

ISYS2120: Data & Information Management

Week 2: SQL and Conceptual Data Design

(Kifer/Bernstein/Lewis – Chapter 4; Ramakrishnan/Gehrke – Chapter 2)

Alan Fekete / Matloob Khushi
School of Information Technologies



Outline

- Introduction to SQL
- Joins and Aggregate Functions
- Conceptual Database Design using the Entity Relationship Model



Based on slides from Kifer/Bernstein/Lewis (2006) “Database Systems” and from Ramakrishnan/Gehrke (2003) “Database Management Systems”, and including material from Ullman, Fekete and Röhms.



SQL - The Structured Query Language

- SQL is the standard *declarative* query language for RDBMS
 - ▶ Describing *what* data we are interested in, but *not how* to retrieve it.
- Based on SEQUEL
 - ▶ Introduced in the mid-1970's as the query language for IBM's System (Structured English Query Language)
- ANSI standard since 1986, ISO-standard since 1987
- 1989: revised to SQL-89
- 1992: more features added – **SQL-92**
- 1999: major rework – **SQL:1999** (SQL 3)
- SQL:2003 – 'bugfix release' of SQL:1999 plus SQL/XML
- SQL:2008 – slight improvements, e.g. INSTEAD OF triggers



SQL Overview

■ DDL (Data Definition Language)

- ▶ Create, drop, or alter the relation schema

■ DML (Data Manipulation Language)

- ▶ The retrieval of information stored in the database
 - A **Query** is a statement requesting the retrieval of information
 - The portion of a DML that involves information retrieval is called a **query language**
- ▶ The insertion of new information into the database
- ▶ The deletion of information from the database
- ▶ The modification of information stored in the database

■ DCL (Data Control Language)

- ▶ Commands that control a database, including administering privileges and users

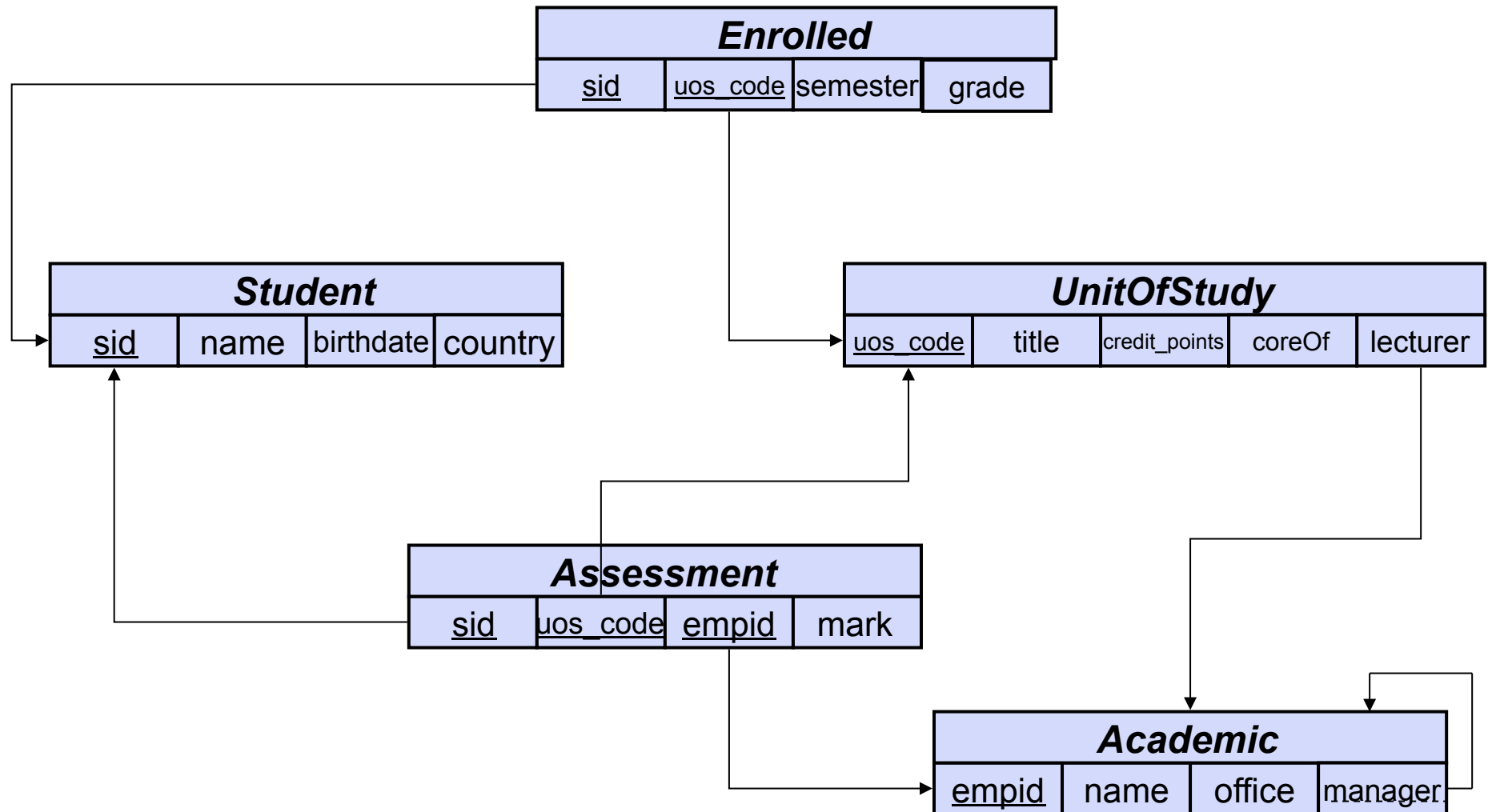


SELECT Statement

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
 - ▶ **SELECT** Lists the columns (and expressions) that should be returned from the query
 - ▶ **FROM** Indicate the table(s) from which data will be obtained
 - ▶ **WHERE** Indicate the conditions to include a tuple in the result
 - ▶ **GROUP BY** Indicate the categorization of tuples
 - ▶ **HAVING** Indicate the conditions to include a category
 - ▶ **ORDER BY** Sorts the result according to specified criteria
- The result of an SQL query is a relation
- a Select-From-Where (SFW) query is equivalent to the relational algebra expression:
$$\Pi_{A_1, A_2, \dots, A_n} (\sigma_{condition} (R_1 \times R_2 \times \dots \times R_m))$$



Running Example - Database Schema



Example: Basic SQL Query

- List the names of all Australian students.

```
SELECT name FROM Student WHERE country='AUS'
```

- Corresponding relational algebra expression

$$\pi_{name} (\sigma_{country='AUS'} (Student))$$

- Note: SQL does not permit the '-' character in names, and SQL names are case insensitive, i.e. you can use capital or small letters.
 - ▶ You may wish to use upper case where-ever we use bold font.



Example: Order By Clause

- List all students (name) from Australia in alphabetical order.

```
select name
from Student
where country='AUS'
order by name
```

- Two options (per attribute):
 - ▶ **ASC** ascending order (default)
 - ▶ **DESC** descending order
- You can order by more than one attribute
 - ▶ e.g., **order by** country **desc**, name **asc**



Duplicates

- In contrast to the relational algebra, SQL allows duplicates in relations as well as in query results.
- To force the elimination of duplicates, insert the keyword **distinct** after **select**.
- Example: List the countries where students come from.

```
select distinct country  
from Student
```

- The keyword **all** specifies that duplicates not be removed.

```
select all country  
from Student
```



Arithmetic Expressions in Select Clause

- An asterisk in the select clause denotes “all attributes”

```
SELECT *  
FROM Student
```

- The select clause can obtain arithmetic expressions involving the operations +, -, * and /, and operating on constants or attributes of tuples.

