4CCS1DBS Database Systems

2ND SEMESTER | FIRST YEAR

KING'S COLLEGE LONDON |

Table of Contents

4CCS1DBS Database Systems

General Info

Lecturers:

- Dr Sophia Tsoka
- Mr Chipp Janssen

Lab Practicals

- 5 practicals

Coursework (15%)

- 4 Quizzes (1%) – every two weeks
- Major database coursework (10%)
- Participation in discussion group (1%)
- Relational Algebra Revision Coursework
    - Optional extra revision coursework (worth 0%)
- Deadlines

Exam is in May MCQ

Databases and Database Use

Basics

Database: a collection of related data.

Data: known facts that can be recorded and have an implicit meaning.

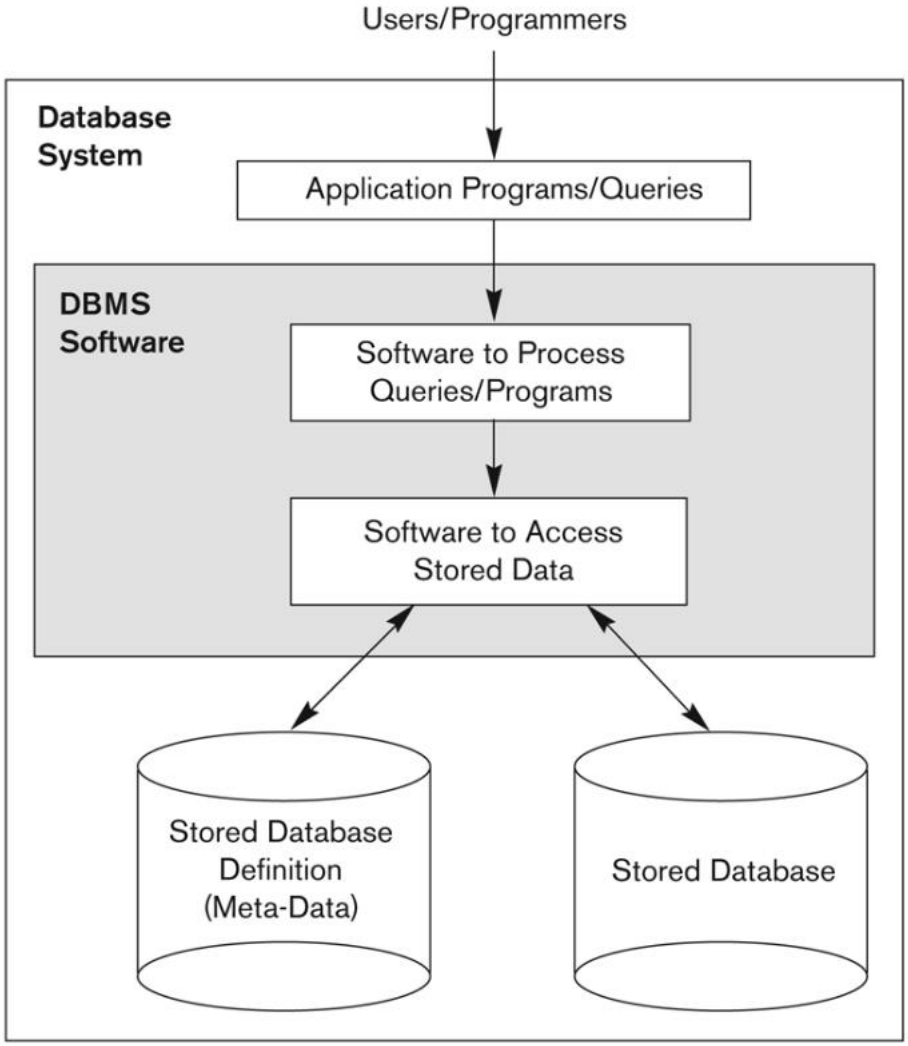
Mini-world: some part of the real world about which data is stored in a database (students' grades in university).

Database Management System (BDMS): A software package / systems to facilitate the creation and maintenance of a computerised database.

Database system: is the DBMS together with the data itself. Applications are also included (sometimes).

- It defines a database in terms of its data types, structures, and constraints.
- Constructs or loads the initial database contents on secondary storage medium.
- Manipulates the database:
    - Retrieval: queries and reports.
    - Modification: insertions, deletions and updates to its content.
    - Accessing the database through web applications.
- Processing and sharing data whilst keeping it valid and consistent.

User -> Applications / queries -> software to process queries and programs -> software to access stored data -> Stored database definition (meta data) & Stored database



Typical DBMS functionality

They:

- Define database in terms of its data types, structures, and constraints.
- Construct the initial database contents on a secondary store medium.
- Manipulates the database:
    - **Retrieval:** querying and generating reports.
    - **Modification:** insertions, deletions and updates to its content
    - **Accessing** the database through Web Applications
- Processing and sharing by the information using application programs to many users, keep the data valid and consistent.
- Other features: security measures to prevent unauthorised access, presentation of data, maintaining other programs associated to the database up to date.

Example: Database which monitors universities

Mini-world **UNIVERSITY**

Some of the mini world ***entities***:

- STUDENTs
- COURSEs
  SECTIONs (of coursers)
- (academic) DEPARTMENTs
- INSTRUCTORs

The above entities have relationships between them, this creates an ENTITY-RELATIONSHIP data model. For example, the COURSEs that are offered by DEPARTMENTs.

Main Characteristics of a database

Self-describing nature of a database system: a **catalog*** stores the description of a particular database (data structures, types and constraints). This description is called **meta-data** and it allows the DBMS software to work with different database applications.

database

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
.....	.....	.....
.....	.....	.....
.....	.....	.....
Prerequisite_number	XXXXNNNN	PREREQUISITE

*this is a simplified database catalog.

Insulation between programs and data: called **program-data independence**. This allows changing data structures and storage organisation without having to change the DBMS access programs.

Main Characteristics of the Database Approach

Data Abstraction: a **data model** is used to hide storage details and present the users with a conceptual view of the database. Programs refer to the data model constructs rather than the data storage details.

Support of multiple views of the data: each user might see a different view of the database that might interest him (for instance a bank manager or a person accessing their bank account).

Sharing of data and multi-user transaction processing: allows a set of concurrent users to retrieve from and to update the database.

> *Concurrency control* within the DBMS guarantees that each transaction is correctly executes or aborted.
>
> *Recovery* subsystem ensures that each completed transaction has its effect perfectly recorded in the database.
>
> **OLTP (Online Transaction Processing)** is a major part of the database applications. This allows hundreds of concurrent transactions to execute per second.

Advantages of using the Database Approach

- Controls redundancy in data storage
- Provides multiple interfaces to each different class of users
- Facilitates sharing data across users
- Restricts availability of data to certain users
- Provides backup and recovery services
- Presents complex relationships amongst data
- Enforcing of standards help interpret, understand and input the data efficiently.
- The time to add new features or applications becomes less since it is adding on rather than building
- Information is current
- **Economies of scale:** wasteful overlap of resources and personnel can be avoided by consolidating data and applications across departments.

Disadvantages of using the Database Approach

- High investment and possible need for additional hardware
- Maintenance costs are high

A DBMS may not be necessary when:

- If the database and applications are simple and they don't expect to change.
- There are real-time requirements which may be very costly to implement.
- If access to data by multiple users is not required.

A DBMS may not be enough:

- If the database system is not able to handle the complexity of the data because of modelling limitations
- If the database users need special operations not supported by the DBMS.

Database System Concepts and Architecture

Data model: a set of concepts that describe the structure of a database, the operations for manipulating these structures, and certain constrains that the database should obey.

- **Structure and constrains:** constructs are used to define the database structure, they include elements (and their data types) as well as groups of elements (e.g. entity, record, table) and relationships among such groups. Constraints specify some restrictions on all valid data; these must be enforced always.
- **Data Model Operations:** these are the operations used to specify the database retrievals and updates by referring to the constructs of the data model.
    - **Basic model operations**: generic insert, delete or update.
    - **User-defined operations:** e.g. compute_student_gpa

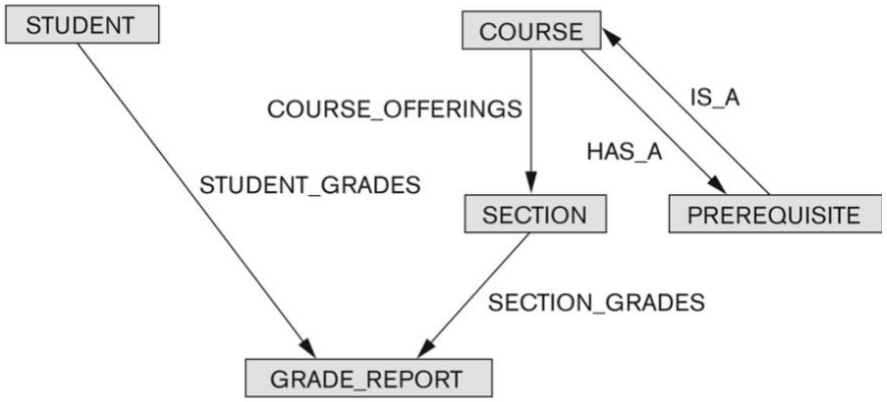
Categories of Data Models

- **Conceptual (high-level, semantic)**: provide concepts that are close to the way many users perceive data.
- **Physical (low-level, internal):** provide concepts that describe details of how data is stored in the computer. (These are usually specified in the manuals)
- **Implementation (representational)**: provide concepts are between both of the ones above. These are the most common for commercial use.

Types of Data Models

- Relational Model
- Network Model
- Hierarchical Model
- Object-oriented Data Models
- Object-Relational Models

Network Model



Advantages:

- Able to model complex relationships
- Language is navigational; uses constructs like FIND, FIND member, set, etc.

Disadvantages:

- Navigational and procedural nature of processing
- Database contains a complex array of pointers that thread through a set of record. There is little space for query optimisation

Relational Model

Database schema: a ***description*** of a database that includes the database structure, data types and constraints.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Schema diagram: an illustrative display of a database schema.

Schema construct: a component of the schema or an object within the schema.

Database State

Database state: the actual data stored in the database at a particular time (also called database instance, occurrence or snapshot).

Initial state: refers to the start when its initially loaded into the system.

Valid state: a state that satisfies the structure and constraints of the database.

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

Note that a database schema does not change frequently whilst a database state changes every time the database is updated.

Three-Schema Architecture

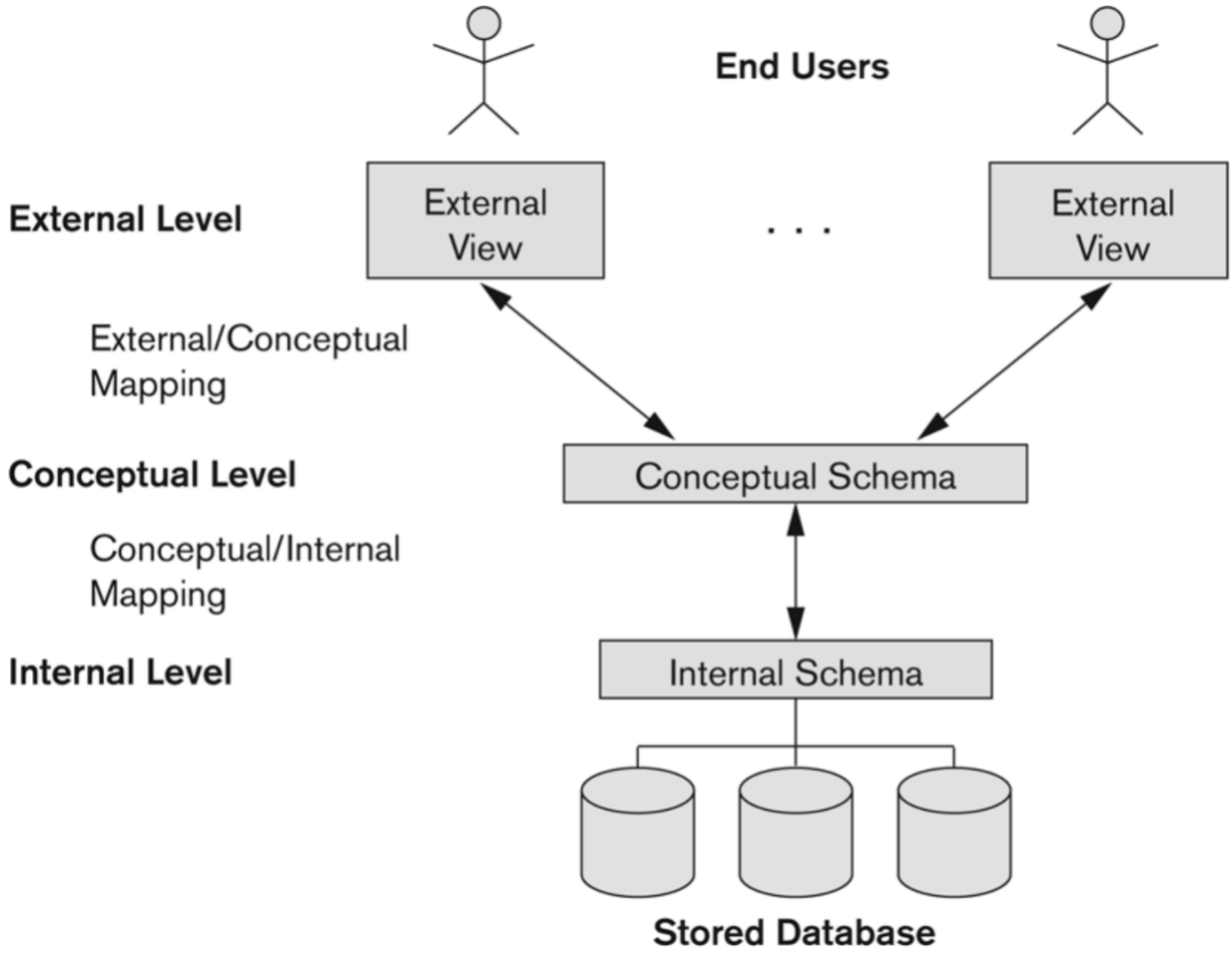
Proposed to support DBMS's program-data independence and multiple views of the data.

This is not explicitly used in commercial DBMS products but has been useful in explaining database system organisation.

It defines the DBMS schemas at three levels:

- **Internal schema:** describes physical storage structures and access paths (uses physical data model)

- **Conceptual schema:** describes the structure and constraints for the whole database for the community of users (uses conceptual or an implementation data model)
- **External schema:** describes the various user views.



DBMS Languages

Data Definition Language (DDL): used to specify conceptual and internal schemas

Data Manipulation Language (DML): used to query and manipulate database (includes SQL), these are used to specify retrievals and updates.

