Dr Greg Wadley

INFO90002 Database Systems & Information Modelling

Week 02

Data Modelling & SQL (1)

The first half of semester

Week	Lecture 1	Lecture 2
2	Modelling 1 conventions, 1-M relationships	SQL 1 group by, having, joins
3	Modelling 2 M-M, 1-1, unary relationships	SQL 2 subqueries, functions
4	Modelling 3 ternary relationships, views	SQL 3 set operations, formatting
5	Normalization 1st, 2nd, 3rd normal forms	Physical design data types, indexes, denormalising
6	Data Dictionaries	Wrapup and Q&A

end of Week 6: Assignment One on data modelling is due

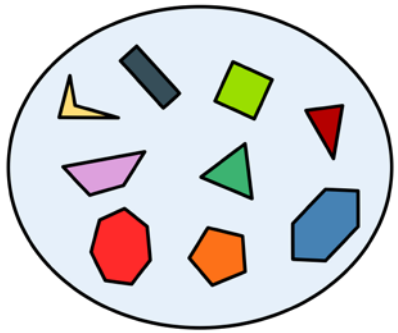
Today's session

- Data modelling
  - ER modelling conventions
  - Identifying entities and business rules
  - one-to-many (1-M) relationships
- SQL
  - Overview and history
  - Create tables
  - Insert data into tables
  - Read data from tables

Databases and ER models

- A database can be thought of as a representation of
  - a collection of entity sets, and
  - relationships between the entities
- An entity is an object or abstract concept or event which can be distinguished from other entities
  - example: product, order, sale, person, movie, tweet
- Entities have attributes that describe the entity and distinguish it from other entities in the same entity set
  - example attributes: EmployeeName, Address

- (reminder – what are “sets”?
  - union, intersection, Cartesian product)



Valid and invalid entities

- An Entity
  - Will have many instances in the database
  - Has several attributes
  - Is necessary for the system to work
- Examples
  - Person: EMPLOYEE, STUDENT, PATIENT
  - Place: STORE, WAREHOUSE, STATE, CITY
  - Object: PRODUCT, MACHINE, BUILDING, VEHICLE
  - Event: SALE, REGISTRATION, BROADCAST
  - Abstract: ACCOUNT, UNI SUBJECT, ROLE
- Entities do not usually include:
  - An output of the system (i.e. a report)
  - The system itself
  - The company that owns the system

Naming Conventions

- Entities
  - singular nouns
    - Employee, Customer, Sale (without or without capital letter)
- Attributes
  - usually a noun
    - itemColour, quantitySold, id
- Relationships
  - verbs or verb phrases
    - has, wants, manages, performs work for
- Use names meaningful to the domain
  - try not to abbreviate names
    - except num or nbr for number, ID for identifier
  - conventions on UPPER and lower case

Modelling a Single Entity

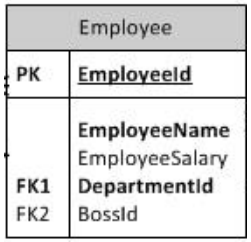
- By searching for nouns in the case we can identify entities

  (for example – Customer)

- What things would we need to record about the Customer
  - these become the customer's Attributes

- How can we identify individual Customers?
  - by name?
  - by address?
- Now we can draw it as an entity in the ER diagram