- The query:

```
SELECT uos_code, title, credit_points*2, lecturer  
FROM UnitOfStudy
```

would return a relation which is the same as the *UnitofStudy* relation except that the credit-point-values are doubled.



The Rename Operation

- SQL allows renaming relations and attributes using the **as** clause:

old_name as new_name

- This is very useful to give, e.g., result columns of expressions a meaningful name.

- Example:

- ▶ Find the student id, grade and lecturer of all assessments for PHYS1001; rename the column name *empid* as *lecturer*.

```
select sid, empid as lecturer, grade
from Assessment
where uos_code = 'PHYS1001'
```



The WHERE Clause

- The where clause specifies conditions that the result must satisfy
 - ▶ corresponds to the selection predicate of the relational algebra.
- Comparison operators in SQL: = , > , >= , < , <= , != , <>
- Comparison results can be combined using the logical connectives **and**, **or**, and **not**.
- Comparisons can be applied to results of arithmetic expressions
- Example: Find all UoS codes for units taught by employee 1011 that are worth more than four credit points:

```
SELECT uos_code
FROM UnitOfStudy
WHERE lecturer = 1011 AND credit_points > 4
```



The WHERE Clause (cont'd)

- SQL includes a between comparison operator (called “range queries”)
 - ▶ Example: Find all students (by SID) who gained marks in the distinction and high-distinction range in COMP5138.

```
SELECT sid
  FROM Assessment
 WHERE uos_code = 'COMP5138' AND
       mark BETWEEN 75 AND 100
```



String Operations

- SQL includes a string-matching operator for comparisons on character strings.
 - ▶ **LIKE** is used for string matching
- Patterns are described using two special characters (“wildcards”):
 - ▶ percent (%). The % character matches any substring.
 - ▶ underscore (_). The _ character matches any character.
- List the titles of all “COMP” unit of studies.

```
select title
  from UnitOfStudy
 where uos_code like 'COMP%'
```

- SQL supports a variety of string operations such as
 - ▶ concatenation (using “||”)
 - ▶ converting from upper to lower case (and vice versa)
 - ▶ finding string length, extracting substrings, etc.



Regular Expression Matches

- New since SQL:2003: regular expression string matching
 - ▶ typically implemented as set of SQL functions, e.g. **regexp_like(...)**
- What are regular expressions?
 - ▶ Pattern consisting of *character literals* and/or *metacharacters*
 - ▶ *metacharacters* specify how to process a regular expression
 - () grouping
 - | alternative
 - [] character list
 - . matches any character
 - * repeat preceeding pattern zero, one, or more times
 - + repeat preceeding pattern one or more times
 - ^ start of a line
 - \$ end of line
- Example:

```
select title
  from UnitOfStudy
 where regexp_like(uos_code, '^COMP[:digit:]{4}')
```



Date and Time in SQL

SQL Type	Example	Accuracy	Description
DATE	'2012-03-26'	1 day	a date (some systems incl. time)
TIME	'16:12:05'	ca. 1 ms	a time, often down to nanoseconds
TIMESTAMP	'2012-03-26 16:12:05'	ca. 1 sec	Time at a certain date: SQL Server: DATETIME
INTERVAL	'5 DAY'	years - ms	a time duration

■ Comparisons

- ▶ Normal time-order comparisons with '=', '>', '<', '<=', '>=', ...

■ Constants

- ▶ CURRENT_DATE db system's current date
- ▶ CURRENT_TIME db system's current timestamp

■ Example:

```
SELECT name
FROM Student
WHERE enrolmentDate < CURRENT_DATE
```



Date and Time in SQL (cont'd)

- Database systems support a variety of date/time related ops
 - ▶ Unfortunately not very standardized – a lot of slight differences
- Main Operations
 - ▶ **EXTRACT**(*component* **FROM** *date*)
 - e.g. `EXTRACT(year FROM enrolmentDate)`
 - ▶ **DATE** *string* (Oracle syntax: `TO_DATE(string,template)`)
 - e.g. `DATE '2012-03-01'`
 - Some systems allow templates on how to interpret *string*
 - Oracle syntax: `TO_DATE('01-03-2012', 'DD-Mon-YYYY')`
 - ▶ **+/- INTERVAL**:
 - e.g. `'2012-04-01' + INTERVAL '36 HOUR'`
- Many more -> check database system's manual



The FROM Clause

- The **from** clause lists the relations involved in the query
 - ▶ corresponds to the Cartesian product operation of the relational algebra.
 - ▶ join-predicates must be explicitly stated in the **where** clause

- Examples:

- ▶ Find the Cartesian product *Student* x *UnitOfStudy*

```
SELECT *  
FROM Student, UnitofStudy
```

- ▶ Find the student ID, name, and gender of all students enrolled in ISYS2120:

```
SELECT sid, name, gender  
FROM Student, Enrolled  
WHERE Student.sid = Enrolled.sid AND  
      uos_code = 'ISYS2120'
```



Aliases

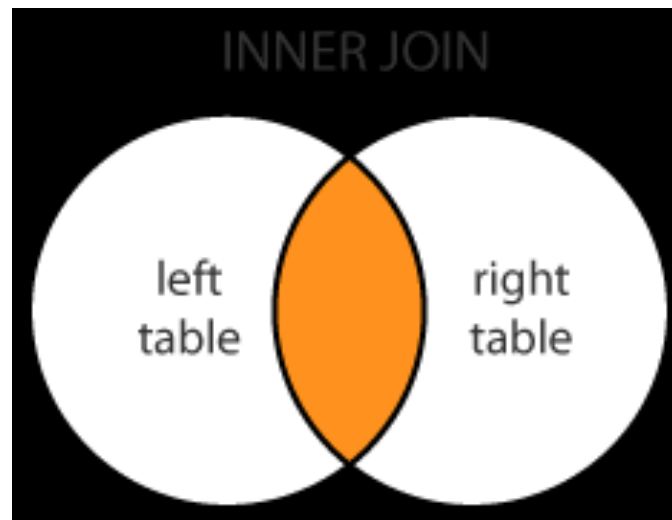
- Some queries need to refer to the same relation twice
- In this case, aliases are given to the relation name
 - ▶ Example: For each academic, retrieve the academic's name, and the name of his or her immediate supervisor.

```
SELECT      A.name, M.name
FROM        Academic A, Academic M
WHERE       A.manager = M.empid
```

- ▶ We can think of **L** and **M** as two different copies of **Academic**; **A** represents lecturers in role of supervisees and **M** represents lecturers in role of supervisors (managers)



SQL Join Queries



Join Example

- Which students did enroll in what semester?