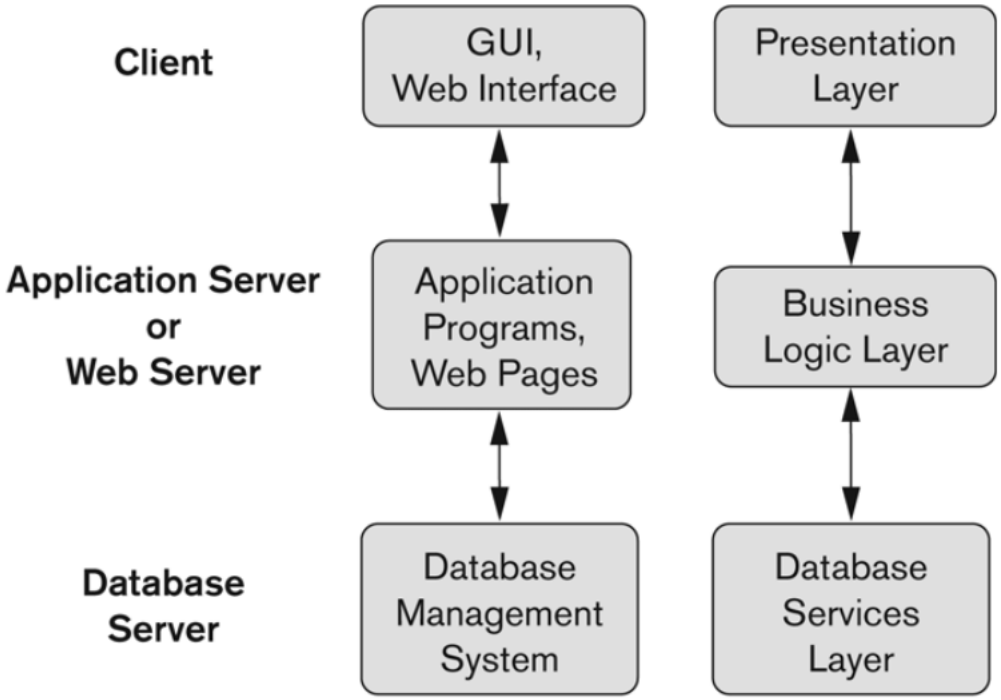
Data Definition Language (DDL): used by the DBA and database designers to specify the conceptual schema of a database.

Centralised and Client-Server DBMS Architectures

A centralised DBMS combines everything into a single system including DBMS software, hardware, application programs and user interface processing. The user can still connect through a remote terminal but all processing is done at a centralised site.

Three Tier Client-Server Architecture

This architecture allows the user to manipulate the database but the information is stored in a database server. Using this can enhance security as clients cannot directly access the database server.



Classification of DBMSs

Based on the data model used;

- Traditional: Relational, Network, Hierarchical.
- Emerging: Object-oriented, Object-relational.

Other classifications:

- Single-user (personal computer) vs. multi-user (most DBMSs)
- Centralised (uses a single computer with one database) vs. distributed (uses multiple computers/databases.

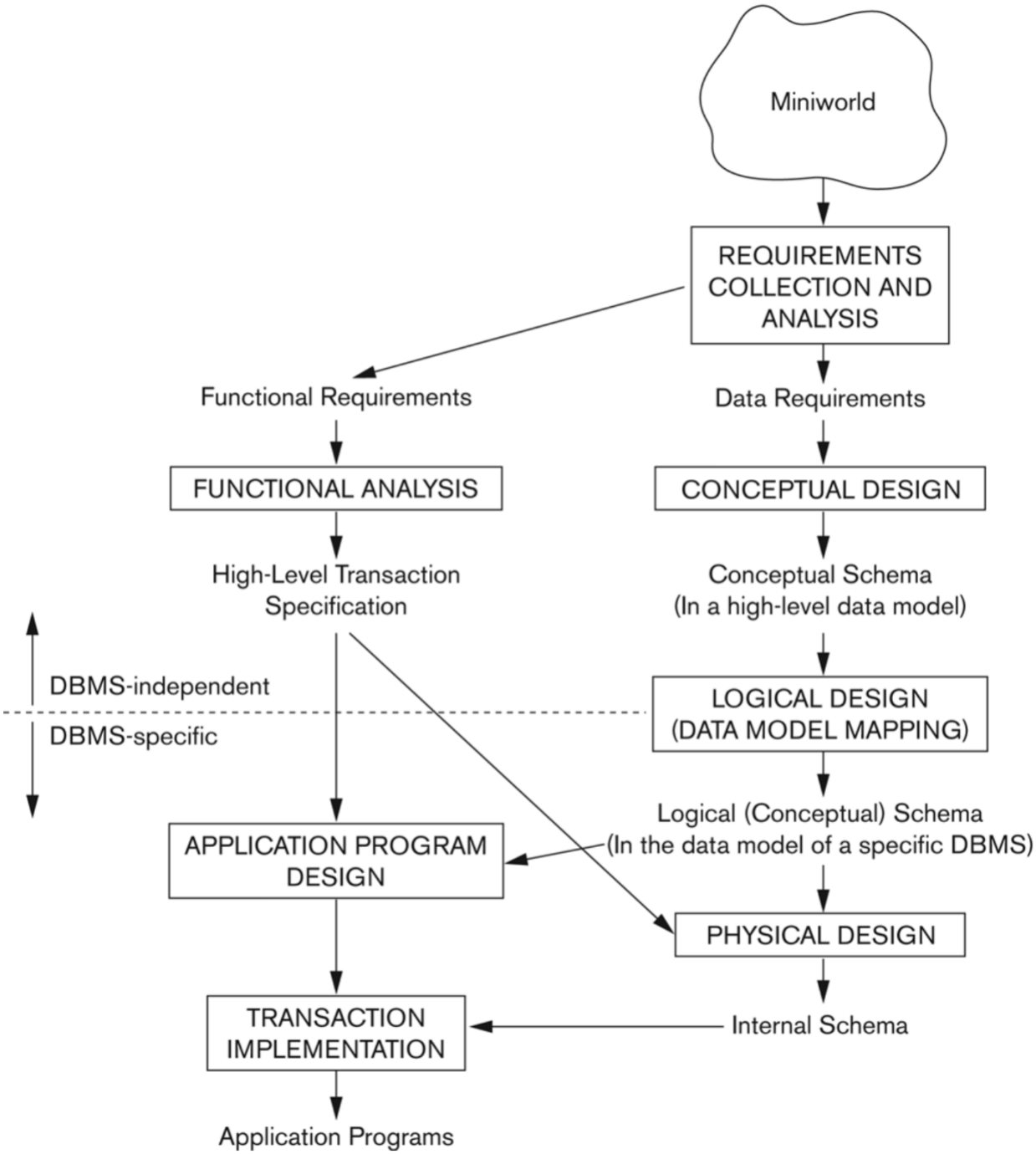
Variations of Distributed DBMSs

- Homogeneous: same DBMS in all sites
- Heterogeneous: several autonomous DMBS software at multiple sites, connected through a network
- Federated or Multi-database Systems: loosely coupled and have some autonomy.

Database Design Process

- Database design: design conceptual schema for a database application.

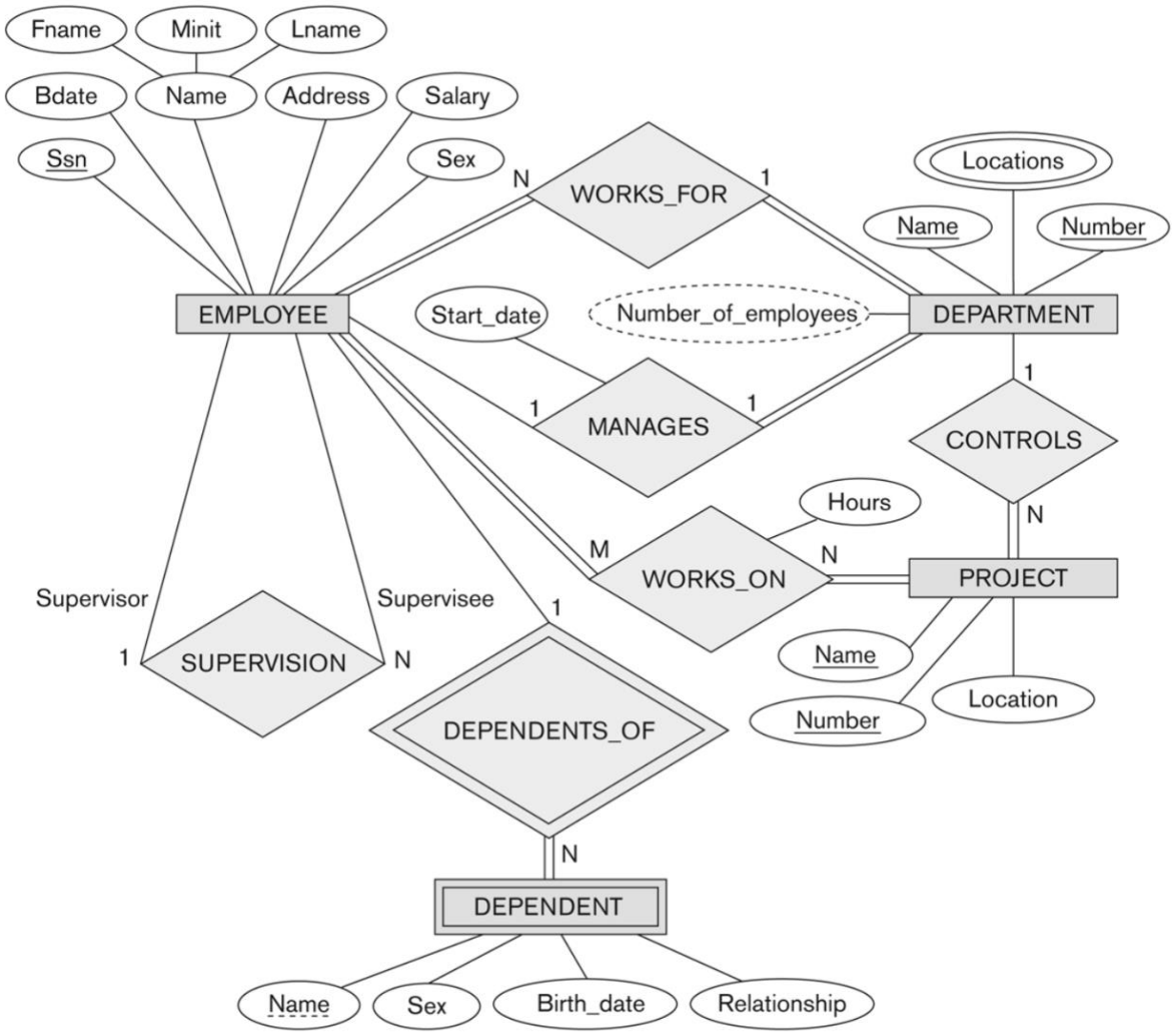
- Applications design: creating programs and interfaces that access the database (software engineering).



Example: COMPANY Database

- A company is separated into DEPARTMENTs which has a name, number and employee who manages the department. We keep track of the start date of the department management. A department may have several locations.
- Each department controls a number of PROJETCs. Each project has a unique name, unique number and is located at a single location.
- We store each EMPLOYEE's social security number, address, salary, gender and birthdate. Each employee works for one department but may work on several projects. We keep track of the number of hours per week that an employee currently works on each project. We also keep track of the direct supervisor of each employee.
- Each employee may have a number of DEPENDENTs, for each dependent, we keep track of their name, sex, birthdate and relationship to employee.

We are going to learn how to translate the English based information into the schematic form below.



Entities are specific objects or things in the mini-world that are represented in the database.

- The EMPLOYEE John Johnson
- The Research DEPARTMENT

Capitalized nouns are entities

All entities have **attributes**, properties.

- Say for John Johnson:
  - Name='John Smith', SSN='29930928', Address='731, Fonner, Houston, TX', Sex='M', Birthdate='09-JAN-55'

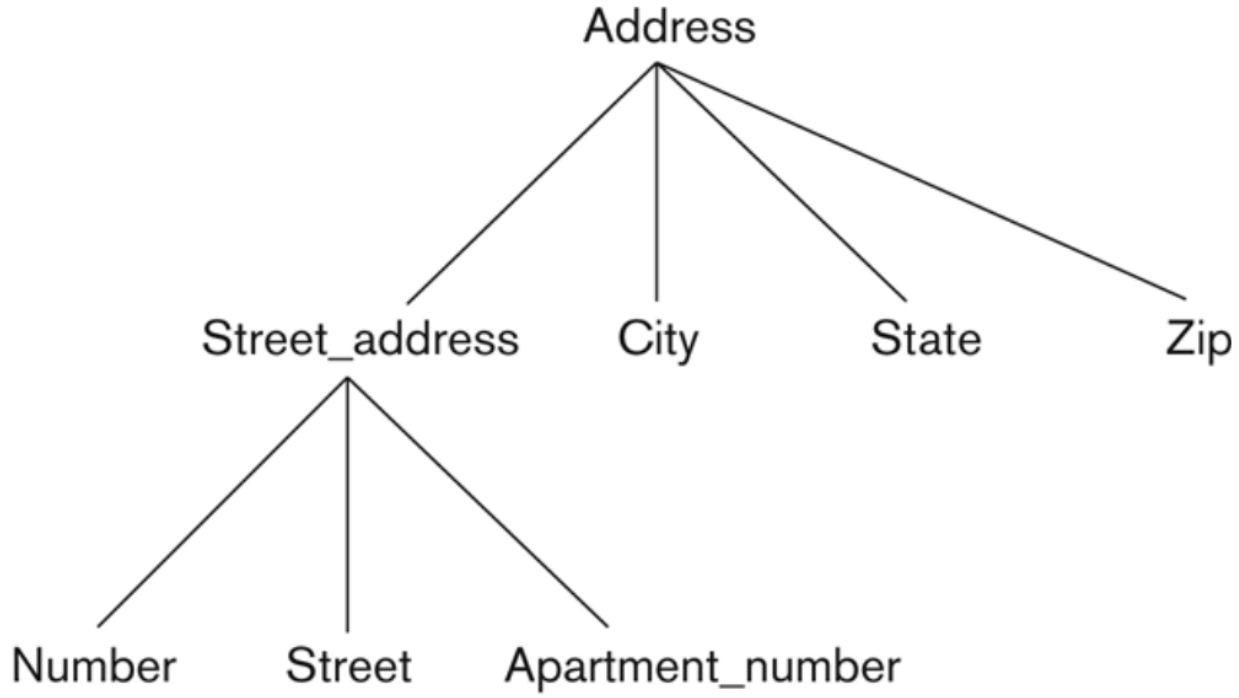
Types of attributes

Simple: each entity has a single atomic value for the attribute. (SSN)

- Atom: cannot be divided into smaller parts.

Composite: composed of different components. In some cases we might have a hierarchy.

- We define it by saying the name of the attribute, then listing in brackets the different divisions.
- Name(FristName, SecondName, LastName)



Entity Types

Single valued: an entity that at some point in time will have a single value.

