the StaffBranch relation (slide 7)
  - staffNo → sName, position, salary, branchNo, bAddress
  - branchNo → bAddress
- Transitive dependency, branchNo → bAddress exists on staffNo via branchNo.

THE PROCESS OF NORMALIZATION

- Formal technique for analyzing a relation based on its primary key and the functional dependencies between the attributes of that relation.
- Often executed as a series of steps. Each step corresponds to a specific normal form, which has known properties.

IDENTIFYING FUNCTIONAL DEPENDENCIES

- Identifying all functional dependencies between a set of attributes is relatively simple if the meaning of each attribute and the relationships between the attributes are well understood.
- This information should be provided by the enterprise in the form of discussions with users and/or documentation such as the users' requirements specification.
- However, if the users are unavailable for consultation and/or the documentation is incomplete then depending on the database application it may be necessary for the database designer to use their common sense and/or experience to provide the missing information.

EXAMPLE - IDENTIFYING A SET OF FUNCTIONAL DEPENDENCIES FOR THE STAFFBRANCH RELATION

- Examine semantics of attributes in StaffBranch relation (slide 7). Assume that position held and branch determine a member of staff's salary.
- With sufficient information available, identify the functional dependencies for the StaffBranch relation as:
  - staffNo → sName, position, salary, branchNo, bAddress
  - branchNo → bAddress
  - bAddress → branchNo
  - branchNo, position → salary
  - bAddress, position → salary

EXAMPLE - USING SAMPLE DATA TO IDENTIFY FUNCTIONAL DEPENDENCIES.

- Consider the data for attributes denoted A, B, C, D, and E in the Sample relation (slide 23).
- Important to establish that sample data values shown in relation are representative of all possible values that can be held by attributes A, B, C, D, and E. Assume true despite the relatively small amount of data shown in this relation.

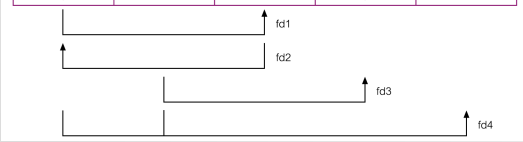
QUOTE OF THE DAY

- I never guess. It is a shocking habit destructive to the logical faculty.
  - Arthur Conan Doyle

EXAMPLE - USING SAMPLE DATA TO IDENTIFY FUNCTIONAL DEPENDENCIES.

Sample Relation

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



EXAMPLE - USING SAMPLE DATA TO IDENTIFY FUNCTIONAL DEPENDENCIES.

- Function dependencies between attributes A to E in the Sample relation.
  - A → C (fd1)
  - C → A (fd2)
  - B → D (fd3)
  - A, B → E (fd4)

IDENTIFYING THE PRIMARY KEY FOR A RELATION USING FUNCTIONAL DEPENDENCIES

- Main purpose of identifying a set of functional dependencies for a relation is to specify the set of integrity constraints that must hold on a relation.
- An important integrity constraint to consider first is the identification of candidate keys, one of which is selected to be the primary key for the relation.

EXAMPLE - IDENTIFY PRIMARY KEY FOR STAFFBRANCH RELATION

- StaffBranch relation has five functional dependencies (slide 21).
- The determinants are staffNo, branchNo, bAddress, (branchNo, position), and (bAddress, position).
- To identify all candidate key(s), identify the attribute (or group of attributes) that uniquely identifies each tuple in this relation.
- All attributes that are not part of a candidate key should be functionally dependent on the key.
- The only candidate key and therefore primary key for StaffBranch relation, is staffNo, as all other attributes of the relation are functionally dependent on staffNo.

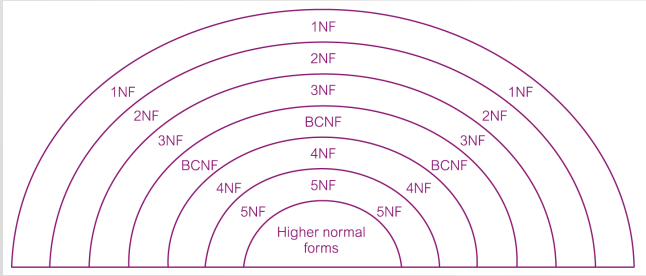
EXAMPLE - IDENTIFYING PRIMARY KEY FOR SAMPLE RELATION

- Sample relation has four functional dependencies (slide 21).
- The determinants in the Sample relation are A, B, C, and (A, B). However, the only determinant that functionally determines all the other attributes of the relation is (A, B).
- (A, B) is identified as the primary key for this relation.

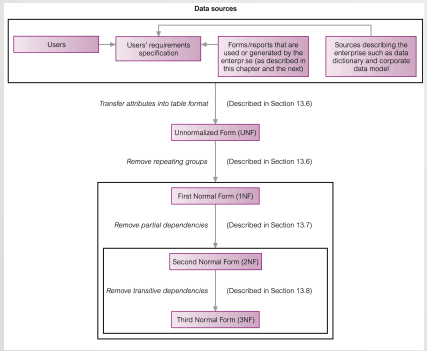
THE PROCESS OF NORMALIZATION

- As normalization proceeds, the relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.

THE PROCESS OF NORMALIZATION



THE PROCESS OF NORMALIZATION



UNNORMALIZED FORM (UNF)

- A table that contains one or more repeating groups.
- To create an unnormalized table
  - Transform the data from the information source (e.g. form) into table format with columns and rows.

FIRST NORMAL FORM (1NF)

- A relation in which the intersection of each row and column contains one and only one value.

UNF TO 1NF

- Nominate an attribute or group of attributes to act as the key for the unnormalized table.
- Identify the repeating group(s) in the unnormalized table which repeats for the key attribute(s).
- Remove the repeating group by
  - Entering appropriate data into the empty columns of rows containing the repeating data ('flattening' the table).
  - Or by
  - Placing the repeating data along with a copy of the original key attribute(s) into a separate relation.

SECOND NORMAL FORM (2NF)

- Based on the concept of full functional dependency.
- Full functional dependency indicates that if
  - A and B are attributes of a relation,
  - B is fully dependent on A if B is functionally dependent on A but not on any proper subset of A.
- A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on the primary key.

1NF TO 2NF

- Identify the primary key for the 1NF relation.
- Identify the functional dependencies in the relation.
- If partial dependencies exist on the primary key remove them by placing then in a new relation along with a copy of their determinant.

THIRD NORMAL FORM (3NF)

- Based on the concept of transitive dependency.
- Transitive Dependency is a condition where
  - A, B and C are attributes of a relation such that if A → B and B → C,
  - then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C).
- A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

2NF TO 3NF

- Identify the primary key in the 2NF relation.
- Identify functional dependencies in the relation.
- If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their dominant.

GENERAL DEFINITIONS OF 2NF AND 3NF

- Second normal form (2NF)
  - A relation that is in first normal form and every non-primary-key attribute is fully functionally dependent on any candidate key.
- Third normal form (3NF)
  - A relation that is in first and second normal form and in which no non-primary-key attribute is transitively dependent on any candidate key.