Join involves multiple tables in FROM clause

```
SELECT name, number, semester  
FROM Student S, Enrolled E  
WHERE S.sid = E.sid;
```

WHERE clause performs the equality check for common columns of the two tables



More on Joins

- **Join** – a relational operation that causes two or more tables with a common domain to be combined into a single table or view
- **Equi-join** – a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- **Natural join** – an equi-join in which one of the duplicate columns is eliminated in the result table
- **Outer join** – a join in which rows that do not have matching values in common columns are nonetheless included in the result table (as opposed to inner join, in which rows must have matching values in order to appear in the result table)
- **Union join** – includes all columns from each table in the join, and an instance for each row of each table

The common columns in joined tables are usually the primary key of the dominant table and the foreign key of the dependent table in 1:M relationships



SQL Join Operators

- SQL offers join operators to directly formulate the natural join, equi-join, and the theta join RA operations.

- ▶ R **natural join** S
- ▶ R **inner join** S **on** <join condition>
- ▶ R **inner join** S **using** (<list of attributes>)

- These additional operations are typically used as subquery expressions in the from clause

- ▶ List all students and in which courses they enrolled.

```
select name, uos_code, semester
from Student natural join Enrolled
```

- ▶ Who is teaching “ISYS2120”?

```
select name
from UnitOfStudy inner join Academic on lecturer=empid
where uos_code='ISYS2120'
```



More Join Operators

■ Available join types:

- ▶ inner join
- ▶ left outer join
- ▶ right outer join
- ▶ full outer join

■ Join Conditions:

- ▶ natural
- ▶ on <join condition>
- ▶ using <attribute list>

e.g: Student **inner join** Enrolled **using** (sid)

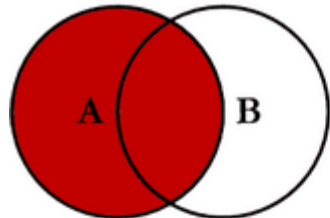
<i>inner join result</i>						
<u>sid</u>	name	birthdate	country	<u>sid2</u>	<u>uos code</u>	grade
112	'A'	01.01.84	India	112	SOFT1	P
200	'B'	31.5.79	China	200	COMP2	C

e.g : Student **left outer join** Enrolled **using** (sid)

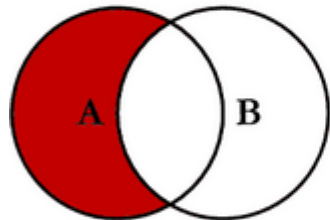
<i>left outer join result</i>						
<u>sid</u>	name	birthdate	country	<u>sid2</u>	<u>uos code</u>	grade
112	'A'	01.01.84	India	112	SOFT1	P
200	'B'	31.5.79	China	200	COMP2	C
210	'C'	29.02.82	Australia	null	null	null



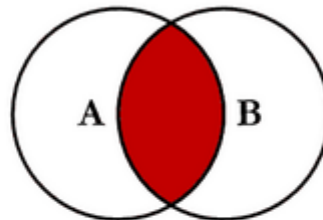
SQL JOINS



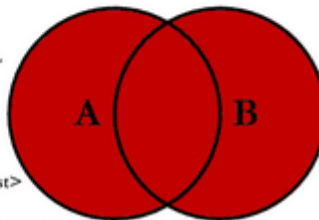
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



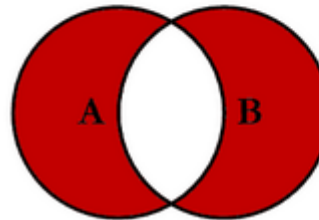
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



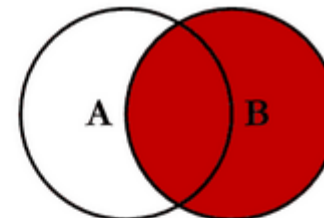
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



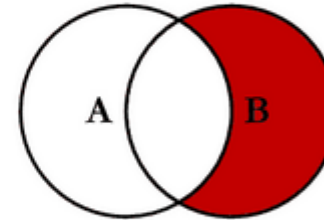
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```

© C.L. Moffatt, 2008

<http://dieswaytoofast.blogspot.com.au/2013/05/sql-joins-visualized.html>

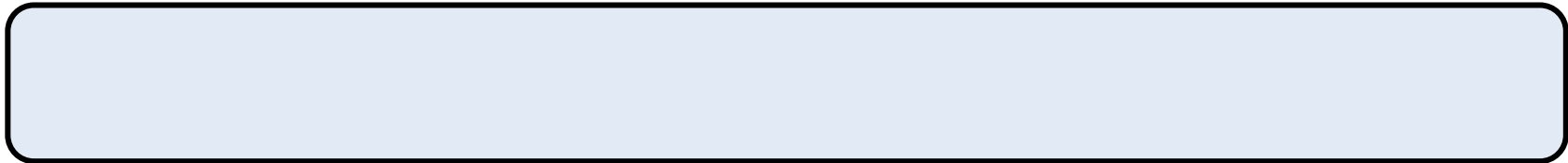


Example Queries I

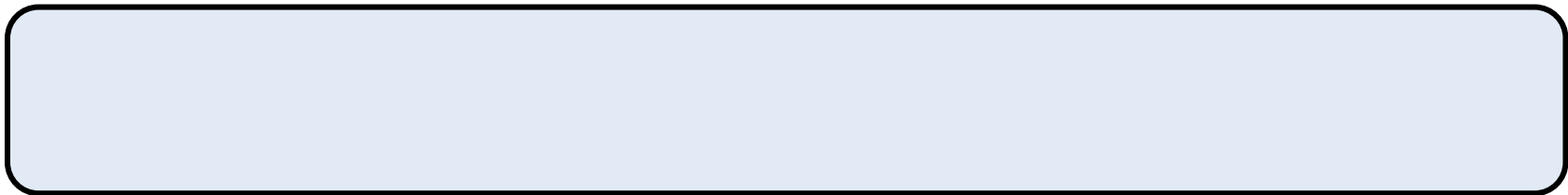
- List all units by title



- List all units of study .



- List in alphabetic order the details of all units of study



Example Queries II

- Find the students with marks between 0 and 10.

- Who is teaching “ISYS2120”?

- List students who got a (high) distinction in ISYS2120.

▶ **Tip:** use **in** (not between) comparison operator for sets



Agenda

- Overview
- Basic SQL Queries
- Join Queries
- **Aggregate Functions and Set Operations**



Aggregate Functions

- These functions operate on the *multiset* of values of a column of a relation, and return a value

avg: average value

min: minimum value

max: maximum value

sum: sum of values

count: number of values

- Must use **distinct** in addition to aggregate over *sets*



Examples for Aggregate Functions

- How many students enrolled?

- Which was the best mark for 'SOFT2007'?

- What was the average mark for SOFT2007?