Mapping ER diagram to database tables



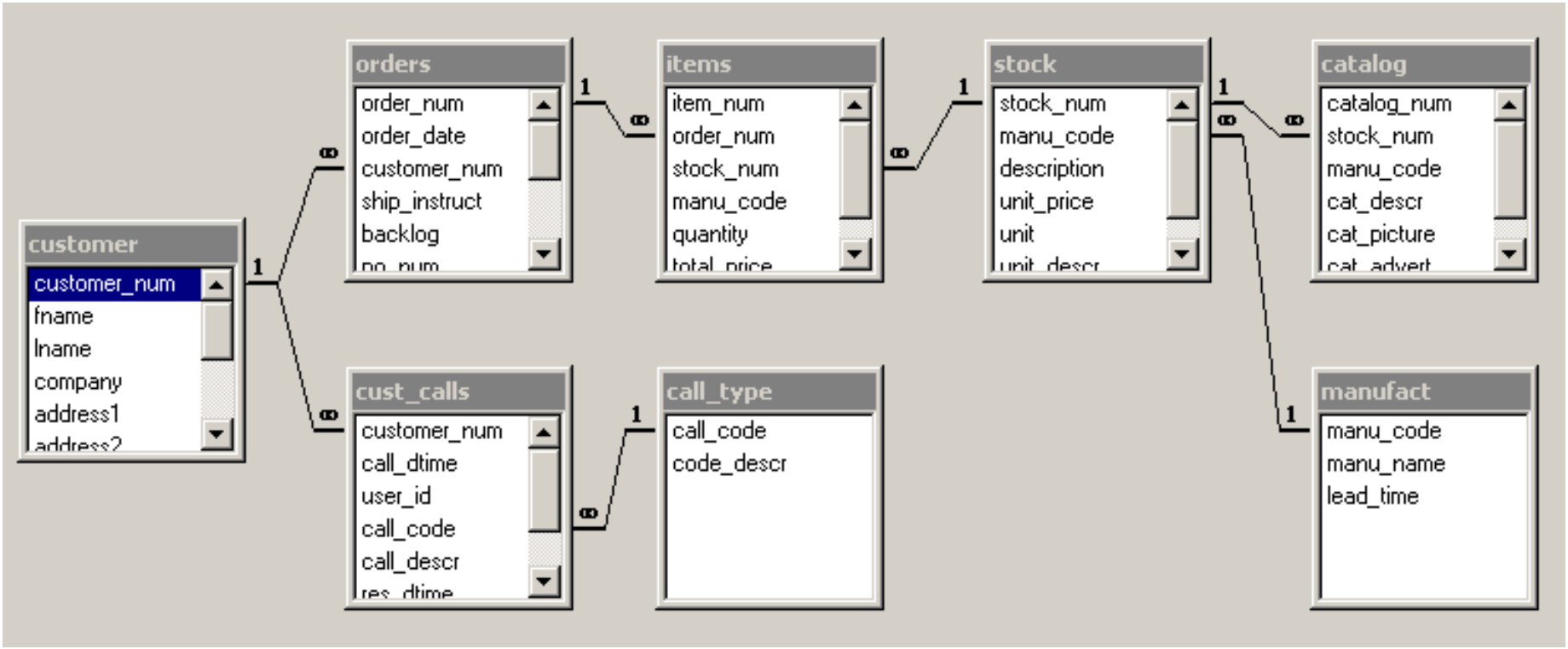
EmployeeID	EmployeeName	EmployeeSalary	DepartmentID	BossID
1	Alice	75000.00	1	0
2	Ned	45000.00	11	1
3	Andrew	25000.00	11	2
4	Clare	22000.00	11	2
5	Todd	38000.00	2	1
6	Nancy	22000.00	2	5
7	Brier	43000.00	9	1
8	Sarah	56000.00	9	7
9	Sophie	35000.00	10	1
10	Sanjay	15000.00	6	3

- Entity set
  - Often corresponds to a table in the database
- Entity instance
  - Often corresponds to a row in a table
- Attribute
  - Often corresponds to a column in a table
- Relationship set (link between entity sets)
  - Often corresponds to a Foreign Key in a table
- Relationship instance (link between entity instances)
  - Foreign Key value = Primary Key value