CSE 204 - INTRO TO DATABASE SYSTEMS ADVANCED NORMALIZATION

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- How inference rules can identify a set of all functional dependencies for a relation.
- How Inference rules called Armstrong's axioms can identify a minimal set of useful functional dependencies from the set of all functional dependencies for a relation.
- Normal forms that go beyond Third Normal Form (3NF), which includes Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF).
- How to identify Boyce–Codd Normal Form (BCNF).
- How to represent attributes shown on a report as BCNF relations using normalization.

OUTLINE

- Concept of multi-valued dependencies and Fourth Normal Form (4NF).
- The problems associated with relations that break the rules of 4NF.
- How to create 4NF relations from a relation, which breaks the rules of to 4NF.
- Concept of join dependency and Fifth Normal Form (5NF).
- The problems associated with relations that break the rules of 5NF.
- How to create 5NF relations from a relation, which breaks the rules of 5NF.

MORE ON FUNCTIONAL DEPENDENCIES

- The complete set of functional dependencies for a given relation can be very large.
- Important to find an approach that can reduce the set to a manageable size.

INFERENCE RULES FOR FUNCTIONAL DEPENDENCIES

- Need to identify a set of functional dependencies (represented as X) for a relation that is smaller than the complete set of functional dependencies (represented as Y) for that relation and has the property that every functional dependency in Y is implied by the functional dependencies in X.
- The set of all functional dependencies that are implied by a given set of functional dependencies X is called the closure of X, written X+ .
- A set of inference rules, called Armstrong's axioms, specifies how new functional dependencies can be inferred from given ones.

INFERENCE RULES FOR FUNCTIONAL DEPENDENCIES

- Let A, B, and C be subsets of the attributes of the relation R. Armstrong's axioms are as follows:

1. Reflexivity
   - If B is a subset of A, then A → B
2. Augmentation
   - If A → B, then A,C → B,C
3. Transitivity
   - If A → B and B → C, then A → C

INFERENCE RULES FOR FUNCTIONAL DEPENDENCIES

- Further rules can be derived from the first three rules that simplify the practical task of computing X+. Let D be another subset of the attributes of relation R, then:

4. Self-determination
   - A → A
5. Decomposition
   - If A → B,C, then A → B and A → C
6. Union
   - If A → B and A → C, then A → B,C
7. Composition
   - If A → B and C → D then A,C → B,D

MINIMAL SETS OF FUNCTIONAL DEPENDENCIES

- A set of functional dependencies Y is covered by a set of functional dependencies X, if every functional dependency in Y is also in X+; that is, every dependency in Y can be inferred from X.
- A set of functional dependencies X is minimal if it satisfies the following conditions:
  - Every dependency in X has a single attribute on its right-hand side.
- We cannot replace any dependency A → B in X with dependency C → B, where C is a proper subset of A, and still have a set of dependencies that is equivalent to X.
- We cannot remove any dependency from X and still have a set of dependencies that is equivalent to X.

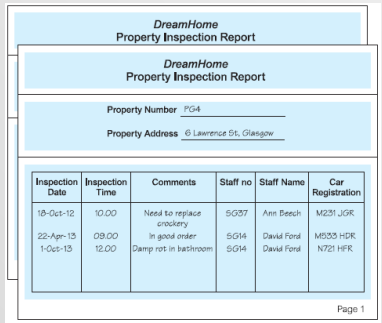
BOYCE–CODD NORMAL FORM (BCNF)

- Based on functional dependencies that take into account all candidate keys in a relation, however BCNF also has additional constraints compared with the general definition of 3NF.
- Boyce–Codd normal form (BCNF)
  - A relation is in BCNF if and only if every determinant is a candidate key.
- Difference between 3NF and BCNF is that for a functional dependency A -> B, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key. Whereas, BCNF insists that for this dependency to remain in a relation, A must be a candidate key.
  - Every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.

BOYCE–CODD NORMAL FORM (BCNF)

- Violation of BCNF is quite rare.
- The potential to violate BCNF may occur in a relation that:
  - contains two (or more) composite candidate keys;
  - the candidate keys overlap, that is have at least one attribute in common.

REVIEW OF NORMALIZATION (UNF TO BCNF)



DreamHome
Property Inspection Report

DreamHome
Property Inspection Report

Property Number PG4

Property Address 6 Lawrence St, Glasgow

Inspection Date	Inspection Time	Comments	Staff no	Staff Name	Car Registration
18-Oct-12	10.00	Need to replace crockery	SG37	Ann Beech	M231 JGR
22-Apr-13	09.00	In good order	SG14	David Ford	M533 HDR
1-Oct-13	12.00	Damp rot in bathroom	SG14	David Ford	N721 HFR

Page 1

REVIEW OF NORMALIZATION (UNF TO BCNF)

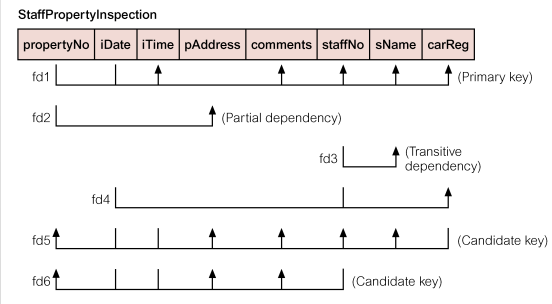
StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-12 22-Apr-13 1-Oct-13	10.00 09.00 12.00	Need to replace crockery In good order Damp rot in bathroom	SG37 SG14 SG14	Ann Beech David Ford David Ford	M231 JGR M533 HDR N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-13 24-Oct-13	13.00 14.00	Replace living room carpet Good condition	SG14 SG37	David Ford Ann Beech	M533 HDR N721 HFR

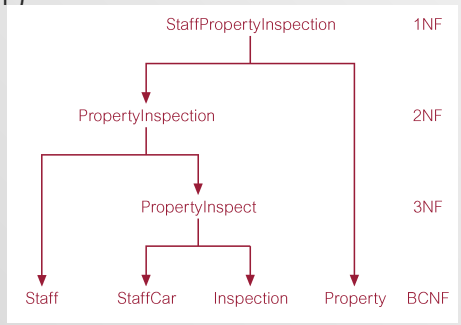
StaffPropertyInspection

propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-12	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-13	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	1-Oct-13	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-13	13.00	5 Novar Dr, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-13	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

REVIEW OF NORMALIZATION (UNF TO BCNF)



REVIEW OF NORMALIZATION (UNF TO BCNF)



FOURTH NORMAL FORM (4NF)

- Although BCNF removes anomalies due to functional dependencies, another type of dependency called a multi-valued dependency (MVD) can also cause data redundancy.
- Possible existence of multi-valued dependencies in a relation is due to 1NF and can result in data redundancy.
- Multi-valued Dependency (MVD)
  - Dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, the set of values for B and C are independent of each other.

FOURTH NORMAL FORM (4NF)

- MVD between attributes A, B, and C in a relation using the following notation:
  - A –>> B
  - A –>> C
- A multi-valued dependency can be further defined as being trivial or nontrivial.

  A MVD A –>> B in relation R is defined as being trivial if (a) B is a subset of A or (b) A U B = R.