NULL Values

- It is possible for tuples to have a null value, denoted by **null**, for some of their attributes
 - ▶ Integral part of SQL to handle missing / unknown information
 - ▶ **null** signifies that a value *does not exist*, it does *not mean* “0” or “blank”!
- The predicate **is null** can be used to check for null values
 - ▶ e.g. Find students which enrolled in a course without a grade so far.

```
SELECT sid
FROM Enrolled
WHERE grade IS NULL
```

- Consequence: Three-valued logic
 - ▶ The result of any arithmetic expression involving null is null
 - e.g. 5 + null returns null
 - ▶ However, (most) aggregate functions simply ignore nulls



NULL Values and Three Valued Logic

- Any comparison with *null* returns *unknown*
 - ▶ e.g. $5 < null$ or $null <> null$ or $null = null$
- Three-valued logic using the truth value *unknown*:
 - ▶ OR: $(unknown \text{ or } true) = true$, $(unknown \text{ or } false) = unknown$
 $(unknown \text{ or } unknown) = unknown$
 - ▶ AND: $(true \text{ and } unknown) = unknown$, $(false \text{ and } unknown) = false$,
 $(unknown \text{ and } unknown) = unknown$
 - ▶ NOT: $(\text{not } unknown) = unknown$
- Result of **where** clause predicate is treated as false if it evaluates to unknown
 - ▶ e.g: **select** sid **from** enrolled **where** grade = 'unknown'
ignores all students without a grade so far



NULL Values and Aggregation

- Aggregate functions except **count(*)** ignore null values on the aggregated attributes
 - ▶ result is null if there is no non-null amount

- Examples:

- ▶ Average mark of all assignments

```
SELECT AVG (mark)
FROM Assessment
```

-- ignores tuples with nulls

- ▶ Number of all assignments

```
SELECT COUNT (*)
FROM Assessment
```

-- counts *all* tuples (only with *)



Summary

■ Introduction to SQL

- ▶ SELECT ... FROM ... WHERE ... ORDER BY ...
- ▶ Joins in SQL
- ▶ NULL values and semantic

■ Aggregate Functions

- ▶ Count, Sum, Min, Max, Avg, ...



Exercise Queries

- List all courses by title
- List all information about courses worth 6 crpts.
- In what room is the person teaching ISYS2120
- Find the students with assessments whose mark is less than 10.
- List students who got a distinction or better as their grade in ISYS2120.
- List in alphabetic order the names of all students
- Find the average mark scored in assessments in ISYS2120
- How many students got a grade of 'H' in a course taught by 'Alan Fakete'.



References

Each database textbook has a pretty standard chapter on SQL that covers all commands that we discussed in this lecture:

- Kifer/Bernstein/Lewis (2nd edition – 2006)
 - ▶ Chapter 5
includes some helpful visualisations on how complex SQL is evaluated
- Ramakrishnan/Gehrke (3rd edition - the 'Cow' book (2003))
 - ▶ Chapter 5
uses the famous 'Sailor-database' as examples
- Ullman/Widom (3rd edition – 2008)
 - ▶ Chapter 6
up-to 6.5 good introduction and overview of all parts of SQL querying
- Silberschatz/Korth/Sudarshan (5th edition - 'sailing boat')
 - ▶ Sections 3.1-3.6
- Elmasri/Navathe (5th edition)
 - ▶ Sections 8.4 and 8.5.1



Conceptual Data Model



How to make an idea a reality?

Let's build a house...



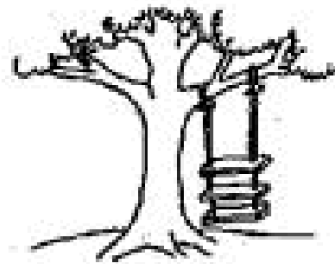
Source: the-self-build-guide.co.uk



Source: www.bg-properties.info



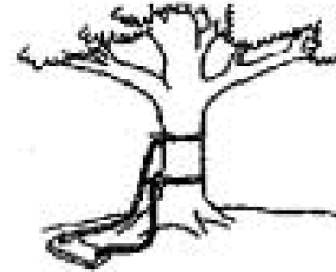
There are some pitfalls though...



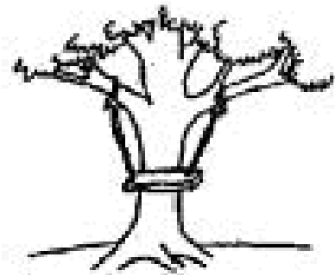
**As proposed
by the project
sponsor.**



**As specified
in the project
request.**



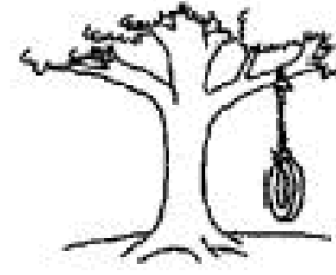
**As designed
by the senior
architect.**



**As produced
by the
engineers.**



**As installed at
the user's
site.**



**What the
customer
really wanted.**

Courtesy: Christopher Alexander, "The Oregon Experiment", 1975.



Database Design Sequence

Requirements Analysis

■ Understand...

- ▶ what data is to be stored
 - ▶ what applications must be built
 - ▶ what operations are most frequent
-

Conceptual Design

■ Develop...

- ▶ high-level description of the data closely matching how users think of the data
 - ▶ Works as communication vehicle
-

Logical Design

■ Convert...

- ▶ conceptual design into a logical database schema
-

Physical Design

■ Convert...

- ▶ logical schema into a physical schema for a specific DBMS and tuned for app.



Conceptual Data Model

- Goal: Specification of database schema
- Methodology:
 - ▶ **Conceptual Design**: A technique for understanding and capturing business information requirements graphically
 - depicts the associations among different categories of data within a business or information system.
 - ▶ Convert conceptual database design to DDL
- Conceptual Database Design does *not* imply how data is implemented, created, modified, used, or deleted.
 - ▶ Works as communication vehicle
 - ▶ Facilitate planning, operation & maintenance of various data resources
- An conceptual data model is *model & database independent*



Conceptual Data Models

- Entity-Relationship Model (ERM)

- Object-oriented Data Models

- ▶ Unified Modelling Language (UML)
- ▶ OMT, Booch, ...

- Etc.

- ▶ Object Role Modelling (ORM)
- ▶ Semantic Object Model (SOM)
- ▶ Semantic Data Models (SDM)
- ▶ KL-ONE etc.



Entity Relationship Model

- Data model for conceptual database design:
High-level *graphical representation* of what data needs to be contained in the system.
 - ▶ Used to interpret, specify, and document database systems.
 - ▶ Tools: ERwin, Sybase Power Designer, ...
 - ▶ E-R diagram templates for drawing tools, e.g., Microsoft Visio
- First designed by Peter Chen in 1976
 - ▶ Several variations have since appeared
 - ▶ Here: **enhanced** or **extended E-R model**



Entity Relationship Model (cont' d)

- A data modeling approach that depicts the associations among different categories of data within a business or information system.
 - ▶ What are the *entities* and *relationships* in the enterprise?
 - ▶ What information about these entities and relationships should we store in the database?
 - ▶ What are the *integrity constraints* or *business rules* that hold?
- A database 'schema' in the ER Model is represented pictorially (*ER diagrams*).
 - ▶ We can convert an ER diagram into a relational schema.
- It is about what data needs to be stored
 - ▶ It does **not** imply how data is created, modified, used, or deleted.