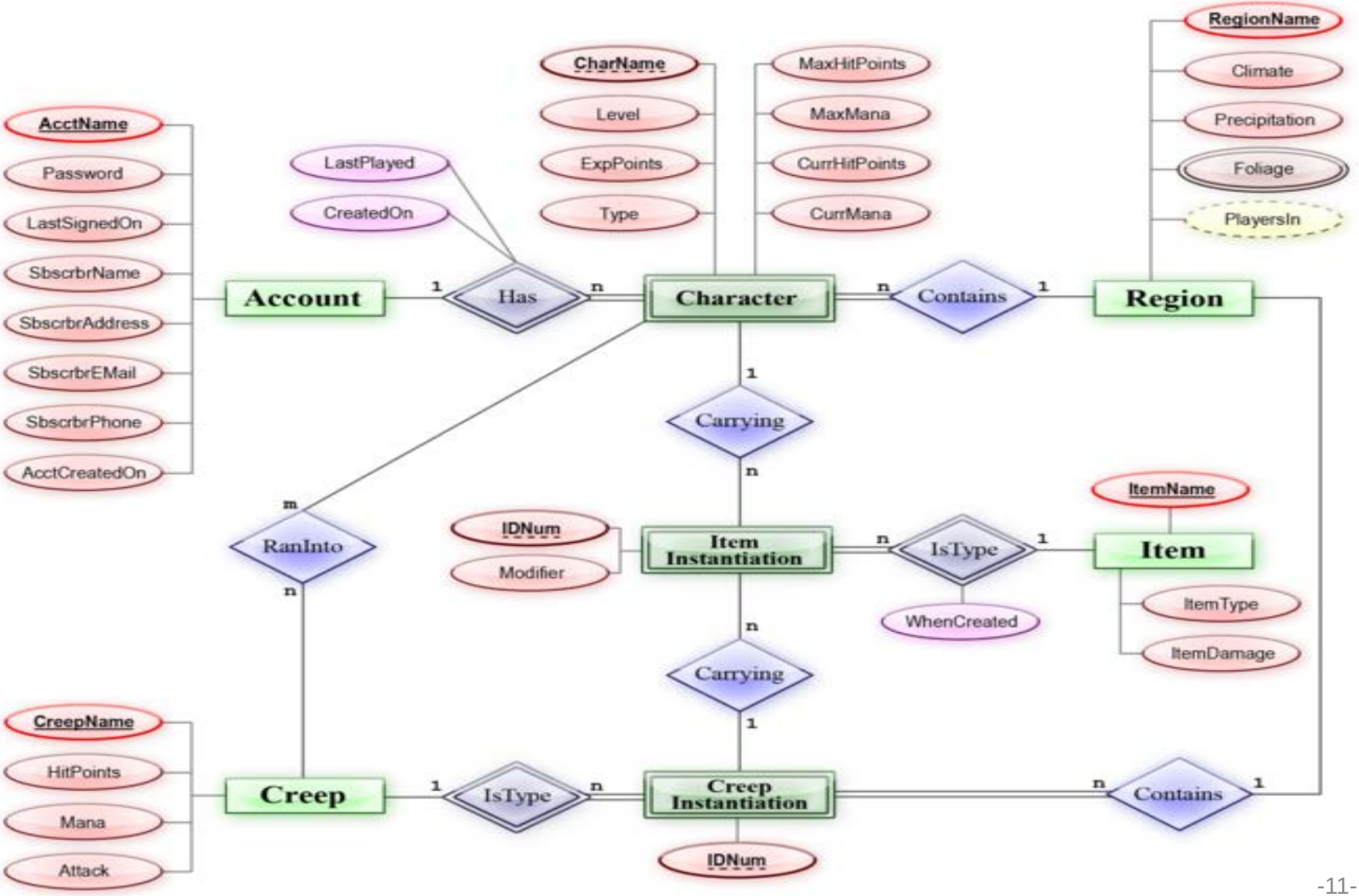
How to draw ER diagrams?

- You can use paper or the software of your choice
- e.g. Visio, https://www.lucidchart.com/ …
- Assignment 1 model should be made in *MySQL Workbench*
- In the exam you'll need to model *on paper*
- Diagrams in lecture slides are made in Workbench and Visio
- My suggestion is:
  - Use pen-and-paper or whiteboard for early Conceptual modelling
  - Use paper, Visio or Workbench for subsequent Logical modelling
  - Use Workbench for the final Physical model