  A MVD is defined as being nontrivial if neither (a) nor (b) are satisfied.

  A trivial MVD does not specify a constraint on a relation, while a nontrivial MVD does specify a constraint.
- Defined as a relation that is in Boyce-Codd Normal Form and contains no nontrivial multi-valued dependencies.

4NF - EXAMPLE

BranchStaffOwner

branchNo	sName	oName
B003	Ann Beech	Carol Farrel
B003	David Ford	Carol Farrel
B003	Ann Beech	Tina Murphy
B003	David Ford	Tina Murphy

BranchStaff

branchNo	sName
B003	Ann Beech
B003	David Ford

BranchOwner

branchNo	oName
B003	Carol Farrel
B003	Tina Murphy

FIFTH NORMAL FORM (5NF)

- A relation decompose into two relations must have the lossless-join property, which ensures that no spurious tuples are generated when relations are reunited through a natural join operation.
- However, there are requirements to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and fifth normal form (5NF).
- Defined as a relation that has no join dependency.

5NF - EXAMPLE

(a) **PropertyItemSupplier** (Illegal state)

propertyNo	itemDescription	supplierNo
PG4	Bed	S1
PG4	Chair	S2
PG16	Bed	S2

When this tuple is added to relation.

(b) **PropertyItemSupplier** (Legal state)

propertyNo	itemDescription	supplierNo
PG4	Bed	S1
PG4	Chair	S2
PG16	Bed	S2
PG4	Bed	S2

This new tuple must also be added to exist in any legal state of the relation.

5NF - EXAMPLE

PropertyItem

propertyNo	itemDescription
PG4	Bed
PG4	Chair
PG16	Bed

ItemSupplier

itemDescription	supplierNo
Bed	S1
Chair	S2
Bed	S2

PropertySupplier

propertyNo	supplierNo
PG4	S1
PG4	S2
PG16	S2

CSE 204 - INTRO TO DATABASE SYSTEMS CONCEPTUAL DATABASE DESIGN

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- The purpose of a design methodology.
- Database design has three main phases: conceptual, logical, and physical design.
- How to decompose the scope of the design into specific views of the enterprise.
- How to use Entity–Relationship (ER) modeling to build a conceptual data model based on the data requirements of an enterprise.
- How to validate the resultant conceptual model to ensure it is a true and accurate representation of the data requirements enterprise.
- How to document the process of conceptual database design.
- End-users play an integral role throughout the process of conceptual database design.

DESIGN METHODOLOGY

- A structured approach that uses procedures, techniques, tools, and documentation aids to support and facilitate the process of design.

DATABASE DESIGN METHODOLOGY

- Three main phases
  - Conceptual database design
  - Logical database design
  - Physical database design

CONCEPTUAL DATABASE DESIGN

- The process of constructing a model of the data used in an enterprise, independent of all physical considerations.

LOGICAL DATABASE DESIGN

- The process of constructing a model of the data used in an enterprise based on a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.

PHYSICAL DATABASE DESIGN

- The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes design used to achieve efficient access to the data, and any associated integrity constraints and security measures.

CRITICAL SUCCESS FACTORS IN DATABASE DESIGN

- Work interactively with the users as much as possible.
- Follow a structured methodology throughout the data modeling process.
- Employ a data-driven approach.
- Incorporate structural and integrity considerations into the data models.
- Combine conceptualization, normalization, and transaction validation techniques into the data modeling methodology.

CRITICAL SUCCESS FACTORS IN DATABASE DESIGN

- Use diagrams to represent as much of the data models as possible.
- Use a Database Design Language (DBDL) to represent additional data semantics.
- Build a data dictionary to supplement the data model diagrams.
- Be willing to repeat steps.

OVERVIEW DATABASE DESIGN METHODOLOGY

Conceptual database design

- Step 1 Build conceptual data model
  - Step 1.1 Identify entity types
  - Step 1.2 Identify relationship types
  - Step 1.3 Identify and associate attributes with entity or relationship types
  - Step 1.4 Determine attribute domains
  - Step 1.5 Determine candidate, primary, and alternate key attributes
  - Step 1.6 Consider use of enhanced modeling concepts (optional step)
  - Step 1.7 Check model for redundancy
  - Step 1.8 Validate conceptual model against user transactions
  - Step 1.9 Review conceptual data model with user

OVERVIEW DATABASE DESIGN METHODOLOGY

Logical database design

- Step 2 Build and validate logical data model
  - Step 2.1 Derive relations for logical data model
  - Step 2.2 Validate relations using normalization
  - Step 2.3 Validate relations against user transactions
  - Step 2.4 Define integrity constraints
  - Step 2.5 Review logical data model with user
  - Step 2.6 Merge logical data models into global model (optional step)
  - Step 2.7 Check for future growth

OVERVIEW DATABASE DESIGN METHODOLOGY

Physical database design

- Step 3 Translate logical data model for target DBMS
  - Step 3.1 Design base relations
  - Step 3.2 Design representation of derived data
  - Step 3.3 Design general constraints
- Step 4 Design file organizations and indexes
  - Step 4.1 Analyze transactions
  - Step 4.2 Choose file organization
  - Step 4.3 Choose indexes
  - Step 4.4 Estimate disk space requirements

OVERVIEW DATABASE DESIGN METHODOLOGY

- Step 5 Design user views
- Step 6 Design security mechanisms
- Step 7 Consider the introduction of controlled redundancy
- Step 8 Monitor and tune the operational system

STEP 1 BUILD CONCEPTUAL DATA

- To build a conceptual data model of the data requirements of the enterprise.
  - Model comprises entity types, relationship types, attributes and attribute domains, primary and alternate keys, and integrity constraints.
- Step 1.1 Identify entity types
  - To identify the required entity types.
- Step 1.2 Identify relationship types
  - To identify the important relationships that exist between the entity types.

STEP 1 BUILD CONCEPTUAL DATA

- Step 1.3 Identify and associate attributes with entity or relationship types
  - To associate attributes with the appropriate entity or relationship types and document the details of each attribute.
- Step 1.4 Determine attribute domains
  - To determine domains for the attributes in the data model and document the details of each domain.
- Step 1.5 Determine candidate, primary, and alternate key attributes
  - To identify the candidate key(s) for each entity and if there is more than one candidate key, to choose one to be the primary key and the others as alternate keys.
- Step 1.6 Consider use of enhanced modeling concepts (optional step)
  - To consider the use of enhanced modeling concepts, such as specialization / generalization, aggregation, and composition.

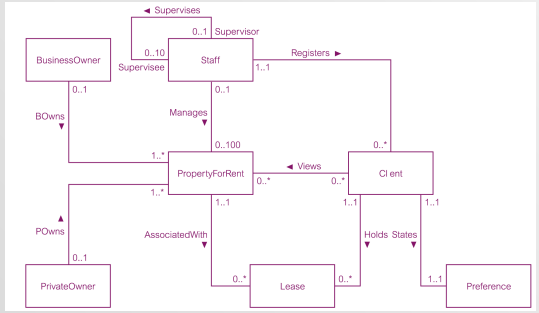
STEP 1 BUILD CONCEPTUAL DATA

- Step 1.7 Check model for redundancy
  - To check for the presence of any redundancy in the model and to remove any that does exist.
- Step 1.8 Validate conceptual model against user transactions
  - To ensure that the conceptual model supports the required transactions.
- Step1.9 Review conceptual data model with user
  - To review the conceptual data model with the user to ensure that the model is a 'true' representation of the data requirements of the enterprise.