- Age

Multi-valued: an entity that any given point you might have more than one single value.

- Previous degrees of students (someone could have multiple degrees) we denote this by using curly brackets {Previous Degrees}

At the same time a composite, multi-valued attribute may look like this: {Previous Degrees(College, Degree Name, Grade)}

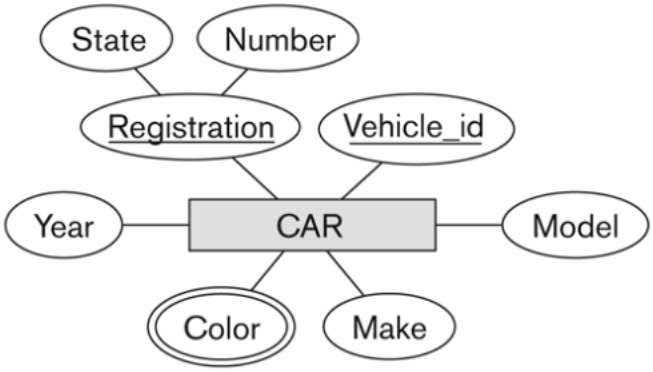
- An employee is an **entity type.**
- John is an **entity.**

Key attributes: those that are unique to every instance or entity. These key attributes are underlined in an ER diagram. (These can also be asas; the combination must be unique).

NOTE there may be more than one key attribute.

ER Diagrams

- **Entity types** are displayed in a rectangular box.
- **Attributes** are displayed in ovals. (Multivalued ones have double ovals)
  - **Key attributes** are underlined.



CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

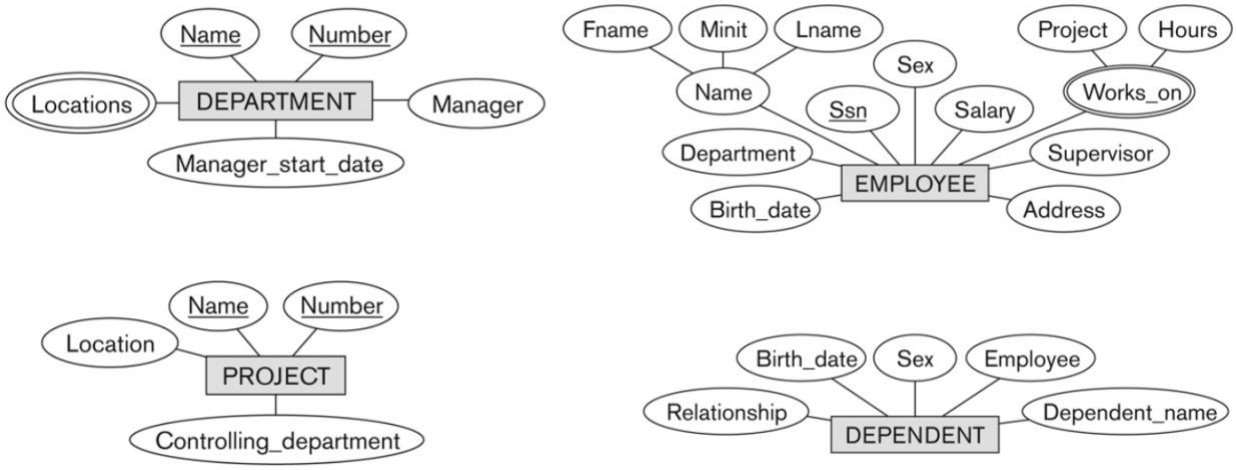
The second part is an entity set.

All entity types will have an entity set which shows all the instances of these types.

COMPANY Example

We have defined the following entity types:

- DEPARTMENT
- PROJECT
- EMPLOYEE
- DEPENDENT



Relationship Types

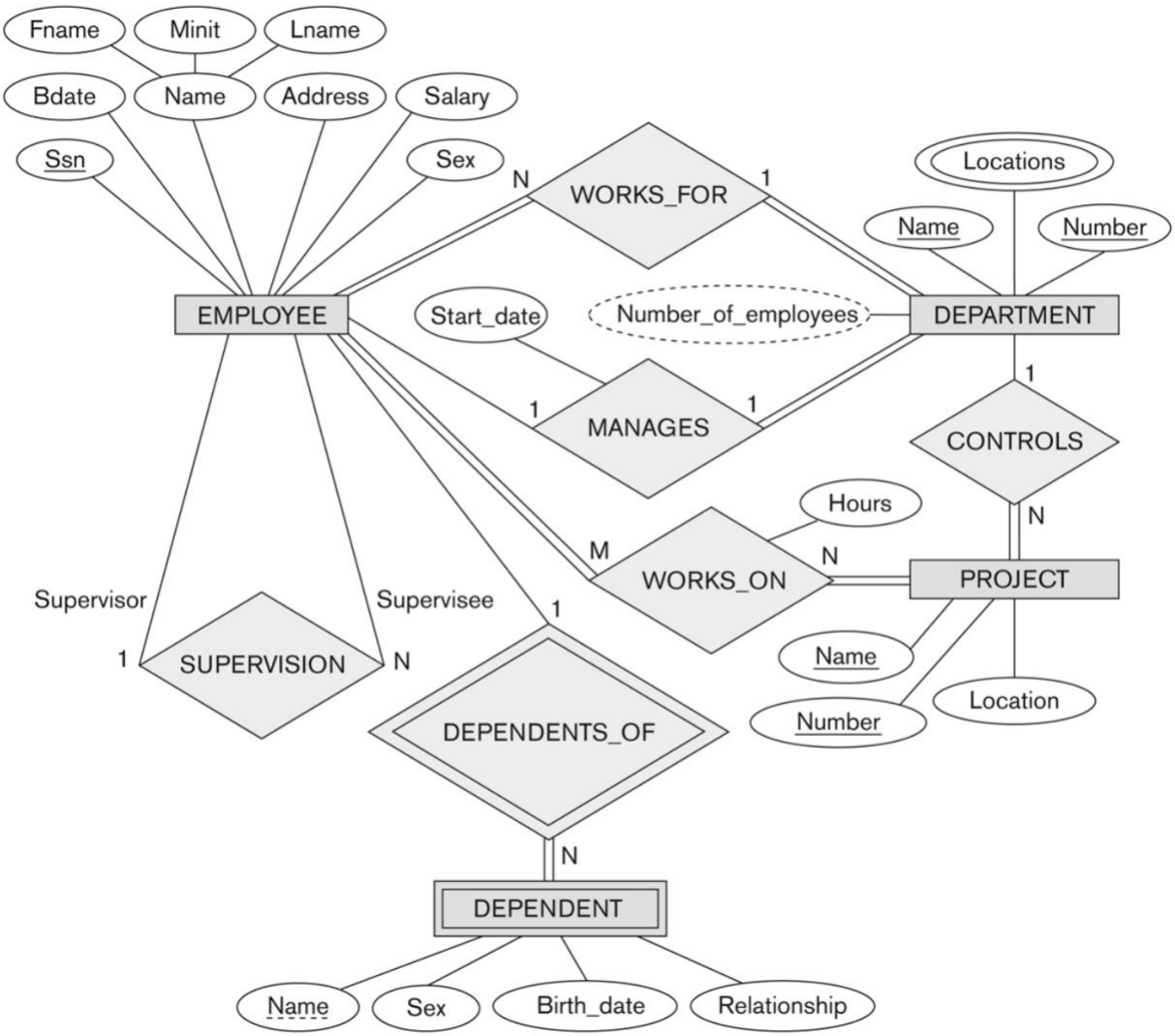
Relationship: relates two or more distinct entities with a specific meaning.

- EMPLOYEE John Johnson works on the ProductX PROJECT.

Relationship type: a specific link.

- WORKS_ON, MANAGES…
- Note that they can also have degrees, in these case the ones above are binary.
    - Relationships with 3 degrees are called ternary. Any other is called n-ary.

We will use **diamonds** for relationships. Numbers around these are the number of relationships for each entity.



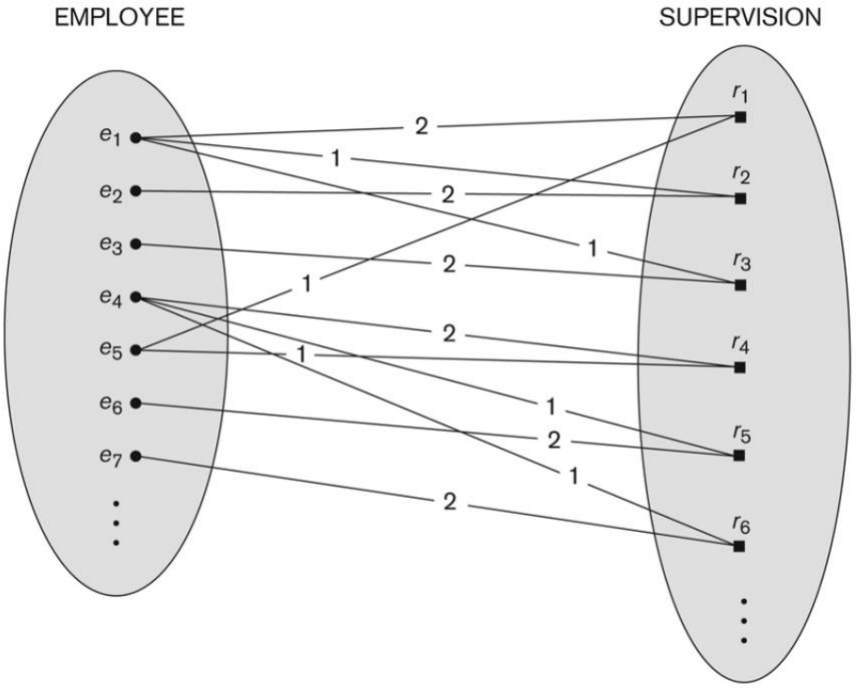
Double lines mean that there should ALWAYS be a relation. For instance, an employee must always work for a department but an employee might not Manage a department.

Weak entity type: entity types which do not have key attributes. The entity is marked with a double rectangle.

Partial key attribute: unlikely to be not key, but could. As seen under name in dependent.

Recursive Relationship Types

If an employee is a supervisor of another one, then that one employee has employee 1 as a supervisee. So, an employee can supervise a supervisor.

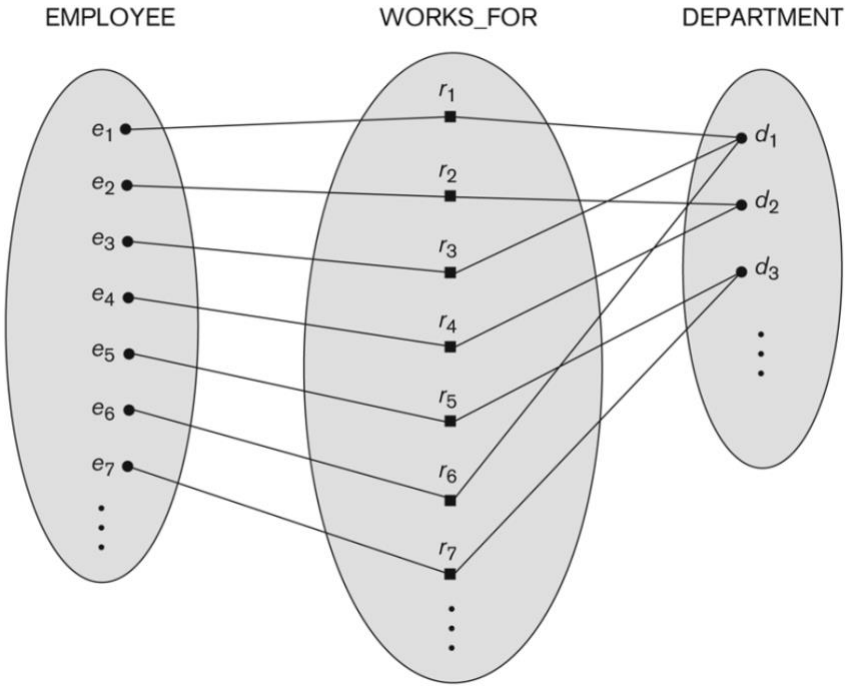


e1 will be the supervisor, e3 will be the supervisee. The first role is labelled with 1 and the second one in labelled with 2.

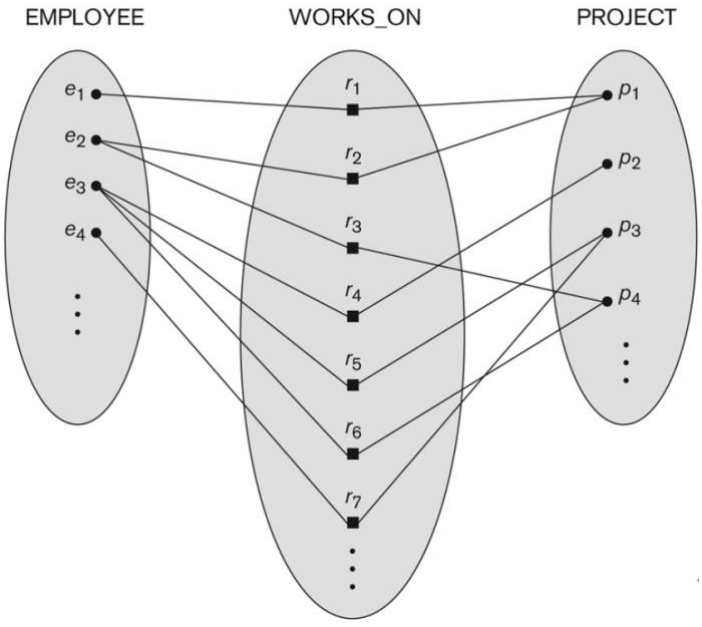
Constraints on Relationships

- **Cardinality Ratio:** specifies the maximum number of relationship instances that an entity can participate in.
- **Existence Dependency Constraint:** specifies minimum participation (if an entity exists or not).