Entities

- **Entity**: a person, place, object, event, or concept about which you want to gather and store data.
 - ▶ it must be distinguishable from other entities
 - ▶ Example: John Doe, unit COMP5138, account 4711
- **Entity Type** (also: **entity set**): a collection of entities that share common properties or characteristics
 - ▶ Example: students, courses, accounts
 - ▶ Rectangle represent entity type
 - ▶ Note: entity sets need not to be disjoint (e.g. person who is manager)
- **Attribute**: describes one aspect of an entity type
 - ▶ Example: people have *names* and *addresses*
 - ▶ depicted by an ellipses



Entity Type

- An **Entity Type** is described by a set of attributes
 - ▶ Descriptive properties possessed by all members of an entity type
 - ▶ Example: Person has ID, Name, Address, Hobbies
- **Domain**: possible values of an attribute
 - ▶ In contrast to relational model values can be complex / set-oriented!
 - **Simple** and **composite** attributes.
 - **Single-valued** and **multi-valued** attributes
 - ▶ Example see next slide
- **Key**: minimal set of attributes that uniquely identifies an entity in the set (several such candidate keys possible)
 - ▶ One chosen as **Primary Key** (PK) => depicted by underlining attr.
- **Entity Schema**: entity type name, attributes (+domains), PK



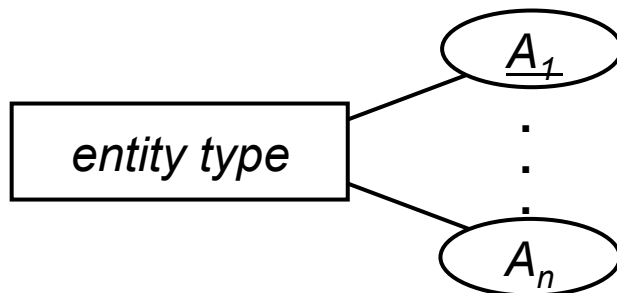
Graphical Representation in E-R Diagram

Symbols:

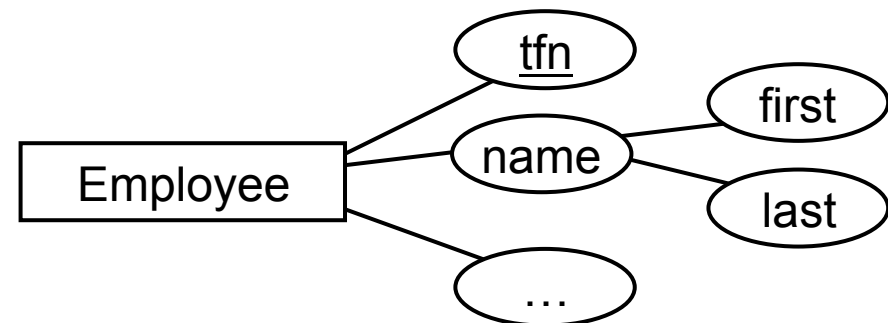
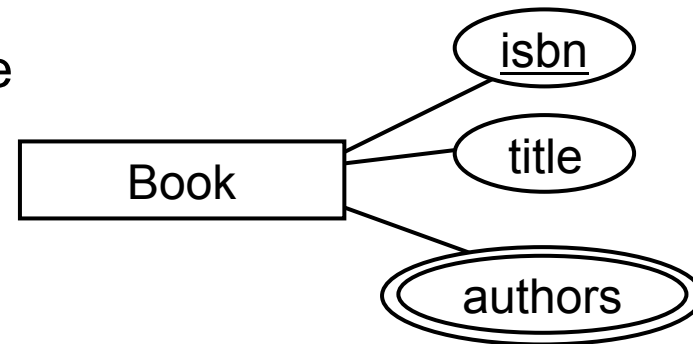
- ▶ Entity Types represented by a rectangle



- ▶ Attributes depicted by ellipses
 - ▶ Double ellipses for multi-valued attributes
 - ▶ Keys are underlined



Examples:



Remarks:

Book.authors is a *multi-valued attribute*;
Employee.name is a *composite attribute*.



Relationships

- **Relationship**: relates two or more entities
 - ▶ number of entities is also known as the **degree** of the relationship
 - ▶ Example: John *is enrolled in* ISYS2120
- **Relationship Type (R.ship Set)**: set of similar relationships
 - ▶ Formally: a relation among $n \geq 2$ entities, each from entity sets:
$$\{(e_1, e_2, \dots, e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots, e_n \in E_n\}$$
 - ▶ Example: **Student** (entity type) related to **Subject** (entity type) by **EnrolledIn** (relationship type).
- **Distinction**:
 - ▶ relation (relational model) - set of tuples
 - ▶ relationship (E-R Model) – describes relationship between entities
 - ▶ Both entity sets and relationship sets (E-R model) may be represented as relations (in the relational model)



Relationship Attributes & Roles

■ Relationship-Attribute

Relationships can also have additional properties

- ▶ E.g., John enrolls in ISYS2120 *in* the Second Semester 2018
- ▶ John and ISYS2120 are related
- ▶ 2018sem2 describes the relationship - value of the *Semester* attribute of the **EnrolledIn** relationship set

■ Relationship-Role

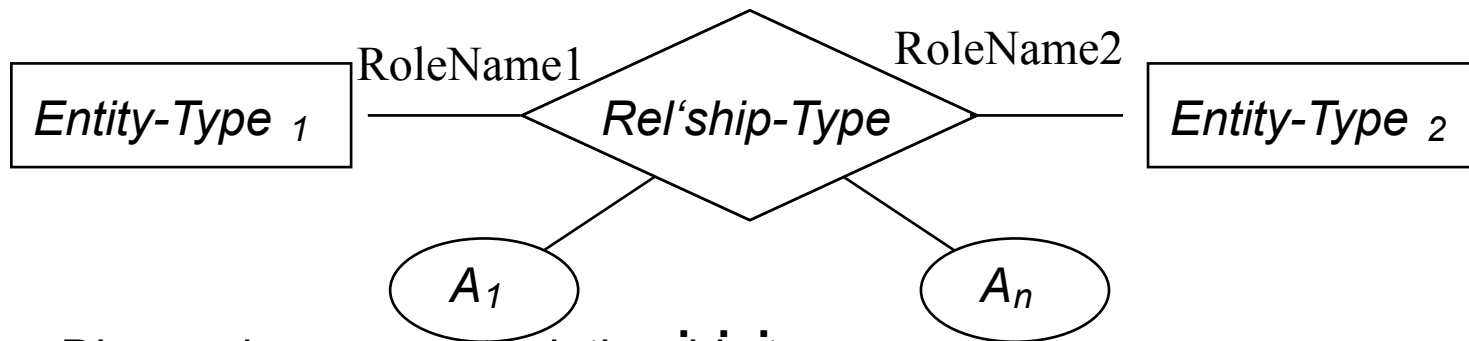
Each participating entity can be named with an explicit role.

- ▶ E.g. John is value of *Student* role, ISYS2120 value of *Subject* role
- ▶ useful for relationship that relate elements of same entity type
- ▶ Example: **Supervises (Employee:Manager, Employee)**



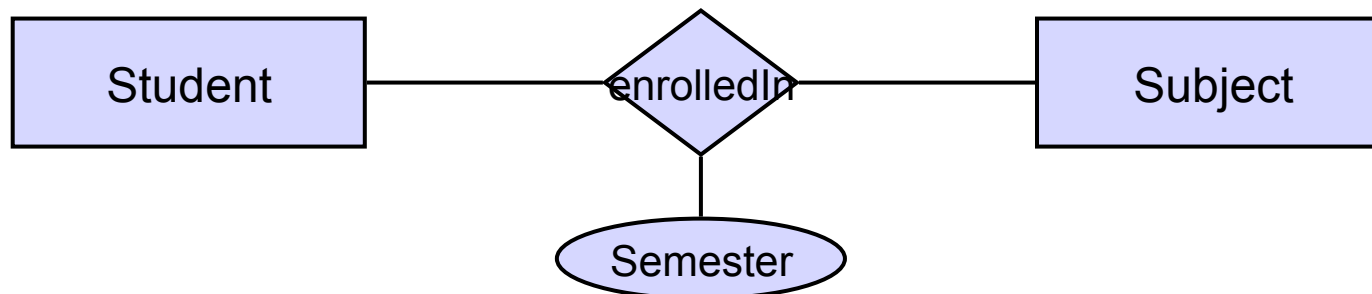
Graphical Representation of Relationships in E-R Diagrams