EXTRACT FROM DATA DICTIONARY FOR STAFF USER VIEWS OF DREAMHOME SHOWING DESCRIPTION OF ENTITIES

Entity name	*Description*	*Aliases*	*Occurrence*
Staff	General term descr bing al staff employed by *DreamHome*.	Employee	Each member of staff works at one particular branch.
PropertyForRent	General term descr bing all property for rent.	Property	Each property has a s ngle owner and is available at one specific branch, where the property is managed by one member of staff. A property is viewed by many clients and rented by a single client, at any one time.

FIRST-CUT ER DIAGRAM FOR STAFF USER VIEWS OF DREAMHOME



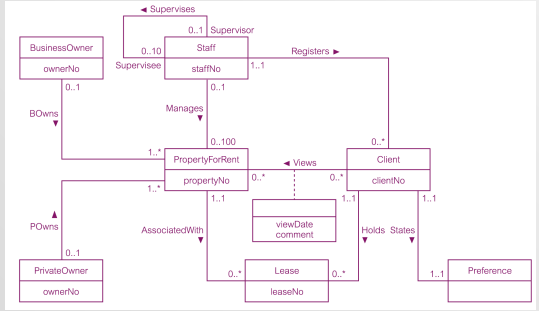
EXTRACT FROM DATA DICTIONARY FOR STAFF USER VIEWS OF DREAMHOME SHOWING DESCRIPTION OF RELATIONSHIPS

Entity name	*Multiplicity*	*Relationship*	*Multiplicity*	*Entity name*
Staff	0..1 0..1	*Manages* *Supervises*	0..100 0..10	**PropertyForRent** **Staff**
PropertyForRent	1..1	*AssociatedWith*	0..*	**Lease**

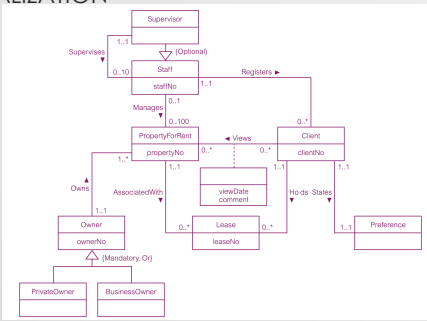
EXTRACT FROM DATA DICTIONARY FOR STAFF USER VIEWS OF DREAMHOME SHOWING DESCRIPTION OF ATTRIBUTES

Entity name	*Attributes*	*Description*	*Data Type & Length*	*Nulls*	*Multi-valued*	...
Staff	**staffNo**	Unique y identifies a member of staff	5 variable characters	No	No	
	name					
	fName	First name of staff	15 variable characters	No	No	
	lName	Last name of staff	15 variable characters	No	No	
	position	Job title of member of staff	10 variable characters	No	No	
	sex	Gender of member of staff	1 character (M or F)	Yes	No	
	DOB	Date of birth of member of staff	Date	Yes	No	
PropertyForRent	**propertyNo**	Unique y identifies a property for rent	5 variable characters	No	No	

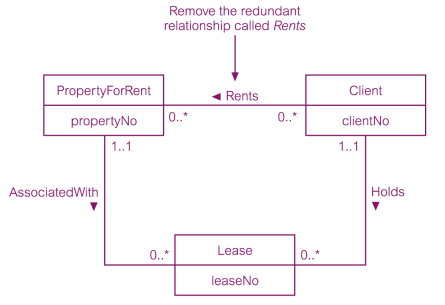
ER DIAGRAM FOR STAFF USER VIEWS OF DREAMHOME WITH PRIMARY KEYS ADDED



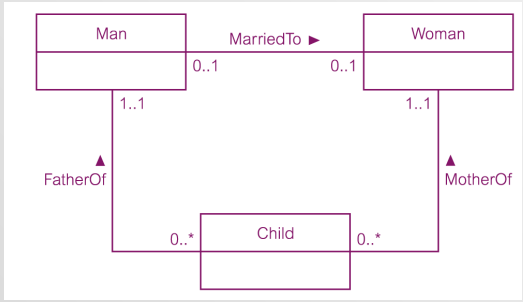
REVISED ER DIAGRAM FOR STAFF USER VIEWS OF DREAMHOME WITH SPECIALIZATION / GENERALIZATION



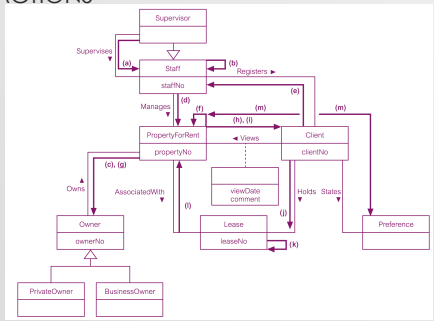
EXAMPLE OF REMOVING A REDUNDANT RELATIONSHIP CALLED RENTS



EXAMPLE OF A NON-REDUNDANT RELATIONSHIP FATHEROF



USING PATHWAYS TO CHECK THAT THE CONCEPTUAL MODEL SUPPORTS THE USER TRANSACTIONS



CSE 204 - INTRO TO DATABASE SYSTEMS LOGICAL DATABASE DESIGN

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

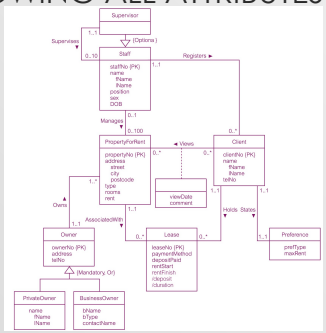
OUTLINE

- How to derive a set of relations from a conceptual data model.
- How to validate these relations using the technique of normalization.
- How to validate a logical data model to ensure it supports the required transactions.
- How to merge local logical data models based on one or more user views into a global logical data model that represents all user views.
- How to ensure that the final logical data model is a true and accurate representation of the data requirements of the enterprise

STEP 2 BUILD AND VALIDATE LOGICAL DATA MODEL

- To translate the conceptual data model into a logical data model and then to validate this model to check that it is structurally correct using normalization and supports the required transactions.
- Step 2.1 Derive relations for logical data model
  - To create relations for the logical data model to represent the entities, relationships, and attributes that have been identified.