Many-to-one (N:1) or one to many

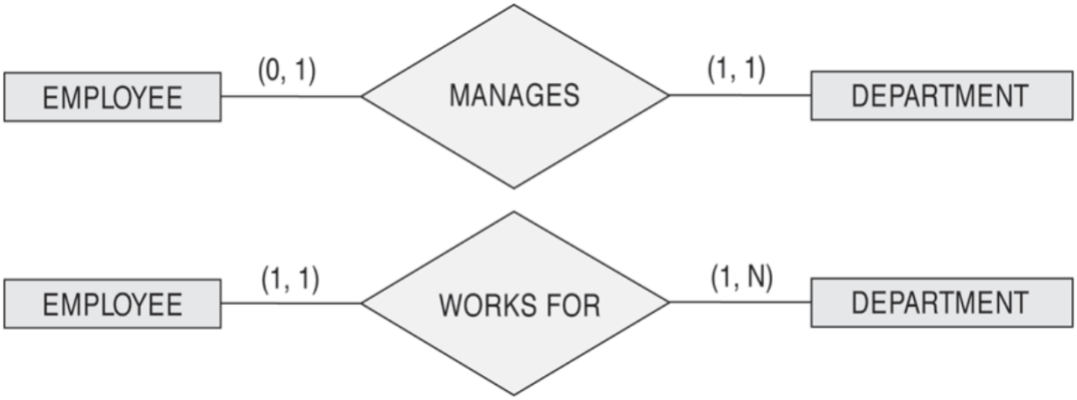


Many-to-many (M: N)

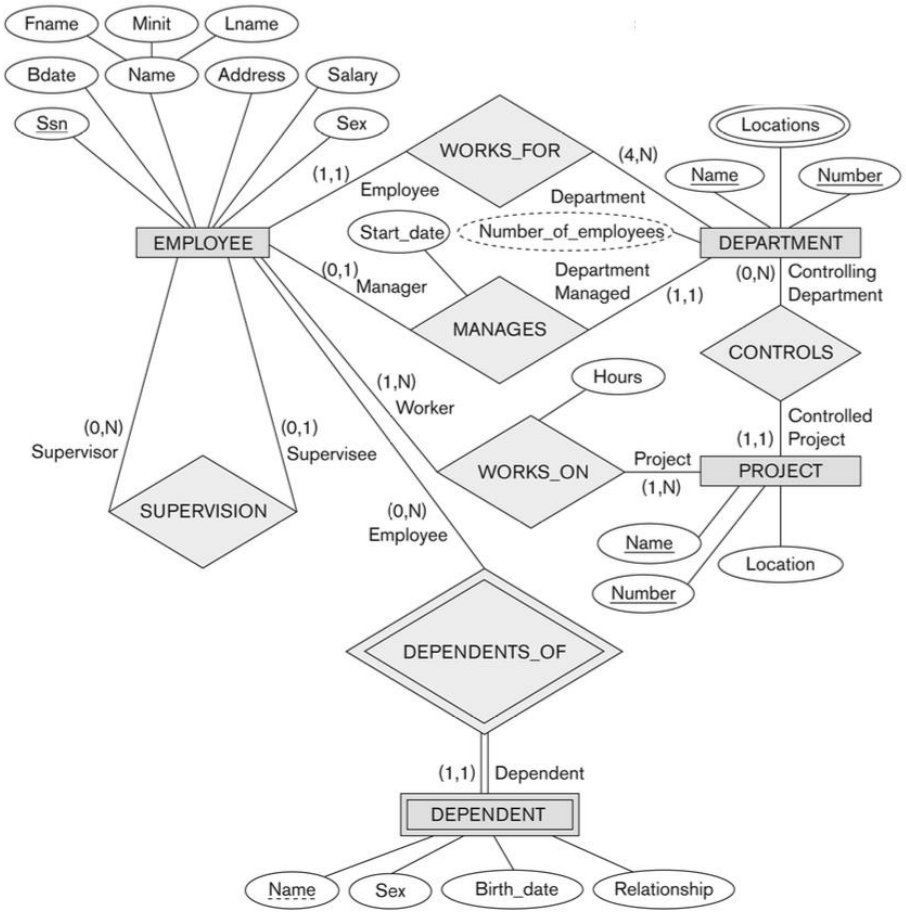


A 1:1 relationship type also exists. We can express cardinalities in two ways. As seem in the diagram or:

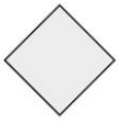
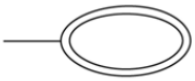
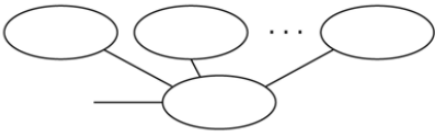
The (min,max) notation.

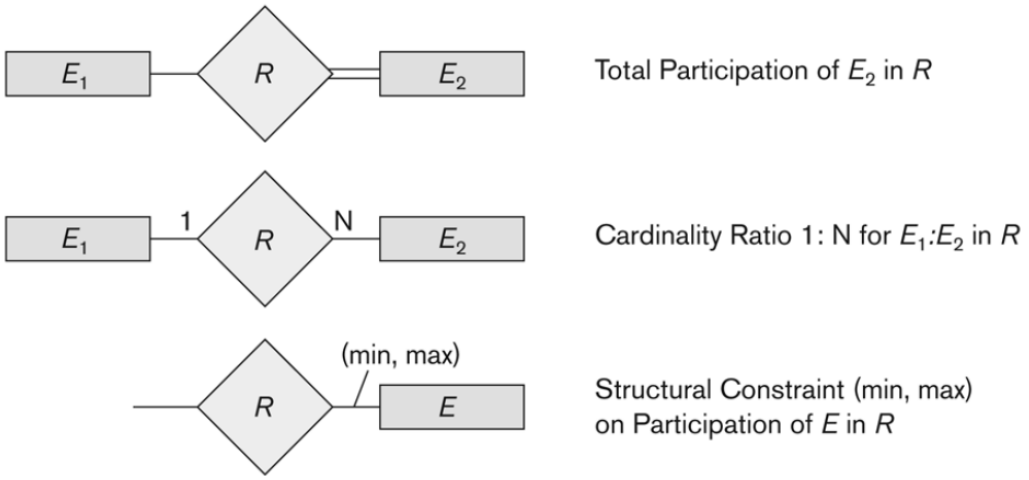


The diagram presented before would be like this:



Summary for ER Diagram Notation

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute



Derived attribute: an attribute that can be calculated from the data in our database but is not strictly included.

Relationships of Higher Degree

Relationship types of degree 2 are called binary, three ternary and above that they are called n-ary.

Showing 3 binary relationships to represent a ternary does not always give the same information about the relationship

Relational Data Model

Relation: mathematical concepts based on the idea of sets.

Informally a relation looks like a table of values. The data in each tow represents facts that correspond to the real-world entity or relationship. We call these rows tuples. The column header gives an indication of the meaning of the data items in that column.

Relation Name: **STUDENT**

Attributes

Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	749-1253	25	3.53
Rohan Panchal	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	NULL	19	3.25

Tuples

Key of a relation: each row has a value of a data item (or set of items) that uniquely identifies that row in the table, in this example it's the SSN.

Tuple

Every row is called a **tuple**: which are an ordered set of values (enclosed in angled brackets <>).

A 4-tuple(has 4 values) of the CUSTOMER relation would look this this:

<"72683,"John Smith","Adress","76786">

A row would be n-tuple in which n is the number of attributes.

Key: each row has an item or set of items that uniquely identifies that row in the table.

Domain

The range of values an attribute can take (format, length, type etc).

State

Relation state: a subset of the Cartesian product of the domains of its attributed.

- Cust-name is defined over a domain of character strings of maximum length 25: dom(Cust-name) is varchar(25)

The **schema of a relation:**

- Denoted by R(A1 to An)
- R is the capitalized name
- The attributes are the As
- E.g. CUSTOMER(Cust-id, Cust-name)
- Each attribute has a domain (a set of values that an attribute can take). The domain also has a data-type.

- Let R(A1, A2) be a relation schema:
  - Let dom(A1) = {0,1}
  - Let dom(A2) = {a,b,c}
- Then: dom(A1) X dom(A2) is all possible combinations:
  {<0,a> , <0,b> , <0,c>, <1,a>, <1,b>, <1,c> }

- The relation state r(R) ⊂ dom(A1) X dom(A2)
- For example: r(R) could be {<0,a> , <0,b> , <1,c> }
  - this is one possible state (or "population" or "extension") r of the relation R, defined over A1 and A2.
  - It has three 2-tuples: <0,a> , <0,b> , <1,c>

Informal Terms		Formal Terms
Table		Relation
Column Header		Attribute
All possible Column Values		Domain
Row		Tuple
Table Definition		Schema of a Relation
Populated Table		State of the Relation

Characteristics of Relations

- r(R) is the notation we use to specify state of the relation.
- In a relation, all values are considered atomic (cannot be divided further).
- Each value in a tuple must in its correct domain,
- Null is used to represent unknown values to tuples.

Notation:

We refer to component values of a tuple t by:

$$t[A_i] \; or \; t.A_i$$

If we say $t[A_u, A_v, \ldots, A_w]$ this refers to a subtuple of t, containing the attributes in the brackets.

Relational Integrity Constraints

Constraints are conditions that must hold on all valid relation states.

- Key constraints
- Entity integrity constraints
- Referential integrity constraints
- (another implicit) is the domain constraint.

Key Constraints

Superkey of R: is a set of attributes SK of R with the following conditions:

- No two tuples in any valid relation state r(R) will have the same value for SK.
- For any two distinct tuples $t1[SK] \neq t2[SK]$

Key of R: a key is a superkey K such that removal of any attribute from K results in a set of attributes that is not a superkey (is not unique).

Example: CAR(State,Reg#,SerialNo,Make,Model,Year)

Has two keys {State,Reg#} and {SerialNo} and they are both superkeys.

But {SerialNo, Make} is a superkey but not a key – because it may be unique in r(R) but it might not ALWAYS be in the future.

Generally, any key is a superkey (but not vice versa)

In a relation from several candidates keys a **primary key** is chosen and underlined. This is the key that uniquely identifies that tuple in the relation. It gives it a tuple identity.

CAR(State,Reg#,SerialNo,Make,Model,Year)

SerialNo is the primary key, generally it is good to associate the primary key to the smallest key in size.

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Relational Database Schema

A set of relation schemas that belong to the same database. S = {R1 to Rn}.

The following example has 6 relation schemata.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Foreign Key

Foreign key: a first key between R1 and R2 is true if the primary key of each relation has the same domain and a value from a tuple from R1 occurs in R2 as well or is null. (A bit like a subset).

Integrity Constraints

Entity integrity: the primary key attributes of each relation schema R in S cannot have null values in any tuple of r(R) (sine we use these to identify tuples)

- t[PK]!= null in r(R)

Referential integrity: a constraint involving two relations. They are used to specify a relationship among tuples in two relations. The referencing and the referenced relation.

The value in the foreign key column which is being referenced must either be:

1. a value of an existing primary key value of a corresponding primary key "PK" in the referenced relation R2
2. a null
   NOTE: if it's a null, then it should not be part of R1 **primary key**.

Semantic Integrity Constraints: other constraints based on semantics that are not expressed in the model per se.

Update operations on Relations

- INSERT a tuple
- DELETE a tuple
- MODIFY a tuple
- Update operations may have to be grouped together
- Integrity constraints should not be violated by the update
- Updates may propagate to cause other updates.

In case of integrity violation:

- Cancel the operation that causes the violation (RESTRICT or REJECT)
- Perform the operation but inform the user of the violation
- Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
- Execute a user-specified error-correction routine

Violations for each operation

INSERT may violate the … constraint:

- Domain: one of the attribute values provided for the tuple is not in the specified domain.
- Key: if the key attribute of a new tuple already exists in another tuple relation
- Referential integrity: if the new tuple references a primary key value that does not exist in the referenced relation
- Entity integrity: if the primary key value is null in the new tuple.

DELETE:

- If the primary key being deleted is reference from other tuples in the database (can be fixed by setting the reference to null, reject deletion or delete referencing tuples this must be specified).

UPDATE:

- Domain constraint or not null constraint.
- Updating primary key (if this is referenced to other tuples)
- Updating foreign key (may violate referential integrity)
- Updating ordinary attribute (may violate domain constraints)

SQL (Structured Query Language)

DDL (Data Definition Language)

Data Definition Language (DDL) Commands: allows you to modify the schema of the database.

- CREATE
- DROP (delete)
- ALTER (update)

Comments

-- line comments are two dashes;

CREATE

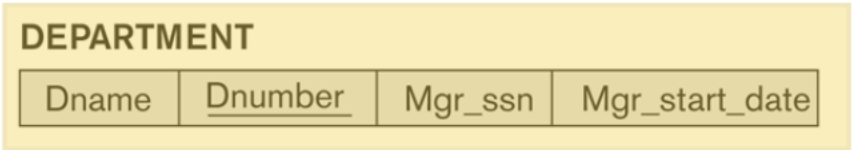
CREATE SCHEMA

CREATE DATABASE COMPANY AUTHORIZATION **JSMITH**;

- We can add authorization for different elements of the schema.

We can have multiple schemas that exist within the database to show certain users, certain things.

The COMPANY Database Schema



```
CREATE TABLE DEPARTMENT (
        DNAME           VARCHAR(10)   NOT NULL,
        DNUMBER         INTEGER             NOT NULL,
        MGRSSN          CHAR(9),
        MGRSTARTDATE    DATE,
);
        ATTRIBUTES      DOMAIN              CONSTRAINTS
```

There are also reserved words which we cannot use to name our relations, therefore we must use other words or ' ' within our word.

Data Types

CORE SQL

Type	Storage (Bytes)	Minimum Value (Signed/Unsigned)	Maximum Value (Signed/Unsigned)
TINYINT	1	-128	127
		0	255
SMALLINT	2	-32768	32767
		0	65535
MEDIUMINT	3	-8388608	8388607
		0	16777215
INT	4	-2147483648	2147483647
		0	4294967295
BIGINT	8	-9223372036854775808	9223372036854775807
		0	18446744073709551615

INT

- INT or INTEGER is interchangeable
- UNSIGNED INTEGER means any non-negative int.
- INT(n) - n denotes the number of digits.
- Size depends on implementations

FLOATS

- FLOAT, REAL, DOUBLE
- DOUBLE does rounding
- FLOAT(n) – n is precision, number of bits used to store the mantissa of the float number in scientific notation.

DECIMAL

- DECIMAL(i,j) allows you to store exact formatted numbers.
  - i – precision, total number of digits to store number.
  - j – scale, after the decimal.
- DECIMAL(7,4) – will look like -999.9999 when displayed.
- More expensive to store but more precise.

STRING

- CHAR(n), CHARACTER(n) – fixed length, right padded with spaces
- VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n) – varying length, n is the upper limit.
- CLOB / TEXT – character large object.

Value	CHAR(4)	Storage Required	VARCHAR(4)	Storage Required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

BITS

- BIT(n) – fixed length
- BIT VARYING(n) – varying length, n is the upper limit.
- BLOB – Binary Large Object. Megabyte, Gigabyte.

We do not normally store files, images rather we store a reference (URL) to the file, given a file system.

PRO TIP: Do hash your binary data and store that in the DB with the metadata. (Use "md5 hash" of your binary data)

BOOLEAN

- BOOLEAN – TRUE or FALSE
- This can be implemented with BIT(1)

NULL

When we implement a Boolean, and NULL comes into place, note that NULL does not mean false. Hence we have to bear this in mind.

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT *p*
True	False
False	True
Unknown	Unknown

DATE and TIME

- DATE – made up of (yyyy-mm-dd)
- TIME – made up of (hh:mm:ss)
- TIME(i) – Time plus i addition of digits for fractions of a second (hh:mm:ss:ii...i)
    - TIME(3) – milliseconds.
- DATETIME / TIMESTAMP – both DATE and Time components.

