**CS 31 Database Management Programming**

**ER Diagrams – Conceptual**

**Data Model** – a plan, or blueprint, for a database design; it is a generalized, non-DBMS-specific design; a conceptual design

We will be using the extended entity-relationship (E-R) diagram.

**Entity** – something that users want to track

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **EMPLOYEE** | |  |  | **DEPARTMENT** | |  |
| EmpID | EmpName | EmpDept |  | DeptID | DeptName | ManagerID |
| 100 | Josiah | 10 |  | 10 | Accounting | 400 |
| 200 | Andrea | 20 |  | 20 | Engineering | 500 |
| 300 | Itzel | 30 |  | 30 | Sales | 600 |
| 400 | Dina | 10 |  |  |  |  |
| 500 | Bradley | 20 |  |  |  |  |
| 600 | Marie | 30 |  |  |  |  |

**Entity Instance** – a particular occurrence of an entity

e.g. Employee 100 and the Accounting Department

**Entity Class** – a collection of entity instances of a given type

e.g. EMPLOYEE and DEPARTMENT

**Attribute** – a property of an entity

e.g. EmpName and DeptName

An entity instance is described by its attribute values, (EmpID: 100, EmpName: Josiah, EmpDept: 10)

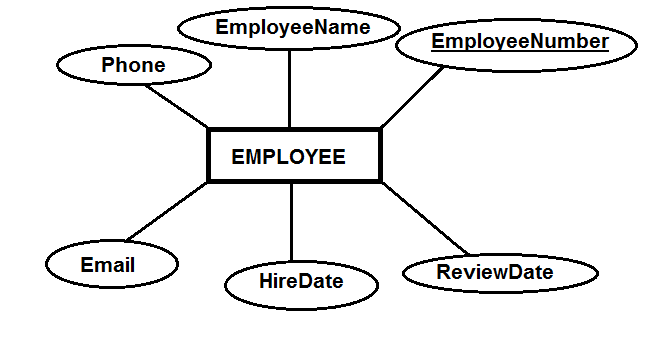
An entity class is described by its attributes, (EmpID, EmpName, EmpDept)

**Identifier** – an attribute that names, or identifies, an entity

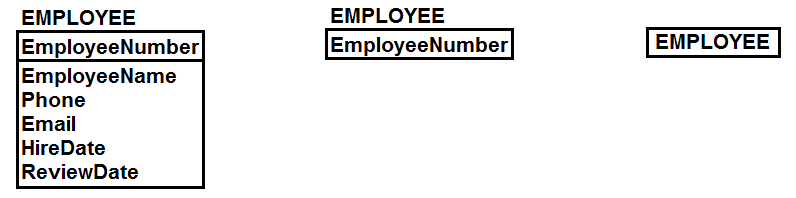
**Composite Identifier** – an identifier consisting of two or more attributes

Think of identifiers as superkeys. They uniquely identify an entity instance (row).

Entity and attributes (ellipses):



Entity and attributes (rectangle):



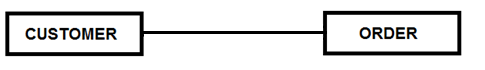
**Relationship** – an association between two entities

A relationship class is an association between entity classes. A relationship instance is an association between entity instances.

e.g. A business may want to track customers and orders (entities). A business rule will identify the relationship. A customer has orders and an order belongs to a customer (relationship class). CUSTOMER 12345 has ORDER 11, 16, and 38 (relationship instance).

NOTE: The entities correspond to nouns and the relationships correspond to verbs.

A line drawn between two entities indicates they are related.



**Degree** – the number of entities participating in the relationship

The book mentions unary, binary and ternary relationships. There could be n-ary relationships. When transforming a model into a database design, relationships of all degrees greater than two are reduced to combinations of binary relationships.

**Maximum Cardinality** – the maximum number of entities on each side of the relationship

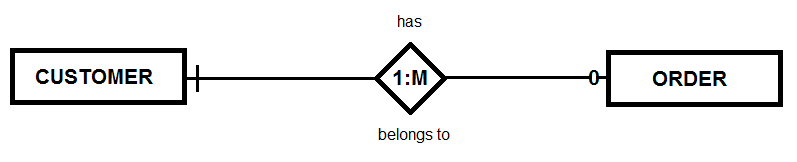
A customer can have many orders and an order belongs to one customer.

**Minimum Cardinality** – the minimum number of entities on each side of the relationship

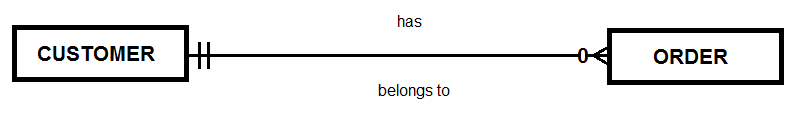
A customer may have an order and an order requires a customer.

**Entity-Relationship (E-R) Diagram** – a graphic used to represent entities and their relationships

Traditional/Chen's Notation (Introduced by Peter Chen):



Crow's Foot:



**SQL Lecture Week 7**

**GROUP BY** – Grouping enables you to divide data into logical sets so you can perform aggregate calculations on each group

**HAVING** – Filters groups based on conditions (supports all operators used in **WHERE** clause)

Some rules/warnings of the **GROUP BY** clause:

* Only selected columns or calculated columns may be used and every selected column must be used
* Required if using columns (or calculated columns) with aggregate functions
* Grouping does not guarantee any specific order of groups – use **ORDER BY**

Let's review the clauses discussed so far and their sequence.

SELECT *columun(s)*

FROM *table(s)*

WHERE *condition(s)*

GROUP BY *column(s)*

HAVING *condition(s)*

ORDER BY *column(s)*

LIMIT *number*;

**Step-by-Step Example for Query 2\_52**

Hopefully, this will help with the understanding of the **GROUP BY** and **HAVING** clauses. Below is the full query.

SELECT Department, COUNT(SKU) AS Dept\_SKU\_Count

FROM sku\_data

WHERE SKU <> 302000

GROUP BY Department

HAVING COUNT(SKU) > 1

ORDER BY Dept\_SKU\_Count;

Start by executing the first three lines without the aggregate column. Leave the aggregate column out until the rows are grouped. The SKU column is added to make it obvious that SKU 302000 is not included in the result set.

SELECT Department, SKU

FROM sku\_data

WHERE SKU <> 302000;

|  |  |
| --- | --- |
| **Department** | **SKU** |
| Water Sports | 100100 |
| Water Sports | 100200 |
| Water Sports | 101100 |
| Water Sports | 101200 |
| Camping | 201000 |
| Camping | 202000 |
| Climbing | 301000 |

Every row is included except the row with **SKU = 302000**. Next we group by Department and count the rows in each group.

The Water Sports group has 4 rows (**COUNT(SKU) = 4**).

|  |  |
| --- | --- |
| Water Sports | 100100 |
| Water Sports | 100200 |
| Water Sports | 101100 |
| Water Sports | 101200 |

The Camping group has 2 rows (**COUNT(SKU) = 2**).

|  |  |
| --- | --- |
| Camping | 201000 |
| Camping | 202000 |

The Climbing group has 1 row (**COUNT(SKU) = 1**).

|  |  |
| --- | --- |
| Climbing | 301000 |

The **HAVING** clause includes groups with a row count greater than 1. The Climbing group will be excluded. Only a single row will appear per group. The result of running the full query is shown below.

|  |  |
| --- | --- |
| Department | Dept\_SKU\_Count |
| Camping | 2 |
| Water Sports | 4 |