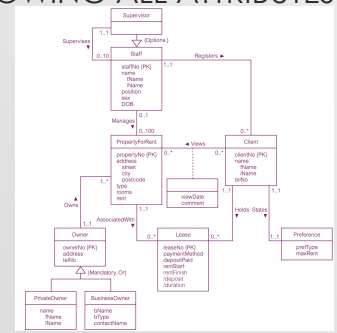
CONCEPTUAL DATA MODEL FOR STAFF VIEW SHOWING ALL ATTRIBUTES



STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

1. Strong entity types
   - For each strong entity in the data model, create a relation that includes all the simple attributes of that entity. For composite attributes, include only the constituent simple attributes.
2. Weak entity types
   - For each weak entity in the data model, create a relation that includes all the simple attributes of that entity. The primary key of a weak entity is partially or fully derived from each owner entity and so the identification of the primary key of a weak entity cannot be made until after all the relationships with the owner entities have been mapped.
3. One-to-many (1:*) binary relationship types
   - For each 1:* binary relationship, the entity on the 'one side' of the relationship is designated as the parent entity and the entity on the 'many side' is designated as the child entity. To represent this relationship, post a copy of the primary key attribute(s) of parent entity into the relation representing the child entity, to act as a foreign key.

CONCEPTUAL DATA MODEL FOR STAFF VIEW SHOWING ALL ATTRIBUTES



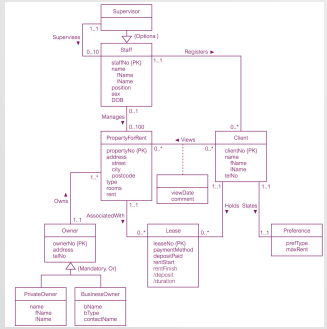
STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

4. One-to-one (1:1) binary relationship types
   - Creating relations to represent a 1:1 relationship is more complex as the cardinality cannot be used to identify the parent and child entities in a relationship. Instead, the participation constraints are used to decide whether it is best to represent the relationship by combining the entities involved into one relation or by creating two relations and posting a copy of the primary key from one relation to the other.
   - Consider the following
     - mandatory participation on both sides of 1:1 relationship;
     - mandatory participation on one side of 1:1 relationship;
     - optional participation on both sides of 1:1 relationship.

STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

- Mandatory participation on both sides of 1:1 relationship
  - Combine entities involved into one relation and choose one of the primary keys of original entities to be primary key of the new relation, while the other (if one exists) is used as an alternate key.
- Mandatory participation on one side of a 1:1 relationship
  - Identify parent and child entities using participation constraints. Entity with optional participation in relationship is designated as parent entity, and entity with mandatory participation is designated as child entity. A copy of primary key of the parent entity is placed in the relation representing the child entity. If the relationship has one or more attributes, these attributes should follow the posting of the primary key to the child relation.
- Optional participation on both sides of a 1:1 relationship
  - In this case, the designation of the parent and child entities is arbitrary unless we can find out more about the relationship that can help a decision to be made one way or the other.

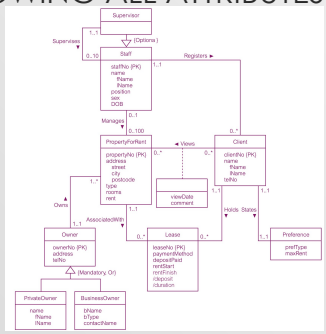
CONCEPTUAL DATA MODEL FOR STAFF VIEW SHOWING ALL ATTRIBUTES



STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

5. One-to-one (1:1) recursive relationships
   - For a 1:1 recursive relationship, follow the rules for participation as described above for a 1:1 relationship.
     - mandatory participation on both sides, represent the recursive relationship as a single relation with two copies of the primary key.
     - mandatory participation on only one side, option to create a single relation with two copies of the primary key, or to create a new relation to represent the relationship. The new relation would only have two attributes, both copies of the primary key. As before, the copies of the primary keys act as foreign keys and have to be renamed to indicate the purpose of each in the relation.
     - optional participation on both sides, again create a new relation as described above.

CONCEPTUAL DATA MODEL FOR STAFF VIEW SHOWING ALL ATTRIBUTES



STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

6. Superclass/subclass relationship types
   - Identify superclass entity as parent entity and subclass entity as the child entity. There are various options on how to represent such a relationship as one or more relations.
   - The selection of the most appropriate option is dependent on a number of factors such as the disjointness and participation constraints on the superclass/subclass relationship, whether the subclasses are involved in distinct relationships, and the number of participants in the superclass/subclass relationship.

GUIDELINES FOR REPRESENTATION OF SUPERCLASS / SUBCLASS RELATIONSHIP

Participation constraint	Disjoint constraint	Relations required
Mandatory	Nondisjoint {And}	Single relation (with one or more discriminators to distinguish the type of each tuple)
Optional	Nondisjoint {And}	Two relations: one relation for superclass and one relation for all subclasses (with one or more discriminators to distinguish the type of each tuple)
Mandatory	Disjoint {Or}	Many relations: one relation for each combined superclass/subclass
Optional	Disjoint {Or}	Many relations: one relation for superclass and one for each subclass

REPRESENTATION OF SUPERCLASS / SUBCLASS RELATIONSHIP BASED ON PARTICIPATION AND DISJOINTNESS

Option 1 – Mandatory, nondisjoint

AllOwner (ownerNo, address, telNo, fName, lName, bName, bType, contactName, pOwnerFlag, bOwnerFlag)
Primary Key ownerNo

Option 2 – Optional, nondisjoint

Owner (ownerNo, address, telNo)
Primary Key ownerNo

OwnerDetails (ownerNo, fName lName, bName, bType, contactName, pOwnerFlag, bOwnerFlag)
Primary Key ownerNo
Foreign Key ownerNo **references** Owner(ownerNo)

Option 3 – Mandatory, disjoint

PrivateOwner (ownerNo, fName, Name, address, telNo)
Primary Key ownerNo

BusinessOwner (ownerNo, bName, bType, contactName, address, telNo)
Primary Key ownerNo

Option 4 – Optional, disjoint

Owner (ownerNo, address, telNo)
Primary Key ownerNo

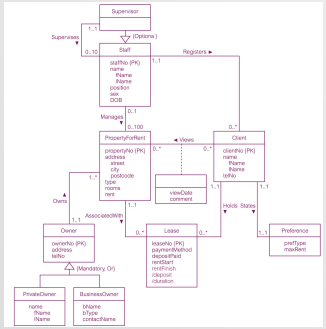
PrivateOwner (ownerNo, fName, Name)
Primary Key ownerNo
Foreign Key ownerNo **references** Owner(ownerNo)

BusinessOwner (ownerNo, bName, bType, contactName)
Primary Key ownerNo
Foreign Key ownerNo **references** Owner(ownerNo)

STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

7. Many-to-many (*:*) binary relationship types
   - Create a relation to represent the relationship and include any attributes that are part of the relationship. We post a copy of the primary key attribute(s) of the entities that participate in the relationship into the new relation, to act as foreign keys. These foreign keys will also form the primary key of the new relation, possibly in combination with some of the attributes of the relationship.
8. Complex relationship types
   - Create a relation to represent the relationship and include any attributes that are part of the relationship. Post a copy of the primary key attribute(s) of the entities that participate in the complex relationship into the new relation, to act as foreign keys. Any foreign keys that represent a 'many' relationship (for example, 1..*, 0..*) generally will also form the primary key of this new relation, possibly in combination with some of the attributes of the relationship.