INTERVAL DATA

- INTERVAL – relative time value as opposed to absolute
    - Can by DAY/TIME intervals of YEAR/MONTH intervals
    - Can be positive or negative when added to or subtracted form an absolute value, the result is an absolute value.
    - E.g.
        - INTERVAL 1 DAY
        - INTERVAL 2 HOUR
        - INTERNAL 6 MONTH

COMPLEX TYPES

- CURRENCY, MONEY
- Spatial types (GIS)
- Geometry types
- Enumerated Types: ENUM("One","Two","Three")
- Collection Types: SET, VALUE_MAP

CREATE DOMAIN

- CREATE DOMAIN – allows you to specify your own data type to use in schema.

```
CREATE DOMAIN SSN AS CHAR(9);

CREATE TABLE DEPARTMENT (
  DNAME          VARCHAR(10) NOT NULL,
  DNUMBER        INTEGER     NOT NULL,
  MGRSSN         SSN,
  MGRSTARTDATE   DATE
);
```

- Need to create the domain before creating the table.

Specifying integrity constraints

- NULL – enforces Entity Integrity Constraint (doesn't allow null for a primary key).

  DNUMBER INTEGER NOT NULL;

- DEFAULT <value>

  MGRSSN CHAR(9) DEFAULT '12345';

- AUTO_INCREMENT – for integer types which adds 1 every time (good for IDs)

  DNUMBER INTEGER NOT NULL AUTO_INCREMENT;

Specifying Domain constraints

- CHECK clause – checks a valid conditional expression

  DNUMBER INT CHECK (DNUMBER > 20 AND DNUMBER < 22);

  Can be used with CREATE DOMAIN (use VALUE to reference attribute)

  CREATE DOMAIN D_NUM AS INTEGER CHECK (VALUE > 0 AND VALUE < 21);

Key & Referential Integrity Constraints

- PRIMARY KEY clause

  DNUMBER INT PRIMARY KEY;

- UNIQUE CLAUSE – for secondary / alternate keys

  DNUMBER CHAR(9) UNIQUE;

- FOREIGN KEY clause – for referential integrity

FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE (SSN);

We create these constraints when we create a table

```
CREATE TABLE DEPARTMENT (
  DNAME          VARCHAR(10) NOT NULL,
  DNUMBER        INTEGER     NOT NULL,
  MGRSSN         CHAR(9),
  MGRSTARTDATE   DATE,
  PRIMARY KEY(DNUMBER),
  UNIQUE(DNAME),
  FOREIGN KEY(MGRSSN) REFERENCES EMPLOYEE(SSN)
);
```

Composite Primary and Foreign keys

```
CREATE TABLE DEPARTMENT (
  DNAME          VARCHAR(10) UNIQUE NOT NULL,
  DNUMBER        INTEGER     NOT NULL,
  MGR_FNAME      CHAR(9),
  MGR_LNAME      CHAR(9),
  MGRSTARTDATE   DATE,
  PRIMARY KEY(DNUMBER, DNAME),
  FOREIGN KEY(MGR_FNAME, MGR_LNAME) REFERENCES
EMPLOYEE(FNAME, LNAME)
);
```

Referential Integrity Options

A referential integrity constraint may be violated when tuples are inserted or deleted.

Default: ***reject*** the operation that violates constraint.

Or: specify a referential triggered action, options:

Event:

- ON DELETE
- ON UPDATE

Action:

- RESTRICT – rejects the delete or update operation (protects parent from deletion not the child).
- CASCADE – does the event on child when cast on parents.
- SET NULL
- SET DEFAULT

Base relations vs Virtual relations

Base tables (or ***base relations*****)**: relations and tuples that store as a file by DBMS, created through the CREATE TABLE statement.

Virtual relations: relations which may not correspond to an actual physical file, created through the CREATE VIEW statement.

Delete

Drop Table

DROP TABLE - used to remove a relation (base table) and its definition.

```
DROP TABLE DEPENDENT;
```

Table dropped if not referenced in any constraints

```
DROP TABLE DEPENDENT RESTRICT;
```

All constraints that reference are table are dropped along with table

```
DROP TABLE DEPENDENT CASCADE;
```

Drop Schema

Dropped only if no elements in the schema:

```
DROP SCHEMA COMPANY RESTRICT;
```

All tables, views and constraints dropped:

```
DROP SCHEMA COMPANY CASCADE;
```

Alter Tables

Add

Used to add an attribute to one of the base relations (these will have NULL for all existing tuples).

```
ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12);
```

In this case database users will have to update every job attribute for all existing employees. That's why we can use a default value.

```
ALTER TABLE EMPLOYEE ADD JOB VARCHAR(12) DEFAULT 'President';
```

Specifying NOT NULL will require a DEFAULT value.

Adding constraints

```
ALTER TABLE EMPLOYEE ADD FOREIGN KEY (DNO) REFERENCES DEPARTMENT (Dnumber);
```

Drop

You can also remove attributes (and its data) [this cannot be done in all RDBS].

```
ALTER TABLE EMPLOYEE DROP JOB;
```

Need to specify the FOREIGN KEY to DROP a constraint.

```
ALTER TABLE DEPARTMENT DROP FOREIGN KEY (MGRSSN);
```

Retrieval Queries in SQL

Bag or **multi-set:** a set that allows duplicate values, order is also irrelevant.

{A, B, C, A} is a bag.

{A, B, C} is also a bag that also is a set.

{A, B, A} = {B, A, A} as bags

- SQL can enforce sets with Key Constraints and DISTINCT.
- Ordered relations (lists) with ORDER BY

SELECT

SELECT <attribute list>
FROM <table list>
WHERE <condition>

- <attribute list> is a list of attribute names whose values are to be retrieved by the query
- <table list> is a list of the relation names required to process the query
- <condition> is a conditional (boolean) expression that identifies the tuples to be retrieved by the query

WHERE <condition>

Basic Logical Operators in the <condition>

- = (equals)
- <, <=
- >, >=
- <>, != (not equals)
- AND
- OR
- IS NULL, IS NOT NULL

If no WHERE clause is selected, then all tuples are returned. If more than one relation is chosen (in the FROM clause) then the Cartesian product of the selected tuples is returned.

```
SELECT   SSN, DNAME
FROM     EMPLOYEE, DEPARTMENT
```

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

Result

Ssn	Dname
123456789	Research
333445555	Research
999887777	Research
987654321	Research
666884444	Research
453453453	Research
987987987	Research
888665555	Research
123456789	Administration
333445555	Administration
999887777	Administration
987654321	Administration
666884444	Administration
453453453	Administration
987987987	Administration
888665555	Administration
123456789	Headquarters
333445555	Headquarters
999887777	Headquarters
987654321	Headquarters
666884444	Headquarters
453453453	Headquarters
987987987	Headquarters
888665555	Headquarters

Be careful: Easy to get LARGE relations as a result!

NOTE THAT * means all the attributes in a relation.

Use of DISTINCT

To eliminate duplicate tuples in a query result, the keyword DISTINCT is used.

```
SELECT DISTINCT SEX, SALARY
FROM   EMPLOYEE
```

Result: one duplicate row is removed

SEX	SALARY
M	30000
M	40000
F	25000
F	43000
M	38000
~~F~~	~~25000~~
M	25000
M	55000

Qualification of Relation Names

In SQL, we can use the same name for two or more attributes as long as they are in different relations.

```
SELECT   EMPLOYEE.FNAME, EMPLOYEE.LNAME,
         EMPLOYEE.ADDRESS
FROM     EMPLOYEE, DEPARTMENT
WHERE    DEPARTMENT.DNAME='Research' AND
         DEPARTMENT.DNUMBER=EMPLOYEE.DNO
```

Aliases

We can abbreviate names of relations in SQL.

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E S
WHERE E.SUPERSSN=S.SSN
```

Set Operations

SQL has some set operations:

- Union
- Minus/Except
- Intersection

Resulting relations of these set operations are sets of tuples (duplicates are eliminated).

Set operations only apply to union compatible relations:

- They must have the same attributes (names).
- Attributes must appear in the same order.

Union

Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

```
(SELECT   PNUMBER
 FROM     PROJECT, DEPARTMENT, EMPLOYEE
 WHERE    DNUM=DNUMBER AND MGRSSN=SSN
          AND LNAME='Smith')
UNION
(SELECT   PNUMBER
 FROM     PROJECT, WORKS_ON, EMPLOYEE
 WHERE    PNUMBER=PNO AND
          ESSN=SSN AND LNAME='Smith')
```

Substring Comparison / Pattern Matching

- LIKE – comparison operator is used to compare partial strings.
- '%' - (or '*' in some implementations) replaces an arbitrary number of characters
- '_' – replaces a single arbitrary character.
- Usually use an escape-character '\' to specify these reserved characters in your search string.

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

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

Retrieve all employees who were born during the 1950s.

```
SELECT FNAME, LNAME
FROM   EMPLOYEE
WHERE  BDATE LIKE '__5_______'
```

The LIKE operator considers each attribute as atomic and indivisible (in SQL, character string attribute values are not atomic).

Arithmetic Operations

The standard operations (add, subtract, multiply and divide) can be applied to numeric values in an SQL query result.

Order By

ORDER BY clause – used to sort tuples in a query result based on the values of some attribute(s). The *default* order is in ascending although if we wish it in descending order we can add the keyword DESC (ASC can be used for ascending order although it is by default).

```
SELECT    DNAME, LNAME, FNAME, PNAME
FROM      DEPARTMENT, EMPLOYEE,
          WORKS_ON, PROJECT
WHERE     DNUMBER=DNO AND SSN=ESSN
          AND PNO=PNUMBER
ORDER BY  DNAME DESC, LNAME ASC
```

Summary of SELECT Queries

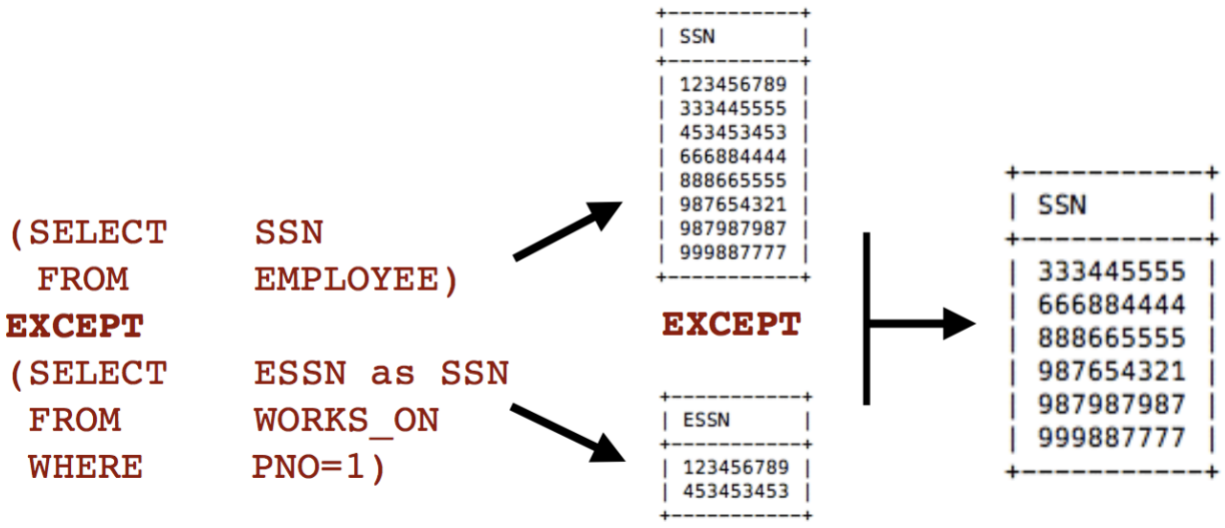
Only the first two are mandatory.

```
SELECT     <attribute list>
FROM       <table list>
[WHERE     <condition>]
[ORDER BY  <attribute list>]
```

Set Operations

EXCEPT clause – works as the minus from a set.

Example: List SSNs from all employees except those who are working on Project 1.



If function

IF(<condition>,<True Value>,<False Value>)

Using if with more than two values you can use CASE() or you can use nested if statements.

```
SELECT   FNAME, LNAME,
         (SALARY / 1000) AS SALARY_K,
         1 AS ONE,
         IF(SALARY > 30000, True, False) AS IS_LOADED,
         IF(SUPERSSN IS NULL, "Boss", "Worker") AS ETYPE
FROM   EMPLOYEE
```

fname	lname	SALARY_K	IS_LOADED	ETYPE
John	Smith	30.0000	0	Worker
Franklin	Wong	40.0000	1	Worker
Joyce	English	25.0000	0	Worker
Ramesh	Narayan	38.0000	1	Worker
James	Borg	55.0000	1	Boss
Jennifer	Wallace	43.0000	1	Worker
Ahmad	Jabber	25.0000	0	Worker
Alicia	Zelaya	25.0000	0	Worker

(Note the use of AS to alias the results as an attribute)

Cast function
Used to convert from a Data Type to another.

CAST(<expression> AS <type>)

```
SELECT FNAME, LNAME,
       CAST((SALARY / 1000) AS UNSIGNED)
         AS SALARY_K
FROM   EMPLOYEE
```

```
+----------+---------+----------+
| fname    | lname   | SALARY_K |
+----------+---------+----------+
| John     | Smith   |       30 |
| Franklin | Wong    |       40 |
| Joyce    | English |       25 |
| Ramesh   | Narayan |       38 |
| James    | Borg    |       55 |
| Jennifer | Wallace |       43 |
| Ahmad    | Jabber  |       25 |
| Alicia   | Zelaya  |       25 |
+----------+---------+----------+
```

Data Manipulation Language

- INSERT
- DELETE
- UPDATE

These don't modify the schema (DML does include SELECT).

Insert

Used to add one or more tuples in the relation.

INSERT INTO <table name>
VALUES <tuple>

```
INSERT INTO  EMPLOYEE
VALUES ('Richard','K','Marini','653298653',
        '30-DEC-52','98 Oak Forest,Katy,TX',
        'M', 37000,'987654321', 4)
```

The values should be inserted in the order the attributes are specified in the CREATE TABLE command.

Insert Specifying Values

INSERT INTO <table name> (<attribute list>)

VALUES <tuple>

Any left-out attributes will be default value or NULL.

```
INSERT INTO EMPLOYEE (FNAME, LNAME, SSN)
VALUES ('Richard', 'Marini', '653298653')
```

Combining Insert with Select

We can also use the SELECT command to choose what we want to INSERT INTO.

Example: We want to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project.

```
INSERT INTO WORKS_ON_INFO(EMP_NAME, PROJ_NAME, HOURS_PER_WEEK)
SELECT      E.Lname, P.Pname, W.Hours
FROM        PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE       P.Pnumber=W.Pno AND W.Essn = E.Ssn;
```

Create Table … As

It is possible create a table inserting certain values (last two queries in one) at the same time.