Symbol:



- ▶ Diamonds represent relationship types
- ▶ Lines link attributes to entity types and entity types to relationship types.
- ▶ Roles are edges labelled with role names
 - ▶ role label is often omitted when same as entity name

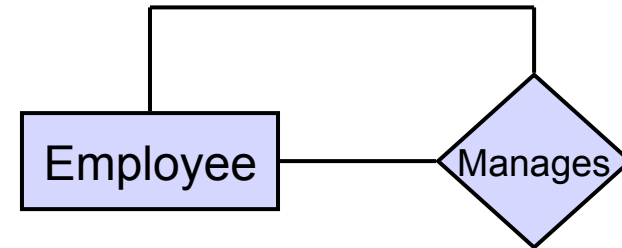
Example



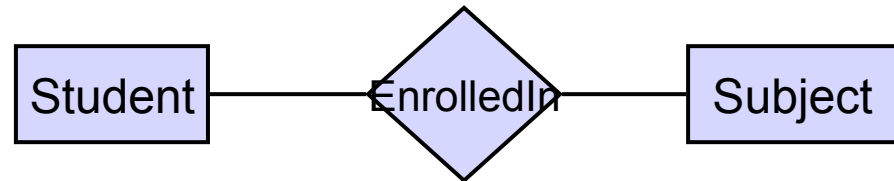
Relationship Degree

- Degree of a Relationship:
of entity types involved

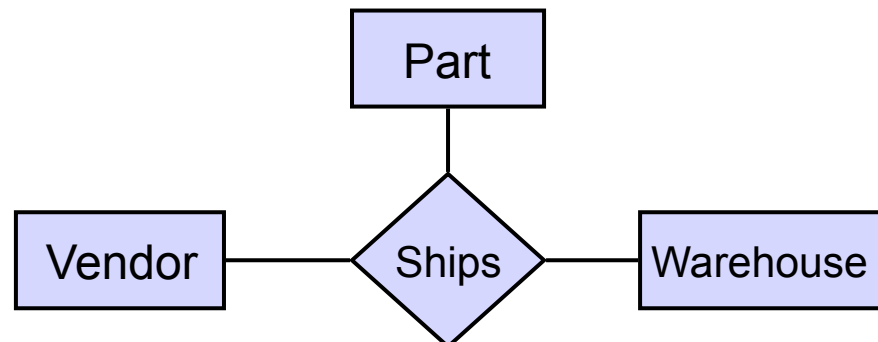
- ▶ Unary Relationship (Recursive)



- ▶ Binary Relationship (most common kind of relationship)

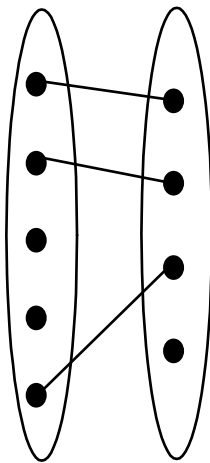


- ▶ Ternary Relationship
(three entity types involved)

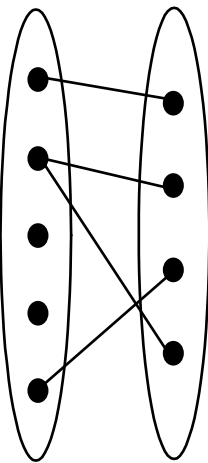


Multiplicity of Relationships

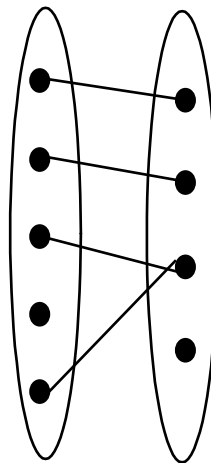
- Consider **Works_In**: An employee can work in many departments; a department can have many employees.
 - ▶ In contrast, each department has at most one manager
- We find examples of each style of relationship
 - ▶ Think carefully about both directions:
 - ▶ To how many instances of B can a given instance of A be related?
 - ▶ To how many instances of A can a given instance of B be related?
- Warning: natural language can be confusing
 - ▶ Many-to-1 means each A is related to *at most* 1 of B



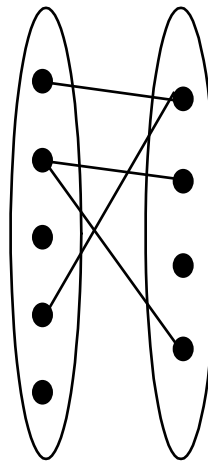
1-to-1



1-to Many



Many-to-1



Many-to-Many

Multiplicities
are depicted in
E-R diagrams as
constraints...
(see next slides)



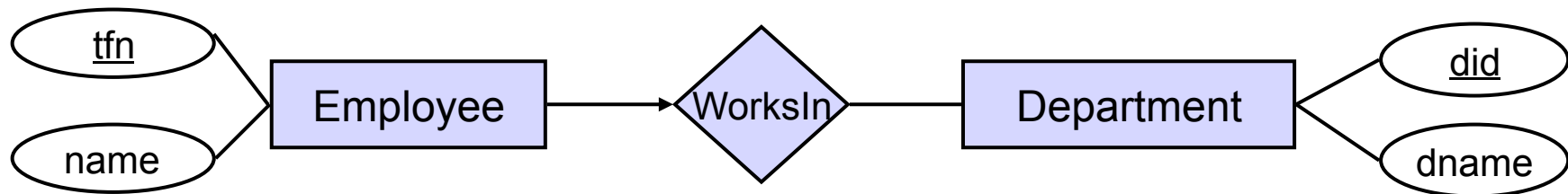
Model comes from domain knowledge

- Consider **Works_In**: we said “An employee can work in many departments; a department can have many employees.”
- This is not true for every enterprise! There may be a policy that assigns each employee to only one department.
- A conceptual data model should indicate whatever we know about the organisation’s business rules
 - ▶ These become constraints that say “every valid database state must have certain characteristics”
 - ▶ These can be captured in a relational design, and then the DBMS can enforce them
- Multiplicity and other constraints are not intrinsic but must be decided in each case, from discussion with the stakeholders
- In ISYS2120, our focus is on how we document the constraint (whatever it is in a particular situation)
 - ▶ and how this impacts on the database schema and contents



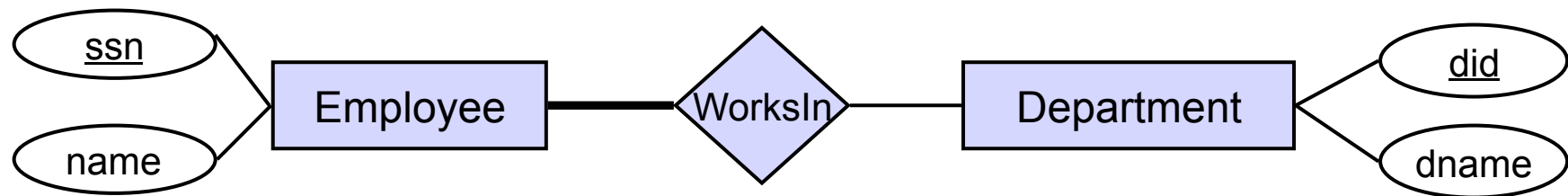
Key Constraints

- If, for a particular participant entity type, each entity participates in at most one relationship, the corresponding role is a key of relationship type
 - ▶ E.g., *Employee* role is unique in *WorksIn*
 - ▶ there may be many employees who are working in a department
 - ▶ also called: **many-to-one** or **N:1 relationship**
- Representation in E-R diagram: arrow from key side to the relationship diamond
- Example: An employee works in at most one department.