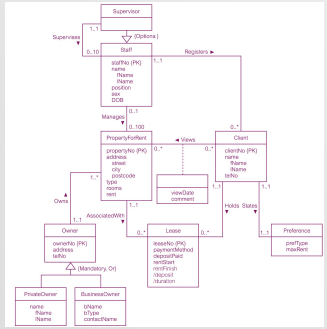
CONCEPTUAL DATA MODEL FOR STAFF VIEW SHOWING ALL ATTRIBUTES



STEP 2.1 DERIVE RELATIONS FOR LOGICAL DATA MODEL

9. Multi-valued attributes
   - Create a new relation to represent multi-valued attribute and include primary key of entity in new relation, to act as a foreign key. Unless the multi-valued attribute is itself an alternate key of the entity, the primary key of the new relation is the combination of the multi-valued attribute and the primary key of the entity.

CONCEPTUAL DATA MODEL FOR STAFF VIEW SHOWING ALL ATTRIBUTES



SUMMARY OF HOW TO MAP ENTITIES AND RELATIONSHIPS TO RELATIONS

Entity/Relationship	Mapping
Strong entity	Create relation that includes all simple attributes.
Weak entity	Create relation that includes all simple attributes (primary key still has to be identified after the relationship with each owner entity has been mapped).
1:* binary relationship	Post primary key of entity on 'one' side to act as foreign key in relation representing entity on 'many' side. Any attributes of relationship are also posted to 'many' side.
1:1 binary relationship: (a) Mandatory participation on both sides (b) Mandatory participation on one side	Combine entities into one relation. Post primary key of entity on 'optional' side to act as foreign key in relation representing entity on 'mandatory' side.
(c) Optional participation on both sides	Arbitrary without further information.
Superclass/subclass relationship	See Table 16.1.
: binary relationship, complex relationship	Create a relation to represent the relationship and include any attributes of the relationship. Post a copy of the primary keys from each of the owner entities into the new relation to act as foreign keys.
Multi-valued attribute	Create a relation to represent the multi-valued attribute and post a copy of the primary key of the owner entity into the new relation to act as a foreign key.

RELATIONS FOR THE STAFF USER VIEWS OF DREAMHOME

Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo)	**PrivateOwner** (ownerNo, fName, lName, address, telNo) **Primary Key** ownerNo
BusinessOwner (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo	**Client** (clientNo, fName, lName, telNo, prefType, maxRent, staffNo) **Primary Key** clientNo **Foreign Key** staffNo **references** Staff(staffNo)
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo)	**Viewing** (clientNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Cl ent(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo) **Primary Key** leaseNo **Alternate Key** propertyNo, rentStart **Alternate Key** clientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** deposit (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	

STEP 2.2 VALIDATE RELATIONS USING NORMALIZATION

- To validate the relations in the logical data model using normalization.

STEP 2.3 VALIDATE RELATIONS AGAINST USER TRANSACTIONS

- To ensure that the relations in the logical data model support the required transactions.

STEP 2.4 CHECK INTEGRITY CONSTRAINTS

- To check integrity constraints are represented in the logical data model. This includes identifying:
  - Required data
  - Attribute domain constraints
  - Multiplicity
  - Entity integrity
  - Referential integrity
  - General constraints

REFERENTIAL INTEGRITY CONSTRAINTS FOR RELATIONS IN STAFF USER VIEWS OF DREAMHOME

Staff (staffNo, fName, lName, position, sex, DOB, supervisorStaffNo)
Primary Key staffNo
Foreign Key supervisorStaffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL

Client (clientNo, fName, lName, telNo, prefType, maxRent, staffNo)
Primary Key clientNo
Foreign Key staffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE NO ACTION

PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo)
Primary Key propertyNo
Foreign Key ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo)
ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key staffNo **references** Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL

Viewing (clientNo, propertyNo, dateView, comment)
Primary Key clientNo, propertyNo
Foreign Key clientNo **references** Client(clientNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key propertyNo **references** PropertyForRent(propertyNo)
ON UPDATE CASCADE ON DELETE CASCADE

Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, cl entNo, propertyNo)
Primary Key leaseNo
Alternate Key propertyNo, rentStart
Alternate Key clientNo, rentStart
Foreign Key clientNo **references** Client(clientNo) ON UPDATE CASCADE ON DELETE NO ACTION
Foreign Key propertyNo **references** PropertyForRent(propertyNo)
ON UPDATE CASCADE ON DELETE NO ACTION

STEP 2.5 REVIEW LOGICAL DATA MODEL WITH USER

- To review the logical data model with the users to ensure that they consider the model to be a true representation of the data requirements of the enterprise.

STEP 2.6 MERGE LOGICAL DATA MODELS INTO GLOBAL MODEL (OPTIONAL STEP)

- To merge logical data models into a single global logical data model that represents all user views of a database.

STEP 2.6.1 MERGE LOCAL LOGICAL DATA MODELS INTO GLOBAL MODEL

- To merge local logical data model into a single global logical data model.
- This activities in this step include:
  - Step 2.6.1 Merge local logical data models into global model
  - Step 2.6.2 Validate global logical data model
  - Step 2.6.3 Review global logical data model with users.

STEP 2.6.1 MERGE LOGICAL DATA MODELS INTO A GLOBAL MODEL

- Tasks typically includes:
  1. Review the names and contents of entities/relations and their candidate keys.
  2. Review the names and contents of relationships/foreign keys.
  3. Merge entities/relations from the local data models
  4. Include (without merging) entities/relations unique to each local data model
  5. Merge relationships/foreign keys from the local data models.
  6. Include (without merging) relationships/foreign keys unique to each local data model.
  7. Check for missing entities/relations and relationships/foreign keys.
  8. Check foreign keys.
  9. Check Integrity Constraints.
  10. Draw the global ER/relation diagram
  11. Update the documentation.

STEP 2.6.2 VALIDATE GLOBAL LOGICAL DATA MODEL

- To validate the relations created from the global logical data model using the technique of normalization and to ensure they support the required transactions, if necessary.

STEP 2.6.3 REVIEW GLOBAL LOGICAL DATA MODEL WITH USERS

- To review the global logical data model with the users to ensure that they consider the model to be a true representation of the data requirements of an enterprise.

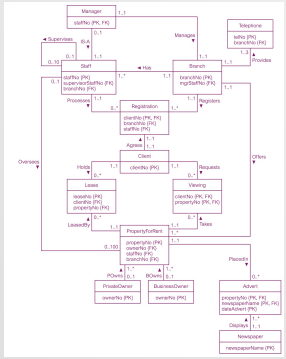
RELATIONS FOR THE BRANCH USER VIEWS OF DREAMHOME

Branch (branchNo, street, c ty, postcode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** postcode **Foreign Key** mgrStaffNo **references** Manager(staffNo)	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Staff (staffNo, name, position, salary, supervisorStaffNo, branchNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Manager** (staffNo, mgrStartDate, bonus) **Primary Key** staffNo **Foreign Key** staffNo **references** Staff(staffNo)
PrivateOwner (ownerNo, name, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (bName, bType, contactName, address, telNo) **Primary Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, bName, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) **Foreign Key** bName **references** BusinessOwner(bName) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Client** (clientNo, name, telNo, prefType, maxRent) **Primary Key** clientNo
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, clientNo, propertyNo) **Primary Key** leaseNo **Alternate Key** propertyNo, rentStart **Alternate Key** clientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** deposit (PropertyForRent.rent*2) **Derived** duration (rentFinish – rentStart)	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** cl entNo **references** C ient(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo

RELATIONS THAT REPRESENT THE GLOBAL LOGICAL DATA MODEL FOR DREAMHOME

Branch (branchNo, street, city, postcode, mgrStaffNo) **Primary Key** branchNo **Alternate Key** postcode **Foreign Key** mgrStaffNo **references** Manager(staffNo)	**Telephone** (telNo, branchNo) **Primary Key** telNo **Foreign Key** branchNo **references** Branch(branchNo)
Staff (staffNo, fName, lName, position, sex, DOB, salary, supervisorStaffNo, branchNo) **Primary Key** staffNo **Foreign Key** supervisorStaffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Manager** (staffNo, mgrStartDate, bonus) **Primary Key** staffNo **Foreign Key** staffNo **references** Staff(staffNo)
PrivateOwner (ownerNo, fName, lName, address, telNo) **Primary Key** ownerNo	**BusinessOwner** (ownerNo, bName, bType, contactName, address, telNo) **Primary Key** ownerNo **Alternate Key** bName **Alternate Key** telNo
PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo) **Primary Key** propertyNo **Foreign Key** ownerNo **references** PrivateOwner(ownerNo) and BusinessOwner(ownerNo) **Foreign Key** staffNo **references** Staff(staffNo) **Foreign Key** branchNo **references** Branch(branchNo)	**Viewing** (c telNo, propertyNo, dateView, comment) **Primary Key** clientNo, propertyNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo)
Client (clientNo, fName, lName, telNo, prefType, maxRent) **Primary Key** clientNo	**Registration** (clientNo, branchNo, staffNo, dateJoined) **Primary Key** clientNo **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** branchNo **references** Branch(branchNo) **Foreign Key** staffNo **references** Staff(staffNo)
Lease (leaseNo, paymentMethod, depositPaid, rentStart, rentFinish, cl entNo, propertyNo) **Primary Key** easeNo **Alternate Key** propertyNo, rentStart **Alternate Key** c ientNo, rentStart **Foreign Key** clientNo **references** Client(clientNo) **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Derived** deposi t (PropertyForRent.rent*2) **Derived** d uration (rentFinish – rentStart)	**Newspaper** (newspaperName, address, telNo, contactName) **Primary Key** newspaperName **Alternate Key** telNo
Advert (propertyNo, newspaperName, dateAdvert, cost) **Primary Key** propertyNo, newspaperName, dateAdvert **Foreign Key** propertyNo **references** PropertyForRent(propertyNo) **Foreign Key** newspaperName **references** Newspaper(newspaperName)	

GLOBAL RELATION DIAGRAM FOR DREAMHOME



CSE 204 - INTRO TO DATABASE SYSTEMS PHYSICAL DATABASE DESIGN

Joseph LEDET
Department of Computer Engineering
Akdeniz University
josephledet@akdeniz.edu.tr

OUTLINE

- Purpose of physical database design.
- How to map the logical database design to a physical database design.
- How to design base relations for target DBMS.
- How to design general constraints for target DBMS.
- How to select appropriate file organizations based on analysis of transactions.
- When to use secondary indexes to improve performance.
- How to estimate the size of the database.
- How to design user views.
- How to design security mechanisms to satisfy user requirements.

LOGICAL V. PHYSICAL DATABASE DESIGN

- Sources of information for physical design process includes logical data model and documentation that describes model.
- Logical database design is concerned with the what, physical database design is concerned with the how.

PHYSICAL DATABASE DESIGN

- Process of producing a description of the implementation of the database on secondary storage.
- It describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

OVERVIEW OF PHYSICAL DATABASE DESIGN METHODOLOGY

- Step 3 Translate logical data model for target DBMS
  - Step 3.1 Design base relations
  - Step 3.2 Design representation of derived data
  - Step 3.3 Design general constraints
- Step 4 Design file organizations and indexes
  - Step 4.1 Analyze transactions
  - Step 4.2 Choose file organizations
  - Step 4.3 Choose indexes
  - Step 4.4 Estimate disk space requirements

OVERVIEW OF PHYSICAL DATABASE DESIGN METHODOLOGY

- Step 5 Design user views
- Step 6 Design security mechanisms
- Step 7 Consider the introduction of controlled redundancy
- Step 8 Monitor and tune operational system

STEP 3 TRANSLATE LOGICAL DATA MODEL FOR TARGET DBMS

To produce a relational database schema from the logical data model that can be implemented in the target DBMS.

- Need to know functionality of target DBMS such as how to create base relations and whether the system supports the definition of:
  - PKs, FKs, and AKs;
  - required data – i.e. whether system supports NOT NULL;
  - domains;
  - relational integrity constraints;
  - general constraints.

STEP 3.1 DESIGN BASE RELATIONS

To decide how to represent base relations identified in logical model in target DBMS.

- For each relation, need to define:
  - the name of the relation;
  - a list of simple attributes in brackets;
  - the PK and, where appropriate, AKs and FKs.
  - referential integrity constraints for any FKs identified.

STEP 3.1 DESIGN BASE RELATIONS

- From data dictionary, we have for each attribute:
  - its domain, consisting of a data type, length, and any constraints on the domain;
  - an optional default value for the attribute;
  - whether it can hold nulls;
  - whether it is derived, and if so, how it should be computed.

DBDL FOR THE PROPERTYFORRENT RELATION

```
Domain PropertyNumber:        variable length character string, length 5
Domain Street:                variable length character string, length 25
Domain City:                  variable length character string, length 15
Domain Postcode:              variable length character string, length 8
Domain PropertyType:          single character, must be one of 'B', 'C', 'D', 'E', 'F', 'H', 'M', 'S'
Domain PropertyRooms:         integer, in the range 1–15
Domain PropertyRent:          monetary value, in the range 0.00–9999.99
Domain OwnerNumber:           variable length character string, length 5
Domain StaffNumber:           variable length character string, length 5
Domain BranchNumber:          fixed length character string, length 4
PropertyForRent(
    propertyNo      PropertyNumber      NOT NULL,
    street          Street              NOT NULL,
    city            City                NOT NULL,
    postcode        Postcode,
    type            PropertyType        NOT NULL DEFAULT 'F',
    rooms           PropertyRooms       NOT NULL DEFAULT 4,
    rent            PropertyRent        NOT NULL DEFAULT 600,
    ownerNo         OwnerNumber         NOT NULL,
    staffNo         StaffNumber,
    branchNo        BranchNumber        NOT NULL,
    PRIMARY KEY (propertyNo),
    FOREIGN KEY (staffNo) REFERENCES Staff(staffNo) ON UPDATE CASCADE ON DELETE SET NULL,
    FOREIGN KEY (ownerNo) REFERENCES PrivateOwner(ownerNo) and BusinessOwner(ownerNo)
                                ON UPDATE CASCADE ON DELETE NO ACTION,
    FOREIGN KEY (branchNo) REFERENCES Branch(branchNo)
                                ON UPDATE CASCADE ON DELETE NO ACTION);
```

STEP 3.2 DESIGN REPRESENTATION OF DERIVED DATA

To decide how to represent any derived data present in logical data model in target DBMS.