```
CREATE TABLE  WORKS_ON_INFO AS
SELECT        E.Lname AS Emp_Name,
              P.Pname AS Proj_Name,
              W.Hours AS Hours_per_week
FROM          PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE         P.Pnumber=W.Pno AND W.Essn = E.Ssn;
```

*Note the use of the keyword **AS** to specify table/attribute names*

AS is used in two different ways in this query.

Delete

Removes tuples from a relation.

DELETE FROM <table name>
WHERE <condition>

- Tuples are deleted from one table at a time unless CASCADE is specified on a referential integrity constraint.
- If there is no where clause, then all tuples in the relation are deleted and the table becomes empty.

Delete

Used to modify attribute values for a selected tuples.

UPDATE <table name>
SET <attribute>=<value>, …
WHERE <condition>

Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
UPDATE  PROJECT
SET     PLOCATION = 'Bellaire',
        DNUM = 5
WHERE   PNUMBER=10;
```

If where Is not there, then it is always considered as WHERE true.

Example 2: Give all employees in the 'Research' department a 10% raise in salary.

```
UPDATE  EMPLOYEE
SET     SALARY = SALARY*1.1
WHERE   DNO IN (SELECT  DNUMBER
                FROM DEPARTMENT
                WHERE DNAME='Research');
```

Nesting of Queries

Select queries can have an inner query inside.

Using comparison operator **IN –** compares a value v with a set of values V and returns TRUE if v is one of the elements in V.

Example: Retrieve the name and address of all employees who work for the 'Research' department.

Way 1:

```
SELECT  FNAME, LNAME, ADDRESS
FROM    EMPLOYEE, DEPARTMENT
WHERE   DNAME='Research' AND DNUMBER=DNO
```

Way 2 (using IN):

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

Exists function

Exists (or not exists) checks whether the result of a correlated nested query (query which in the SELECT clause there is an attribute which appears in the WHERE clause in the nested query) is empty or not.

Example: Retrieve the name of each employee who has a dependent with the same first name as the employee.

Way 1:

```
SELECT FNAME, LNAME
FROM   EMPLOYEE E
WHERE  EXISTS (SELECT  *
               FROM    DEPENDENT
               WHERE   E.SSN=ESSN AND
                       E.FNAME=DEPENDENT_NAME)
```

Way 2:

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

All comparison operator

Compares a singles values (as an attribute) to a set or multiset (nested query)

Example: Retrieve the names of employees whose salary is greater than the salary of all employees in department 5.

```
SELECT     LNAME, FNAME
FROM       EMPLOYEE
WHERE      SALARY > ALL(SELECT  SALARY
                        FROM    EMPLOYEE
                        WHERE   DNO=5)
```

NULLs in SQL Queries

We can use IS NULL or IS NOT NULL to check whether something is null or not.

Example: Retrieve the names of all employees who do not have supervisors

```
SELECT   FNAME, LNAME
FROM     EMPLOYEE
WHERE    SUPERSSN IS NULL
```

Joined Relations

- JOIN (theta join)
- NATURAL JOIN
- LEFT OUTER JOIN, LEFT JOIN
- RIGHT OUTER JOIN, RIGHT JOIN
- FULL OUTER JOIN, OUTER JOIN
- INNER JOIN
- CROSS JOIN

JOIN … ON

SELECT … with a JOIN Condition in the WHERE clause:

```
SELECT FNAME, LNAME, ADDRESS
FROM   EMPLOYEE, DEPARTMENT
WHERE  DNAME='Research' AND DNUMBER=DNO
```

Using JOIN … ON as an "*equi-join*"

```
SELECT FNAME, LNAME, ADDRESS
FROM   (EMPLOYEE JOIN DEPARTMENT
        ON DNUMBER=DNO)
WHERE  DNAME='Research'
```

NATURAL JOIN (Note the use of Aliasing with AS)

```
SELECT   FNAME, LNAME, ADDRESS
FROM     (EMPLOYEE NATURAL JOIN DEPARTMENT
          AS DEPT(DNAME, DNO, MSSN, MSDATE)
WHERE    DNAME='Research'
```

Differences in JOIN Functions

- **NATURAL JOIN:** no join condition may be specified, joins attributes with the same name.
- **INNER JOIN:** tuple is included in the result only if a matching tuple exists in the other relation (default type of JOIN)
- **OUTER JOIN:** all matching tuples are returned depending on the type of outer join.
    - **LEFT OUTER JOIN**
    - **RIGHT OUTER JOIN**
    - **FULL OUTER JOIN**

CROSS JOIN

Returns the Cartesian product of the selected attributes.

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM   (EMPLOYEE E CROSS JOIN EMPLOYEE S)
```

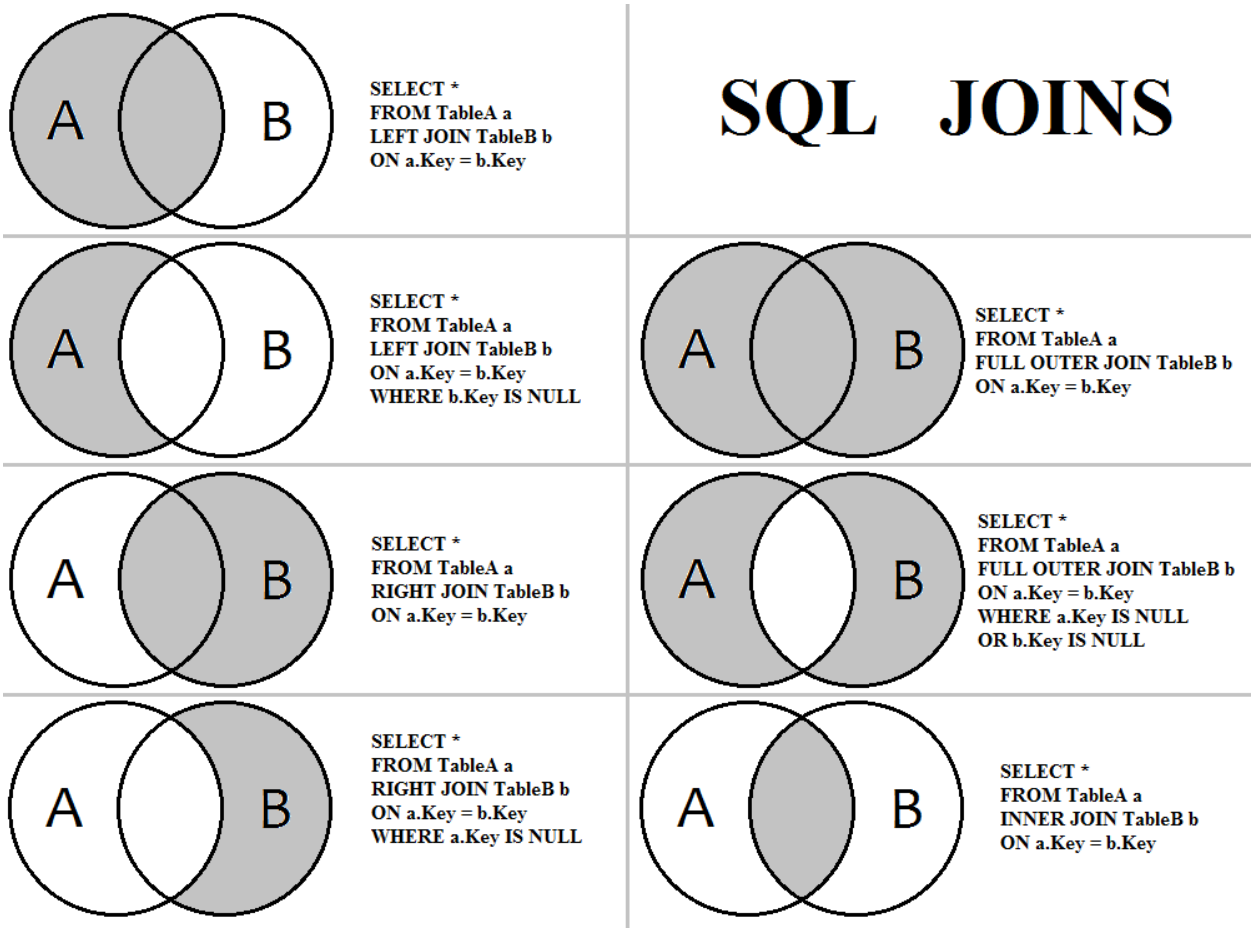
But this can also be done in a different way:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM   EMPLOYEE E S
```

And

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM   EMPLOYEE E JOIN EMPLOYEE S
```

Which shows that Cross Join is implicit in the JOIN.



Aggregate Functions
Functions like:

COUNT, SUM, MAX, MIN and AVG can be used to retrieve data from the database.

- **COUNT(attribute)** – counts the number of values (or rows). Null values are not counted.
- **SUM(attribute)** – adds all the values in the row
- **MAX(attribute)** – returns maximum value
- **MIN(attribute)** – returns minimum value

Example: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
SELECT   MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM     EMPLOYEE, DEPARTMENT
WHERE    DNO=DNUMBER AND DNAME='Research'
```

Grouping

If we want to apply aggregate functions to subgroups of tuples in a relation, we can use grouping.

GROUP-BY

```
SELECT      <attribute list, include grouping attributes>
FROM        <table list>
[WHERE      <condition>]
GROUP BY <grouping attributes>
```

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

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

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

You must include the grouping clause in the select clause.

Having clause

The having clause is used for specifying a selection condition on groups.

```
SELECT      <attribute list, include grouping attributes>
FROM        <table list>
[WHERE      <condition>]
GROUP BY    <grouping attributes>
HAVING      <condition>
```

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

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

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

Summary of SQL Queries

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

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

A query is evaluated by:

1. applying the **WHERE**-clause
2. then **GROUP BY** and HAVING
3. and finally the **SELECT**-clause
4. and then **ORDER BY** the resulting tuples

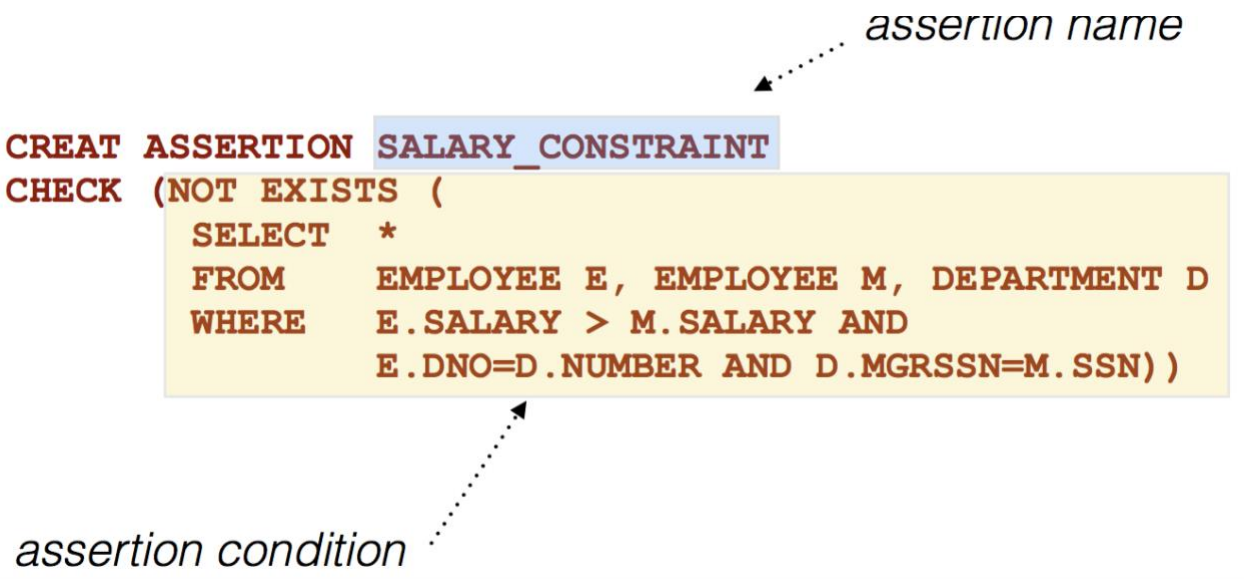
Constraints as Assertions

General constraints: constraints that do not fit in the basic SQL categories. We can create **schema assertions** which are outside of the scope of the built-in relational model constraints (primary, unique keys, entity integrity, referential integrity). It defines whether the state of the database is valid at any given point of time.

CREATE ASSERTION clause has a:

- Constraint name
- Check keyword
- Condition clause

Example: The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.



Views in SQL

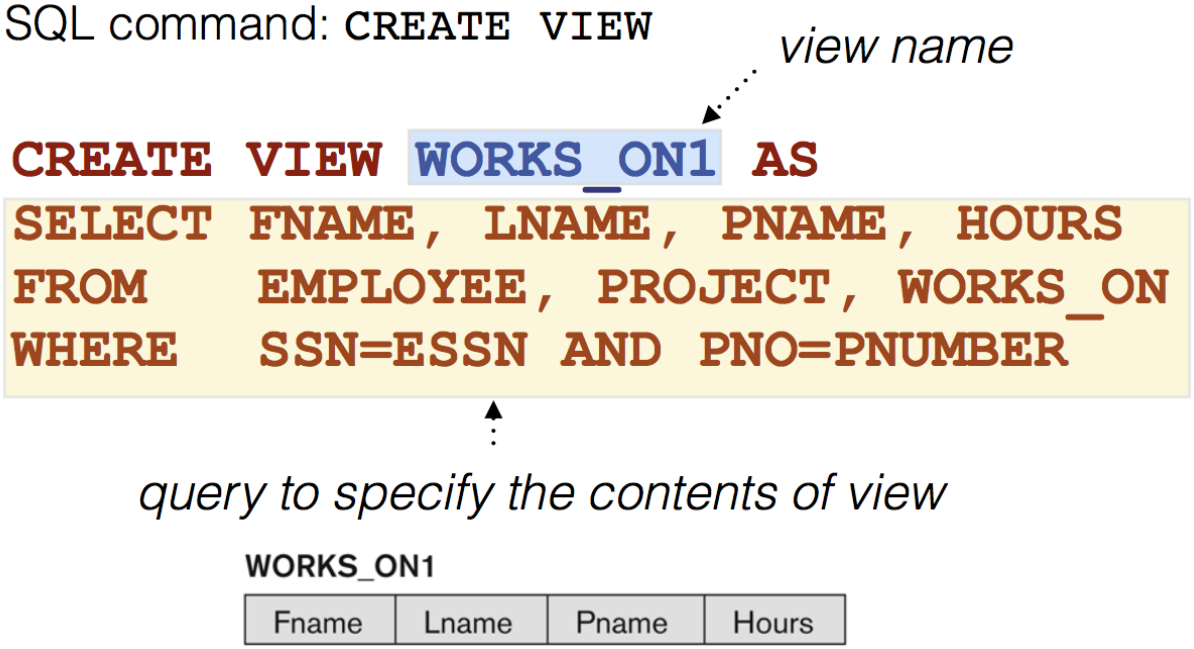
A ***view*** is a virtual table that is derived from other tables. Used for:

- Query modification (copy and paste queries)
- View materialization – short term physical implementation

The table is not physically stored but allows full query operations. Used for security and authorization and prevents redundant storage of data.