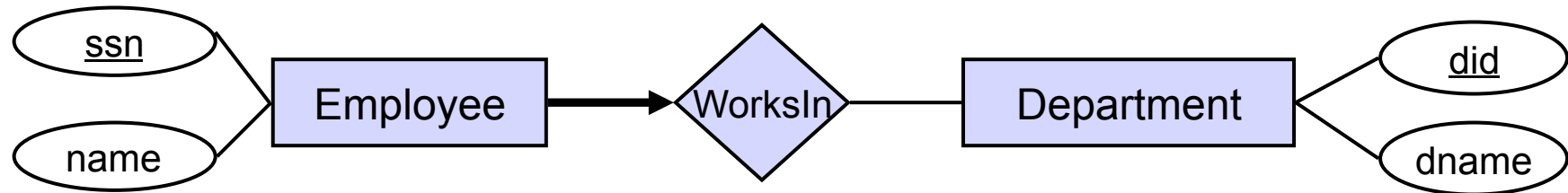
Participation Constraint

- If every entity participates in at least one relationship, a *participation constraint* holds:
 - ▶ Total participation: each entity $e \in E$ must participate in a relationship, it cannot exist without that participation (total participation aka existence dependency).
 - ▶ Partial participation: default; each entity $e \in E$ can participate in a relationship.
- Representation in E-R diagram: thick line from entity type that must participate to relationship diamond
- Example: every employee works in at least one department



Participation and Key Constraint

- If every entity participates in exactly one relationship, both a participation and a key constraint hold.
- Representation in E-R diagrams: thick arrow
- Example:
Every employee works in exactly one department



Cardinality Constraints

- Generalisation of key and participation constraints
- A **cardinality constraint** for the participation of an entity set E in a relationship R specifies how often an entity of set E participates in R at least (minimum cardinality) and at most (maximum cardinality).
 - ▶ In an ER-diagram we annotate the edge between an entity type E and relationship R with min..max, where min is the minimum cardinality and max the maximum cardinality. If no maximal cardinality is specified, we set '*' as max number ("don't care").
- Example: Every employee works in 1 to 3 departments.

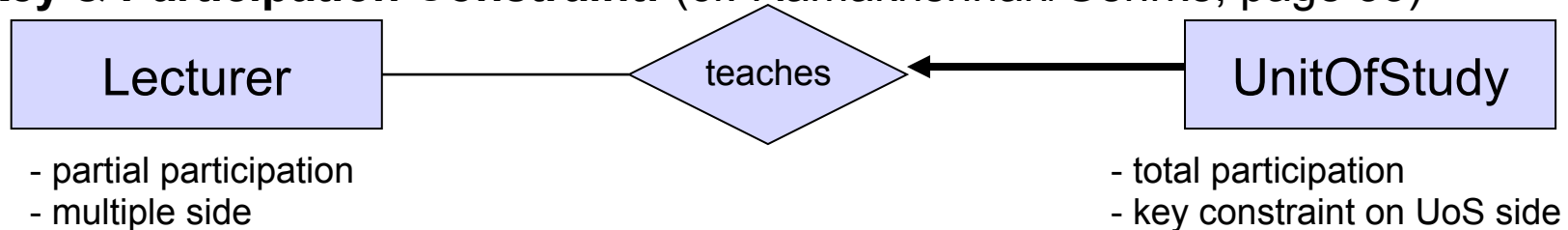


Comparison of Notations

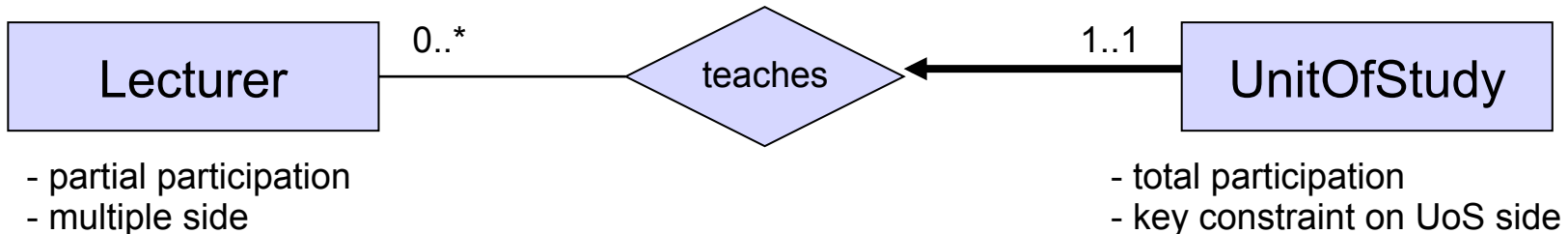
“A lecturer can teach several subjects.”

“Every subject is taught by exactly one lecturer.”

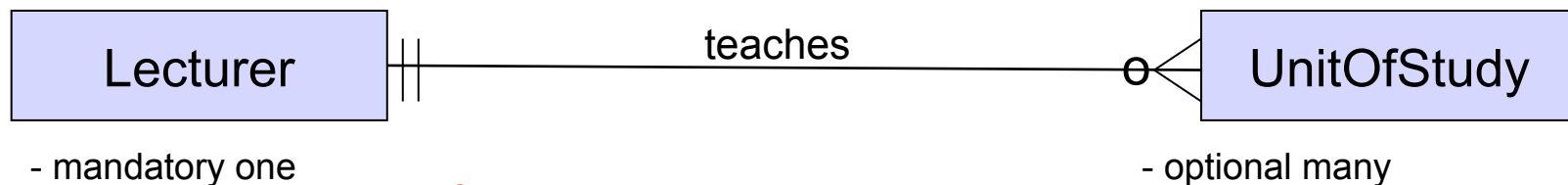
With Key & Participation Constraint: (cf. Ramakrishnan/Gehrke, page 33)



With Cardinality Constraints: (cf. Kifer/Bernstein/Lewis, pages 76/77 and 82/83)



“Crow’ s-foot” notation: (e.g. Hoffer, ed7, pages 95,116-122 – as used in INFO1003)



Be aware that the Crow’ s foot notation puts 1 or many symbol on other end, as compared to our notation! UML is like Crow’s foot in this aspect.