- Examine logical data model and data dictionary, and produce list of all derived attributes.
- Derived attribute can be stored in database or calculated every time it is needed.
- Option selected is based on:
  - additional cost to store the derived data and keep it consistent with operational data from which it is derived;
  - cost to calculate it each time it is required.
- Less expensive option is chosen subject to performance constraints.

PROPERTYFORRENT RELATION AND STAFF RELATION WITH DERIVED ATTRIBUTE NOOFPROPERTIES

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Staff

staffNo	fName	lName	branchNo	noOfProperties
SL21	John	White	B005	0
SG37	Ann	Beech	B003	2
SG14	David	Ford	B003	1
SA9	Mary	Howe	B007	1
SG5	Susan	Brand	B003	0
SL41	Julie	Lee	B005	1

STEP 3.3 DESIGN GENERAL CONSTRAINTS

To design the general constraints for target DBMS.

- Some DBMS provide more facilities than others for defining enterprise constraints. Example:

```
CONSTRAINT StaffNotHandlingTooMuch
CHECK (NOT EXISTS (SELECT staffNo
FROM PropertyForRent
GROUP BY staffNo
HAVING COUNT(*) > 100))
```


STEP 4 DESIGN FILE ORGANIZATIONS AND INDEXES

To determine optimal file organizations to store the base relations and the indexes that are required to achieve acceptable performance; that is, the way in which relations and tuples will be held on secondary storage.

- Must understand the typical workload that database must support.

STEP 4.1 ANALYZE TRANSACTIONS

To understand the functionality of the transactions that will run on the database and to analyze the important transactions.

- Attempt to identify performance criteria, such as:
  - transactions that run frequently and will have a significant impact on performance;
  - transactions that are critical to the business;
  - times during the day/week when there will be a high demand made on the database (called the peak load).
- Use this information to identify the parts of the database that may cause performance problems.
- Also need to know high-level functionality of the transactions, such as:
  - attributes that are updated;
  - search criteria used in a query.

STEP 4.1 ANALYZE TRANSACTIONS

- Often not possible to analyze all transactions, so investigate most 'important' ones.
- To help identify these can use:
  - transaction/relation cross-reference matrix, showing relations that each transaction accesses, and/or
  - transaction usage map, indicating which relations are potentially heavily used.
- To focus on areas that may be problematic:
  1. Map all transaction paths to relations.
  2. Determine which relations are most frequently accessed by transactions.
  3. Analyze the data usage of selected transactions that involve these relations.

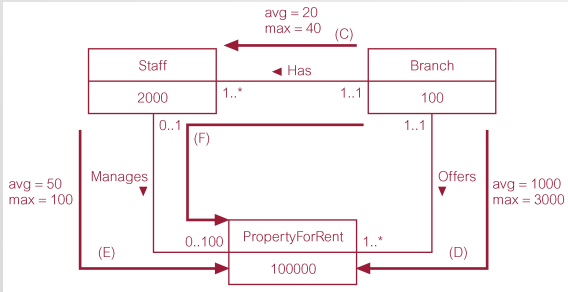
CROSS-REFERENCING TRANSACTIONS AND RELATIONS

Table 17.1 Cross-referencing transactions and relations.

Transaction/ Relation	(A)				(B)				(C)				(D)				(E)				(F)			
	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D	I	R	U	D
Branch										X				X								X		
Telephone																								
Staff		X				X				X								X				X		
Manager																								
PrivateOwner	X																							
BusinessOwner	X																							
PropertyForRent	X					X	X	X						X				X				X		
Viewing																								
Client																								
Registration																								
Lease																								
Newspaper																								
Advert																								

I = Insert; R = Read; U = Update; D = Delete

EXAMPLE TRANSACTION USAGE MAP



EXAMPLE TRANSACTION ANALYSIS FORM

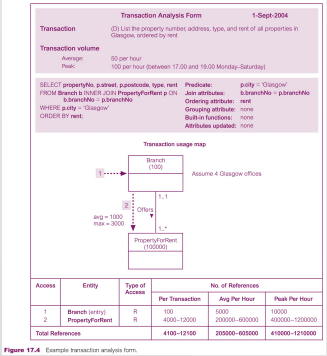


Figure 17.4 Example transaction analysis form.

STEP 4.2 CHOOSE FILE ORGANIZATIONS

To determine an efficient file organization for each base relation.

- File organizations include Heap, Hash, Indexed Sequential Access Method (ISAM), B+-Tree, and Clusters.
- Some DBMSs may not allow selection of file organizations.

STEP 4.3 CHOOSE INDEXES

To determine whether adding indexes will improve the performance of the system.

- One approach is to keep tuples unordered and create as many **secondary indexes** as necessary.
- Another approach is to order tuples in the relation by specifying a primary or clustering index.
- In this case, choose the attribute for ordering or clustering the tuples as:
  - attribute that is used most often for join operations - this makes join operation more efficient, or
  - attribute that is used most often to access the tuples in a relation in order of that attribute.

STEP 4.3 CHOOSE INDEXES

- If ordering attribute chosen is key of relation, index will be a primary index; otherwise, index will be a clustering index.
- Each relation can only have either a primary index or a clustering index.
- Secondary indexes provide a mechanism for specifying an additional key for a base relation that can be used to retrieve data more efficiently.

STEP 4.3 CHOOSE INDEXES

- Have to balance overhead involved in maintenance and use of secondary indexes against performance improvement gained when retrieving data.
- This includes:
  - adding an index record to every secondary index whenever tuple is inserted;
  - updating secondary index when corresponding tuple updated;
  - increase in disk space needed to store secondary index;
  - possible performance degradation during query optimization to consider all secondary indexes.

STEP 4.3 CHOOSE INDEXES – GUIDELINES FOR CHOOSING 'WISH-LIST'

1. Do not index small relations.
2. Index PK of a relation if it is not a key of the file organization.
3. Add secondary index to a FK if it is frequently accessed.
4. Add secondary index to any attribute heavily used as a secondary key.
5. Add secondary index on attributes involved in: selection or join criteria; ORDER BY; GROUP BY; and other operations involving sorting (such as UNION or DISTINCT).

STEP 4.3 CHOOSE INDEXES – GUIDELINES FOR CHOOSING 'WISH-LIST'

6. Add secondary index on attributes involved in built-in functions.
7. Add secondary index on attributes that could result in an index-only plan.
8. Avoid indexing an attribute or relation that is frequently updated.
9. Avoid indexing an attribute if the query will retrieve a significant proportion of the relation.
10. Avoid indexing attributes that consist of long character strings.

STEP 4.4 ESTIMATE DISK SPACE REQUIREMENTS

- To estimate the amount of disk space that will be required by the database.

STEP 5 DESIGN USER VIEWS

- To design the user views that were identified during the Requirements Collection and Analysis stage of the database system development lifecycle.

STEP 6 DESIGN SECURITY MEASURES

- To design the security measures for the database as specified by the users.