Example: A "friendlier" view of WORKS_ON

- SQL command: **CREATE VIEW**



WORKS_ON1

Fname	Lname	Pname	Hours

Now we can use multiple SELECTE queries. When we are done with our view, we can drop it.

DROP WORKS_ON1;

Dropping a view does NOT modify the data.

Relational Algebra

Algebra operations produce new relations, by creating a **relational algebra expression**.

Operations

- Relational Algebra consists of several groups of operations
  - Unary Relational Operations
    - SELECT (symbol: σ (sigma))
    - PROJECT (symbol: π (pi))
    - RENAME (symbol: ρ (rho))
  - Relational Algebra Operations From Set Theory
    - UNION (∪), INTERSECTION (∩), DIFFERENCE (or MINUS, –)
    - CARTESIAN PRODUCT (**x**)
  - Binary Relational Operations
    - JOIN (several variations of JOIN exist)
    - DIVISION
  - Additional Relational Operations
    - OUTER JOINS, OUTER UNION
    - AGGREGATE FUNCTIONS (These compute summary of information: for example, SUM, COUNT, AVG, MIN, MAX)

Unary Relational Operations

SELECT

Sigma σ is a Select operation: takes tuples from a relation and it selects a subset of the tuples from the relation based on a selection condition.

- The conditions acts like a filter.

- Examples:
  - Select the EMPLOYEE tuples whose department number is 4:

$$\sigma_{DNO = 4} (EMPLOYEE)$$

  - Select the employee tuples whose salary is greater than $30,000:

$$\sigma_{SALARY > 30,000} (EMPLOYEE)$$

- What are the resulting relations?

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

General operation

$$\sigma_{<condition>}(R)$$

We select those which evaluate the condition to true.

- SELECT σ is commutative:
  - $\sigma_{<condition1>}(\sigma_{<condition2>}(R)) = \sigma_{<condition2>}(\sigma_{<condition1>}(R))$
- Because of commutativity property, a cascade (sequence) of SELECT operations may be applied in any order:
  - $\sigma_{<cond1>}(\sigma_{<cond2>}(\sigma_{<cond3>}(R)) = \sigma_{<cond2>}(\sigma_{<cond3>}(\sigma_{<cond1>}(R))))$
- A cascade of SELECT operations may be replaced by a single selection with a conjunction of all the conditions:
  - $\sigma_{<cond1>}(\sigma_{<cond2>}(\sigma_{<cond3>}(R)) = \sigma_{<cond1> \text{ AND } <cond2> \text{ AND } <cond3>}(R))$
- The number of tuples in the result of a SELECT is less than (or equal to) the number of tuples in the input relation R

Typo in brackets.

PROJECT

Project π (pi) projects the attributes into a new relation that has all the tuples for each attribute.

- Example: To list each employee's first and last name and salary, the following is used:

$$\pi_{LNAME, FNAME, SALARY}(EMPLOYEE)$$

- What is the resulting relation?

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

In this case all tuples of LName, FName and Salary will be projected.

General

$$\pi_{<attribute\ list>}(R)$$

This operation removes all duplicate tuples, this is because a project operation returns a **set of tuples**.

$\pi_{Lname, Fname, Salary}$ (EMPLOYEE)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

$\pi_{Sex, Salary}$ (EMPLOYEE)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Note that multiple Relational Algebra Expressions can be executed either by nesting operations or by giving names to intermediate results.