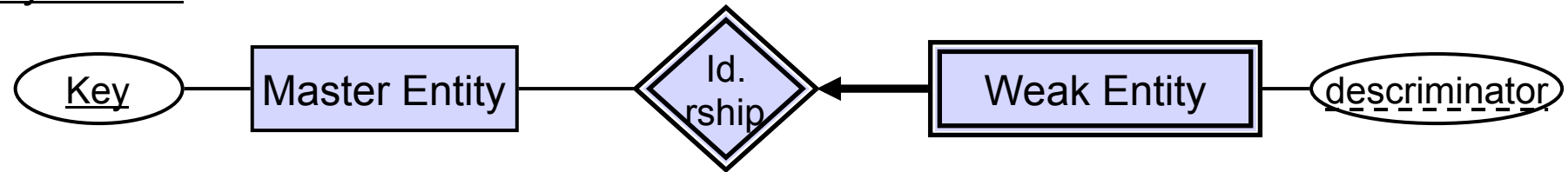
Weak Entities

- **Weak entity type**: An entity type that does not have a primary key.
 - ▶ Can be seen as an **exclusive ‘part-of’ relationship**
 - ▶ Its existence depends on the existence of one or more *identifying entity types*
 - ▶ it must relate to the identifying entity set via a total, one-to-many *identifying relationship type* from the identifying to the weak entity set
 - ▶ Examples:
child from parents, *payment* of a loan
- The **discriminator** (or **partial key**) of a weak entity type is the set of attributes that distinguishes among all the entities of a weak entity type related to the same owning entity.
- The primary key of a weak entity type is formed by the primary key of the strong entity type(s) on which the weak entity type is existence dependent, plus the weak entity type's discriminator.



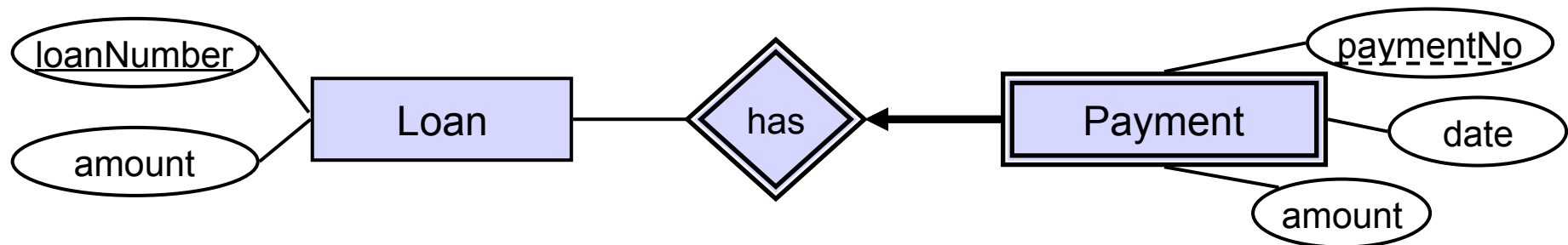
Representation of Weak Entity Types

Symbols:



- ▶ We depict a weak entity type by double rectangles.
- ▶ Identifying relationship depicted using a double diamond
- ▶ underline the discriminator of a weak entity type with a dashed line

Example:



- ▶ paymentNumber: discriminator of the payment entity type
- ▶ Primary key for payment: (loanNumber, paymentNumber)



This week, you have learned...

■ SQL

- ▶ SELECT ... FROM ... WHERE ... ORDER BY ...
- ▶ Joins in SQL
- ▶ NULL values and semantic
- ▶ Aggregate Functions (Count, Sum, Min, Max, Avg, ...)

■ The Database Design Process

- ▶ An understanding of the general database design process and the roles of conceptual and logical data modelling

■ Conceptual Data Modelling using the E-R Model

- ▶ Understanding and experience with conceptual data modelling using the entity-relationship model:
 - Basic Constructs: Entity, Attributes, Relationships, Cardinality Constraints
 - Advanced Concepts: Weak Entities, Inheritance
- ▶ Using the particular notation we covered

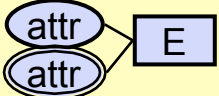
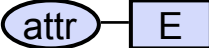

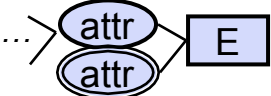
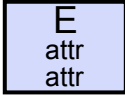
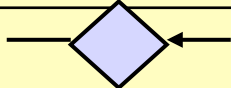
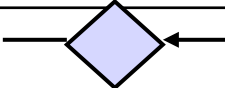
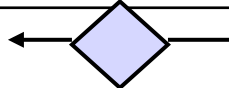
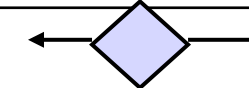
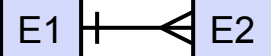
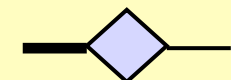
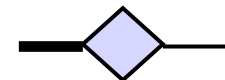
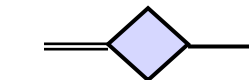
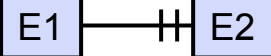
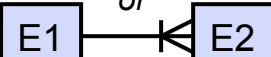
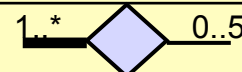
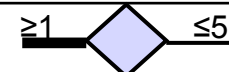
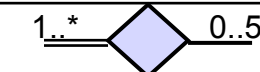
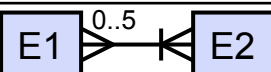

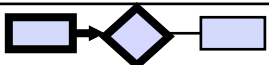
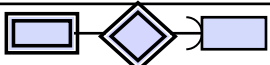
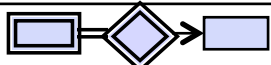
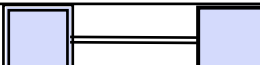


References

- Kifer/Bernstein/Lewis (2nd edition)
 - ▶ Chapter 4
 - ▶ *Lecture uses this ER-D notation and naming scheme; also covers UML*
- Ramakrishnan/Gehrke (3rd edition - the 'Cow' book)
 - ▶ Chapter 2
 - ▶ *similar ER-D notation with few differences (cf. appendix); incl. case study Internet Shop;*
- Ullman/Widom (3rd edition)
 - ▶ Chapter 4
 - ▶ *only basic ER-D with a few differences in its notation;*
- Silberschatz/Korth/Sudarshan (6th edition - 'sailing boat')
 - ▶ Chapter 7
 - ▶ *standard ER-D notation with most details*
- Elmasri/Navathe (7th edition)
 - ▶ Chapters 3 and 4
 - ▶ *uses a different notation for Enhanced E-R diagrams, especially inheritance*



Appendix: Notation Comparison

	Kifer / Bernstein / Lewis (ISYS2120)	Ramakrishnan / Gehrke	Ullman / Widom	Korth / Silberschatz / Sudarshan	Hoffer / Prescott "Crows-Foot"
Entity Name	Entity Type	Entity Set (plural names)	Entity Set (plural names)	Entity Set	Entity Type
Attributes	 only atomic; single- set-valued	 only atomic & single valued	 only single valued (but mention variants with structs & sets)	 single- set-valued composite attr. derived attributes	 single- set-valued composite attr. derived attributes
Key Constraints (1-many relationship)	 (arrow from N-side to diamond)	 (arrow from N- side to diamond)	 (arrow from diamond to 1-side)	 (arrow from diamond to 1-side)	 (no diamond; tick on 1-side, crow's foot on many side)
Participation Constraints	 (thick line on total participation side)	 (thick line on total participation side)	n/a	 (double line - total participation side)	 or 
Cardinality Constraints	 min..max notation	n/a	 limit constraint	 min..max notation	 on opposite side!
Roles	yes	yes	yes	yes	yes
Weak Entity (& identifying rel.ship)					
ISA	