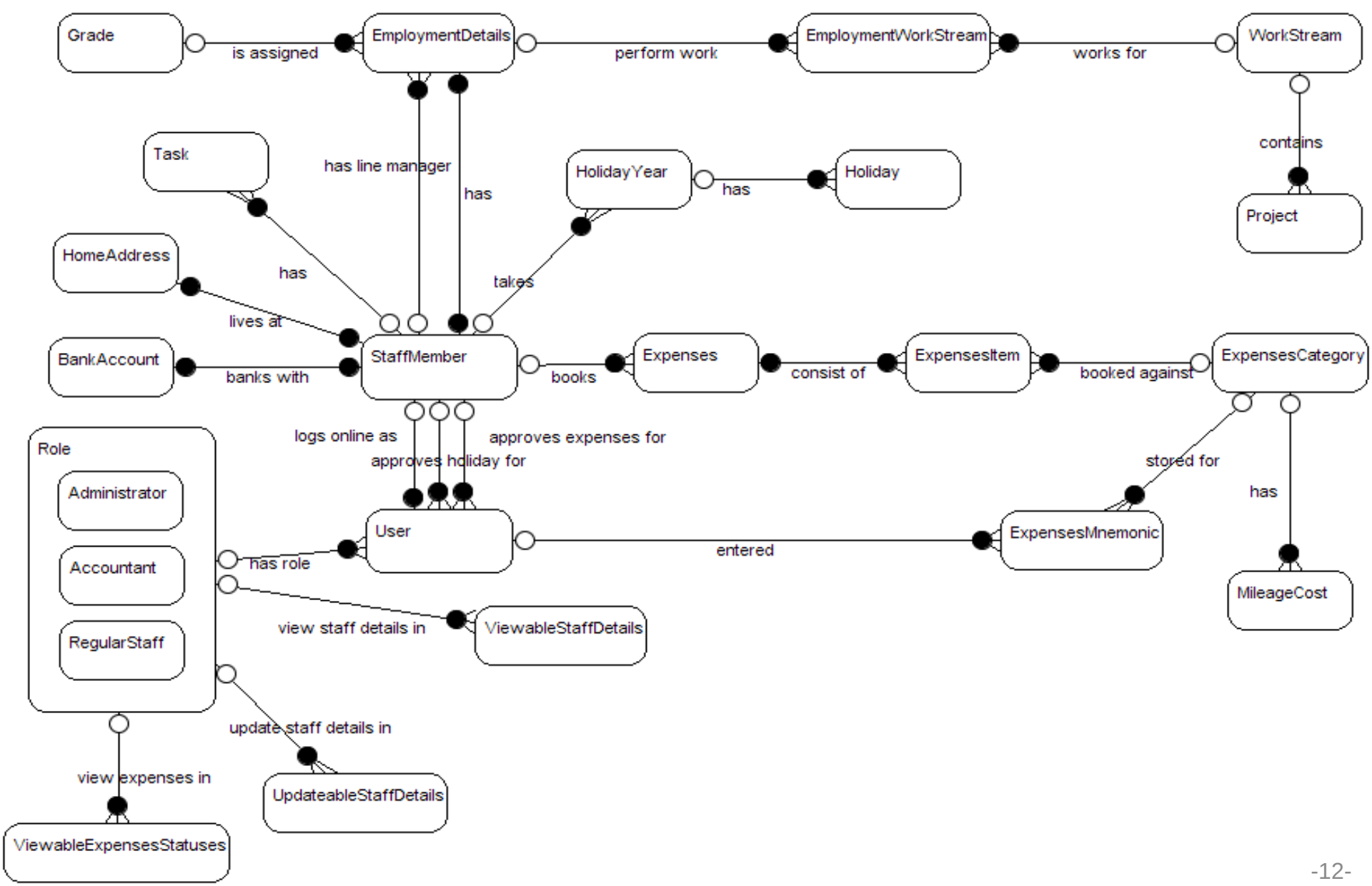
Variation in ER diagram standards



Variation in ER diagram standards



Variation in ER diagram standards

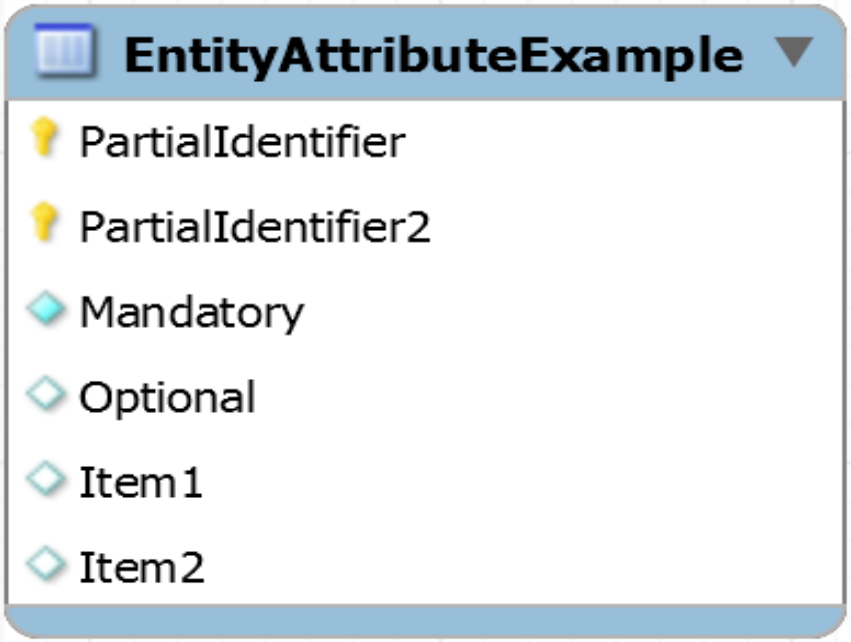


Representing entities and attributes

- Entity

Entity1	
PK	Identifier
	Ent1Attribute1 Ent1Attribute2

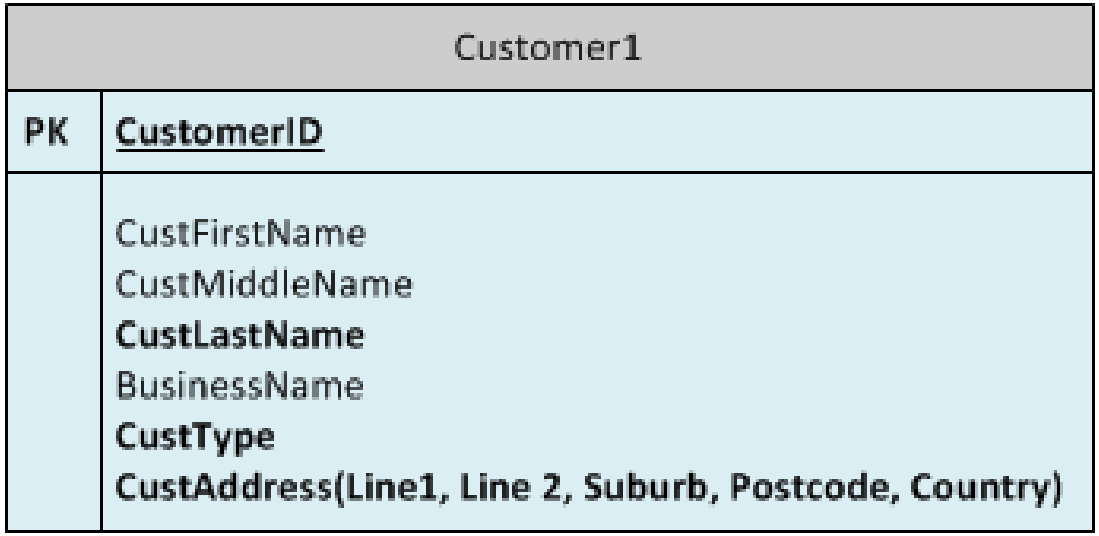
- Attributes



- Key (or Identifier)
  - Fully identifies an instance
- Partial Key
  - Partially identifies an instance
- Attributes
  - Mandatory
  - Optional
  - Derived
    - [YearsEmployed]
  - Multivalued
    - {Skill}
  - Composite
    - Name (First, Middle, Last)

Conceptual design for single entity

(drawn using Visio software)



- underline = primary key
- bold = not null
- () = composite attribute

Convert to *logical* design

Customer1	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType** **CustAddress(Line1, Line 2, Suburb, Postcode, Country)**