- We can write a *single relational algebra expression* as follows:
  - $\pi_{FNAME, LNAME, SALARY}(\sigma_{DNO=5}(EMPLOYEE))$
  - ```
    SELECT FNAME, LNAME, SALARY
    FROM EMPLOYEE
    WHERE DNO=5
    ```

Or:
    ```

- We can explicitly show the *sequence of operations*, giving a name to each intermediate relation:
  - DEP5_EMPS ← $\sigma_{DNO=5}$(EMPLOYEE)
  - RESULT ← $\pi_{FNAME, LNAME, SALARY}$ (DEP5_EMPS)

Rename

RENAME ρ (rho) allows you to rename the relation.

- ρ_S(R) changes:
  - the *relation name* to S
- $\rho_{(B1, B2, \ldots, Bn)}$(R) changes:
  - the *column (attribute) names* to B1, B2, …..Bn
- $\rho_{S\ (B1, B2, \ldots, Bn)}$(R) changes both:
  - the relation name to S, *and*
  - the column (attribute) names to B1, B2, …..Bn

We can also rename attributes by using an arrow.

- Using: S ← R

$$TEMP \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$
$$R(First\_name, Last\_name, Salary) \leftarrow \pi_{FNAME, LNAME, SALARY}(TEMP)$$

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Or we can also use:

- Using $\rho_{S(B1, B2, \ldots, Bn)}(R)$

$$TEMP \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$
$$\rho_{R(First\_name, Last\_name, Salary)}(\pi_{FNAME, LNAME, SALARY}(TEMP))$$

Relational Algebra Operations from Set Theory

Union

- Denoted by ∪.
- Duplicate tuples are eliminated
- The result of RUS, is all the that are in R, S or in both.
- The two operand relations must be compatible (same number of attributes, same domain)

Example: To retrieve the social security numbers of all employees who either work in department 5 (RESULT1 below) or directly supervise an employee who works in department 5 (RESULT2 below)

$$DEP5\_EMPS \leftarrow \sigma_{DNO=5}(EMPLOYEE)$$

$$RESULT1 \leftarrow \pi_{SSN}(DEP5\_EMPS)$$

$$RESULT2(SSN) \leftarrow \pi_{SUPERSSN}(DEP5\_EMPS)$$

$$RESULT \leftarrow RESULT1 \cup RESULT2$$

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Intersection

- The result of $R \cap S$, is a relation that includes all tuples that are both in R and S.
  - The attributes names will be the same as the name in the attribute for R.
- Two operand relations must be type compatible.

Set Difference (Minus or Except)

- The result of R – S is a relation that includes all tuples that are in R but not in S.
  - Names of attributes will be as in R

- The two operand relations must be type compatible.

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT – INSTRUCTOR

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR – STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Notice that both union and intersection are *commutative* operations; that is

- $R \cup S = S \cup R$, and $R \cap S = S \cap R$

Both union and intersection can be treated as **n-ary operations** applicable to any number of relations as both are *associative* operations; that is

- $R \cup (S \cup T) = (R \cup S) \cup T$
- can be written as: $R \cup S \cup T$
- $(R \cap S) \cap T = R \cap (S \cap T)$
- can be written as: $R \cap S \cap T$

The minus operation is not commutative; that is, in general

- $R - S \neq S - R$

CARTESIAN (CROSS) PRODUCT

- Used to combine tuples from two relation in combinatorial fashion
- Denoted by R(A1,A2,…,An) x S(B1,B2,…,Bm)
- The result is tuples which are combinations of all individual elements.
- R x S will have a as many tuples as the number of tuples in R multiplied by the number of tuples in S.

- The two operands **do not** require to be type compatible.

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

EMPNAMES x DEPENDENT

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

- Cross product is not a meaningful operation (normally useful when used by other operations).

Binary Relational Operations

JOIN

denoted by ⋈

- A join is a CARTESIAN PRODUCT followed by the SELECT in one motion
- Binary, applied to two relations

$$R \bowtie_{\text{<join condition>}} S$$

Where R and S can be relations that result from general relational algebra expressions.

- Consider the following JOIN operation:
  - $R(A1, A2, \ldots, An) \bowtie_{R.Ai=S.Bj} S(B1, B2, \ldots, Bm)$
  - Result is a relation Q with degree **n + m attributes**:
    - Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.

Example: Suppose that we want to retrieve the name of the manager of each department.

- To get the manager's name, we need to combine each DEPARTMENT tuple with the EMPLOYEE tuple whose SSN value matches the MGRSSN value in the department tuple.
- We do this by using the join ⋈ operation.

- DEPT_MGR ← DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

MGRSSN=SSN is the join condition

- Combines each department record with the employee who manages the department
- The join condition can also be specified as DEPARTMENT.MGRSSN= EMPLOYEE.SSN

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_MGR ← DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

Boolean expressions

AND

σ Relationship="Spouse" AND Sex="M" (DEPENDENT)

σ Relationship="Spouse" $\wedge$ Sex="M" (DEPENDENT)

Logical AND: $\wedge$

OR

σ Relationship="Spouse" OR Sex="M" (DEPENDENT)

σ Relationship="Spouse" $\vee$ Sex="M" (DEPENDENT)

Logical OR: $\vee$

Theta Join

General Form

$$R \bowtie_{theta} S$$

The theta can be any join condition. A sub-theta join is a n equijoin.

Equijoin

- Only involves join conditions with equality (equals symbol) comparisons only.

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{MGRSSN=SSN} EMPLOYEE$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

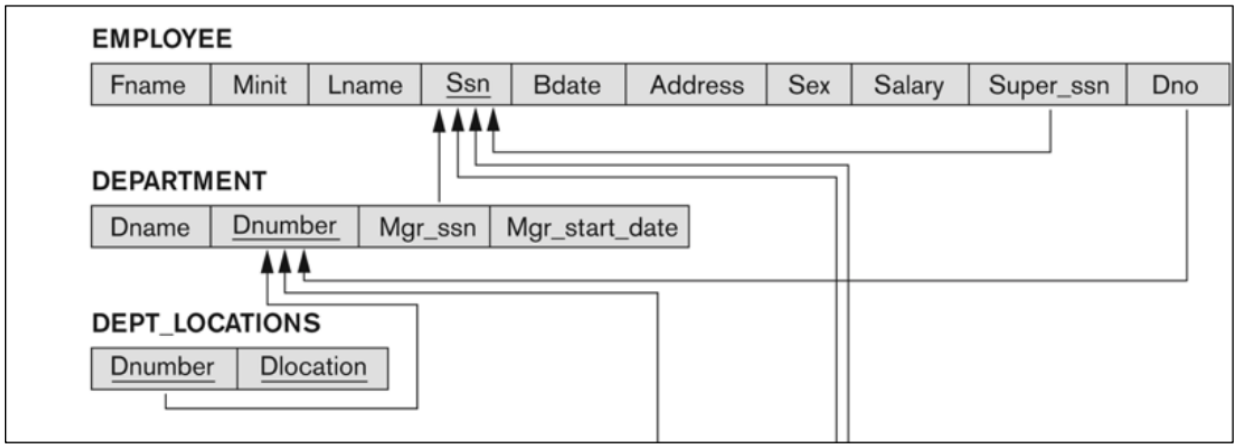
Natural Join

Is a variation of the JOIN which is created to get rid of the second copy of the same attribute (just as you would naturally, you wouldn't keep the same information twice). It is *necessary* that the attribute which is being joined has the **same name** in both relations.

$$R * S$$

Example: To apply a natural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write:

$$DEPT\_LOCS \leftarrow DEPARTMENT * DEPT\_LOCATIONS$$



- In a general example:
  Q ← R(A,B,C,D) * S(C,D,E)
  - The implicit join condition includes *each pair* of attributes with the same name, joined with AND :
    - R.C=S.C AND R.D=S.D
  - Result keeps only one attribute of each such pair:
    - Q(A,B,C,D,E)

Division

- The division operation is applied to **two** relations
- R(Z) ÷ S(X) = T(Y)
  - where X subset of Z.
  - Let Y = Z - X (and hence Z = X ∪ Y); that is, let Y be the set of attributes of R that are not attributes of S.
- For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S.
- DIVISION is like having a *dynamic* SELECT operation

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

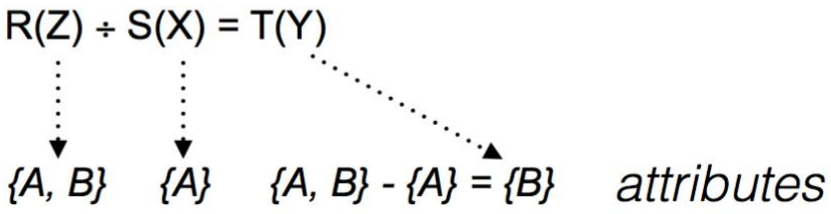
S

A
a1
a2
a3

T

B
b1
b4

- For a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with *every* tuple in S.



Example: Find the names of employees who work on all the projects that 'John Smith' works on.

First retrieve the list of project numbers that 'John Smith' works on:

- SMITH ← $\sigma_{\text{FNAME='John' AND LNAME='Smith'}}$(EMPLOYEE)
- SMITH_PNOS← π_{PNO} (WORKS_ON $\bowtie_{\text{ESSN=SSN}}$ SMITH)

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

SMITH

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5

SMITH_PNOS

Pno
1
2

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

Next create a relation that includes attributes <PNO, ESSN> (SSN and PNOs for everybody):

- SSN_PNOS← $\pi_{ESSN, PNO}$ (WORKS_ON)

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

Finally apply the division operation to the two relations, which will produce the desired outcome:

- SSNS(SSN)←SSN_PNOS ÷ SMITH_PNOS

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

JOIN and PROJECT the resulting SSNS with the Employees to get the result:

- RESULT ← $\pi_{FNAME, LNAME}$ (SSNS*EMPLOYEE)

SSNS

Ssn
123456789
453453453

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

RESULT

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

$SMITH \leftarrow \sigma_{FNAME='John' AND LNAME='Smith'}(EMPLOYEE)$

$SMITH\_PNOS \leftarrow \pi_{PNO}(WORKS\_ON \bowtie_{ESSN=SSN} SMITH)$

$SSN\_PNOS \leftarrow \pi_{ESSN, PNO}(WORKS\_ON)$

$SSNS(SSN) \leftarrow SSN\_PNOS \div SMITH\_PNOS$

$RESULT \leftarrow \pi_{FNAME, LNAME}(SSNS * EMPLOYEE)$

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

RESULT

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>),\ (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Aggregate Function Operation

Use of the Aggregate Functional operation $\mathcal{F}$

- $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
- $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
- $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
- $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
  - *Note: count just counts the number of rows, without removing duplicates*

We can also use these functions to group.

$\text{Dno}\ \mathfrak{I}\ \text{COUNT Ssn, AVERAGE Salary}\ (\text{EMPLOYEE})$.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

OUTER JOIN

In the other joins, those tuples which do not match are eliminated, using outer join, these are kept.

- LEFT OUTER JOIN: R ⟕ S
  - keeps every tuple in the first or left relation R
  - if no matching tuple is found in S, then attributes of S in join result are filled or "padded" with null values.

- RIGHT OUTER JOIN: R ⟖ S
  - keeps every tuple in the second or right relation S
  - attributes of R are "padded" with null values in unlatching tuples

- FULL OUTER JOIN: R ⟗ S
  - keeps all tuples in both the left and the right relations
  - when no matching tuples are found, padding them with null values as needed.

LEFT OUTER JOIN

$$TEMP\_LEFT \leftarrow (EMPLOYEE \overset{}{\bowtie}_{Ssn=Mgt\_ssn} DEPARTMENT)$$

$$RESULT \leftarrow \pi_{Fname,Minit,\ Lname,\ Dname}\ (TEMP\_LEFT)$$

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

TEMP_LEFT

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno	Dname	Dnumber	Mgr_ssn
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5	NULL	NULL	NULL
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5	Research	5	333445555
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4	NULL	NULL	NULL
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4	Administration	4	987654321
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5	NULL	NULL	NULL
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5	NULL	NULL	NULL
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4	NULL	NULL	NULL
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1	Headquarters	1	888665555

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

RIGHT JOIN

$$TEMP\_RIGHT \leftarrow (EMPLOYEE \bowtie_{Ssn=Mgt\_ssn} DEPARTMENT)$$
$$RESULT \leftarrow \pi_{Fname,Minit,\ Lname,\ Dname} (TEMP\_RIGHT)$$

RESULT

Fname	Minit	Lname	Dname
Franklin	T	Wong	Research
Jennifer	S	Wallace	Administration
James	E	Borg	Headquarters

All tuples in the RIGHT relations DEPARTMENT are matched

OUTER JOIN

$$TEMP\_OUTER \leftarrow (EMPLOYEE \mathbin{⟗}_{Ssn=Mgt\_ssn} DEPARTMENT)$$
$$RESULT \leftarrow \pi_{Fname,Minit,\ Lname,\ Dname} (TEMP\_OUTER)$$

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

Same as LEFT JOIN…

OUTER UNION Operations

- Created to take union of tuples from two relations which are not type compatible, it is sufficient that they are partially compatible. Some of them do, some don't.
- Those that are type compatible are represented only once in the result and the others are kept in the relation.

Example: Consider an OUTER UNION of these two relations: Employees OUTER UNION Supervisors

Employees

Fname	SSN	Dno	Supervisor
John	123456789	5	Franklin
Franklin	333445555	5	James
Alicia	999887777	4	Jennifer
Jennifer	987654321	4	James
Ramesh	666884444	5	Franklin
Joyce	453453453	5	Franklin
Ahmad	987987987	4	Jennifer
James	888665555	1	NULL

Supervisors

Fname	SSN	Dno	Rank
Franklin	333445555	5	Vice President
Jennifer	987654321	4	Vice President
James	888665555	1	President
Lela	222222222	3	Lecturer
Martin	333333333	3	Lecturer

Tuples from the two relations are matched based on having the same combination of values of the shared attributes— **Fname, SSN, Dno**.
If tuples match on these attributes, both Supervisor and Rank will have a value; otherwise, one of these two attributes will be null.

The result relation EMPLOYEES_OR_SUPERVISORS will have the following attributes : <**Fname, SSN, Dno**, Supervisor, Rank>

Employees

Fname	SSN	Dno	Supervisor
John	123456789	5	Franklin
Franklin	333445555	5	James
Alicia	999887777	4	Jennifer
Jennifer	987654321	4	James
Ramesh	666884444	5	Franklin
Joyce	453453453	5	Franklin
Ahmad	987987987	4	Jennifer
James	888665555	1	NULL

Supervisors

Fname	SSN	Dno	Rank
Franklin	333445555	5	Vice President
Jennifer	987654321	4	Vice President
James	888665555	1	President
Lela	222222222	3	Lecturer
Martin	333333333	3	Lecturer

Employees_Or_Supervisors

Fname	SSN	Dno	Supervisor	Rank
John	123456789	5	Franklin	NULL
Franklin	333445555	5	James	Vice President
Alicia	999887777	4	Jennifer	NULL
Jennifer	987654321	4	James	Vice President
Ramesh	666884444	5	Franklin	NULL
Joyce	453453453	5	Franklin	NULL
Ahmad	987987987	4	Jennifer	NULL
James	888665555	1	NULL	President
Lela	222222222	3	NULL	Lecturer
Martin	333333333	3	NULL	Lecturer

Functional Dependencies and Normalisation for Relational Databases

Semantics: how to interpret the attribute values stored in a tuple of the relation, i.e. how the attribute values in a tuple relate to one another. The easier it is to explain the semantics, the better the relation schema design.

Guidelines

Guideline 1: each tuple in a relation should represent one entity or relationship instance.

- Attributes of different entities should not be mixed in the same relation.
- Only foreign keys should be used to refer to other entities.
- Entity and relationship attributes should be kept apart as much as possible.

This is done to avoid information being stored redundantly as that wastes storage and causes problems with update anomalies.

Example:

Consider the relation:

- EMP_PROJ (Emp#, Proj#, Ename, Pname, No_hours)

Insert Anomaly:

- Cannot insert a ***new*** project unless an employee is assigned to it.

Conversely

- Cannot insert an employee unless a he/she is assigned to a project.

Also: consistency problems when assigning projects to ***every*** employee

Guideline 2: design a schema that does not suffer from insertion, deletion and update anomalies. If there are anomalies present, note them so that applications can take them into account.

Guideline 3: relations should be designed such that their tuples have as few NULLs as possible. Attributes that are NULL could be placed in separate relations (with the primary key).

> Null values are bad because they waste storage, interfere with understanding of attributes, cause problems with aggregate functions.
>
> ***Example: if 10% of employees have individual offices, attribute should not be included in EMPLOYEE relation. Instead, use separate relation: EMP_OFFICES (ESSN, OFFICE-NUMBER).***

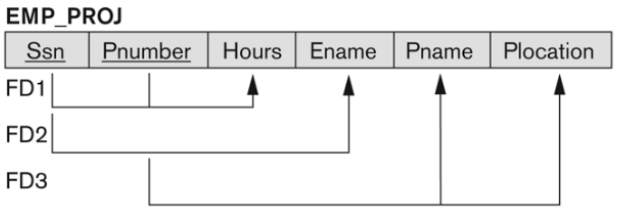
Guideline 4: design relation schemas that can be joined with equality conditions on attributes that are either primary keys or foreign keys, to guarantee that there are no spurious tuples. Avoid relations that contain matching attributes that are not foreign key, primary key, if you join these attributes spurious tuples will be created.

To create an optimal database we should avoid:

- Anomalies that cause redundant work during insertion or modification, as well as accidental loss of information during deletion.
- Waste of storage space due to nulls, difficulty in join or aggregation functions due to null values.
- Generation of invalid or spurious data during joins or improperly related base relations.

Functional Dependencies (FDs)

- Used to specify formal measures of how good a relational database design is
- Keys are used to define normal forms for relations
- Constraints are derived from meaning and interrelationships of the data.



- Social security number determines employee name
  - SSN → ENAME
- Project number determines project name and location
  - PNUMBER → {PNAME, PLOCATION}
- Employee ssn and project number determine the hours per week that the employee works on the project
  - {SSN, PNUMBER} → HOURS

Inference Rules for FDs

Armstrong's interference rules:

- IR1: **Reflexive** – If Y subset-of X, then X → Y
- IR2: **Augmentation** – If X → Y and Y → Z, then XZ → YZ
- IR3: **Transitive** – if X → Y and Y → Z, then X → Z

These IRs form a sound and complete set of rules (which means that all other rules can be deduced form these).

Additional Inference rules:

- **Decomposition:** if X → YZ, then X → Y and X → Z
- **Union:** if X → Y and X → Z, then X → YZ
- **Pseudotransitivity:** If X → Y and WY → Z, then WX → Z

Normal Forms based on Primary Keys

Normalisation: the process of decomposing unsatisfactory relations by breaking up their attributes into smaller relations.

Normal forms: condition using keys and FDs of a relation to certify whether a relation schema is in particular form.

Denormalisation: the process of storing the join of higher normal form relations as a base relation – which is in lower form.

Definitions of Keys and attributes participating in keys

If a relation has more than one key, each is called a **candidate** key. One candidate key is se tto be the **primary key** and the others are called **secondary keys**.

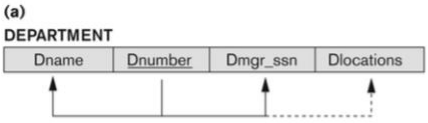
Prime: an attribute that is member of some candidate key.

Nonprime: an attribute that is not a prime attribute (not a member of any candidate key)

First Normal Form

Disallows:

- Composite attributes – we make all the attributes individual attributes.
- Multivalued attributed – we create a new table
- Nested relations; attributes whose values for an individual tuple are non-atomic – we create a new table



(b)
DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)
DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

To normalise: remove attribute that causes the problem and place in separate relation together with the primary key:

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

(a)
EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

(b)
EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, AliciaJ.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)
EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Nested relations, as with composite attributes, are disallowed under 1NF.

Relation EMP_PROJ is NOT in 1NF.

Primary keys: SSN and Pnumber within nested relation.

To normalise: remove nested relation and place in separate relation together with the primary key, as in (c).

Second Normal Form

Definitions

Prime attribute: an attribute that is member of the primary key.

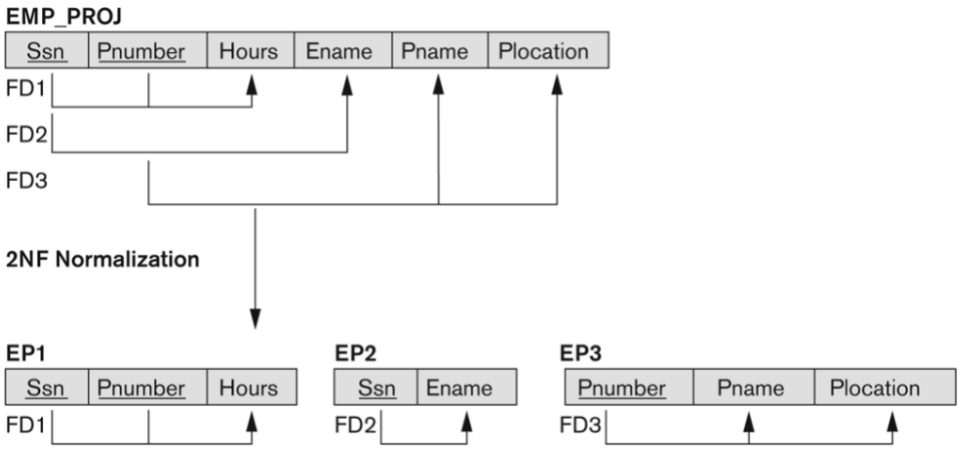
Full functional dependency: a FD Y → Z where removal of any attribute from Y means the FD does not hold any more.

- {SSN, PNUMBER} → HOURS is a full FD
  - since neither SSN → HOURS nor PNUMBER → HOURS hold
- {SSN, PNUMBER} → ENAME is not a full FD
  - since SSN → ENAME also holds
  - it is called a *partial* dependency

Form

A relation schema is in **2NF** if every non-prime attribute A is fully functionally dependent on the primary key.

- The left hand side in the FD is part of the primary key.
- If the primary key is one attribute then this check is not necessary.



Third Normal Form

Definitions

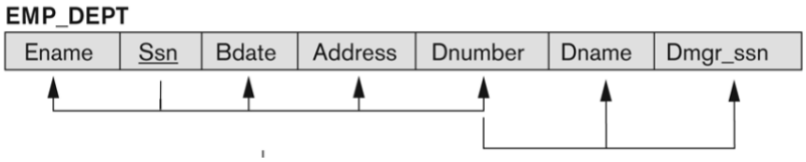
Transitive functional dependency: a FD X → Y that can be derived from two FDs: X → Z and Z → Y.

SSN → DMGRSSN is a **transitive** FD

- SSN → DNUMBER and DNUMBER → DMGRSSN hold

SSN → ENAME is **non-transitive**

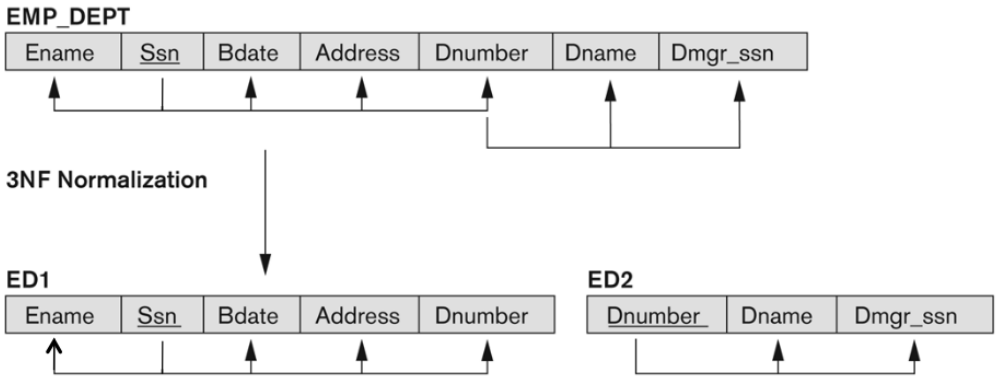
- Since there is no set of attributes X where SSN → X and X→ ENAME



Form

A relation is in **3NF** if:

- It's in 2NF
- No non-prime attribute is transitively dependent on the primary key.



Normal Form Summary

- 1NF – all attributes depend on the key
- 2NF – all attributes depend on the whole key
- 3NF – all attributes depend on nothing but the key.

Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

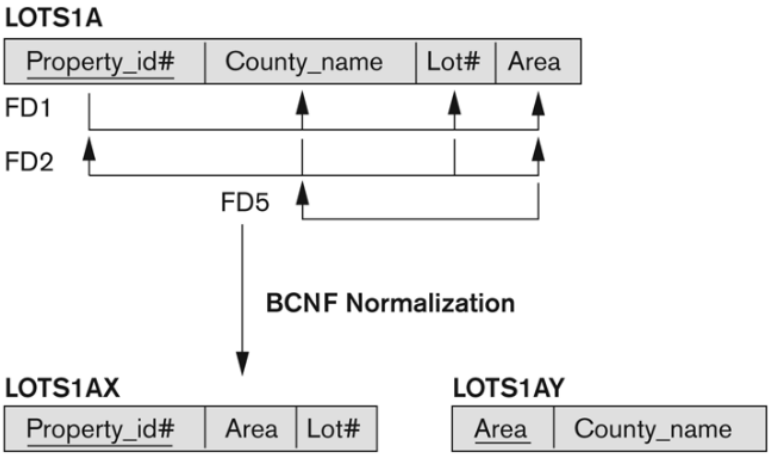
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

A relation schema is in 3NF if when FD X → A in R:

- X is a superkey of R, or
- A is a prime attribute of R

Boyce-Codd Normal Form (BCNF)

A relational schema is in BCNG if whenever a FD X → A holds, X is a superkey of R. ONLY, A cannot be a prime attribute.



FD Area → County_name holds.

The relation is in 3NF but not in BCNF, so decomposition as shown is required.