Customer1	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType** **CustAddLine1** **CustAddLine2** **CustSuburb** **CustPostcode** **CustCountry**

- Composite attributes become individual attributes
- Multi-valued attributes become a new table
- Resolve many-many relationships via a new table
- Add foreign keys at crows foot end of relationships

Convert to *physical* design

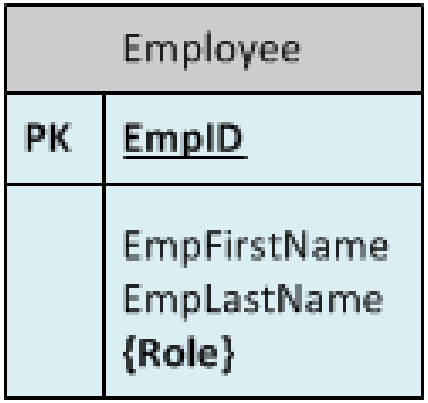
- Determine data types for each attribute
- Key, nullable

Customer1		
PK	**CustomerID**	**SMALLINT**
	CustFirstName	VARCHAR(100)
	CustMiddleName	VARCHAR(100)
	CustLastName	**VARCHAR(100)**
	BusinessName	VARCHAR(100)
	CustType	**CHAR(1)**
	CustAddLine1	**VARCHAR(100)**
	CustAddLine2	**VARCHAR(100)**
	CustSuburb	**VARCHAR(60)**
	CustPostcode	**CHAR(6)**
	CustCountry	**VARCHAR(60)**

Physical Name	Data Type	Req'd	PK	Notes
CustomerID	SMALLINT	☑	☑	CustomerID identifies Customer 1
CustFirstName	VARCHAR(100)	☐	☐	CustFirstName is of Customer 1
CustMiddleName	VARCHAR(100)	☐	☐	CustMiddleName is of Customer 1
CustLastName	VARCHAR(100)	☑	☐	CustLastName is of Customer 1
BusinessName	VARCHAR(100)	☐	☐	BusinessName is of Customer 1
CustType	CHAR(1)	☑	☐	NOTE: This will be implemented as an ENUM type in MySQL with
CustAddLine1	VARCHAR(100)	☑	☐	CustAddLine1 is of Customer 1
CustAddLine2	VARCHAR(100)	☑	☐	CustAddLine2 is of Customer 1
CustSuburb	VARCHAR(60)	☑	☐	CustSuburb is of Customer 1
CustPostcode	CHAR(6)	☑	☐	CustPostcode is of Customer 1
CustCountry	VARCHAR(60)	☑	☐	CustCountry is of Customer 1

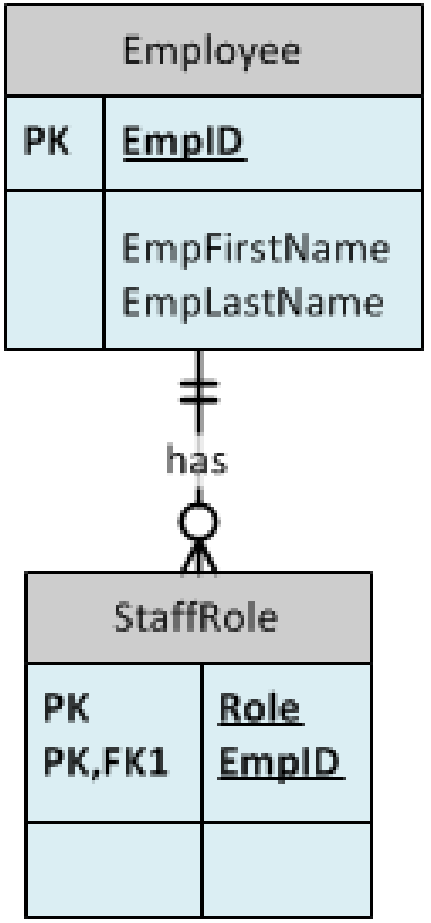
Dealing with multi-valued attributes

Conceptual Design



StaffRole is an example of a *weak entity*

Logical Design



Business Rules and Relationships

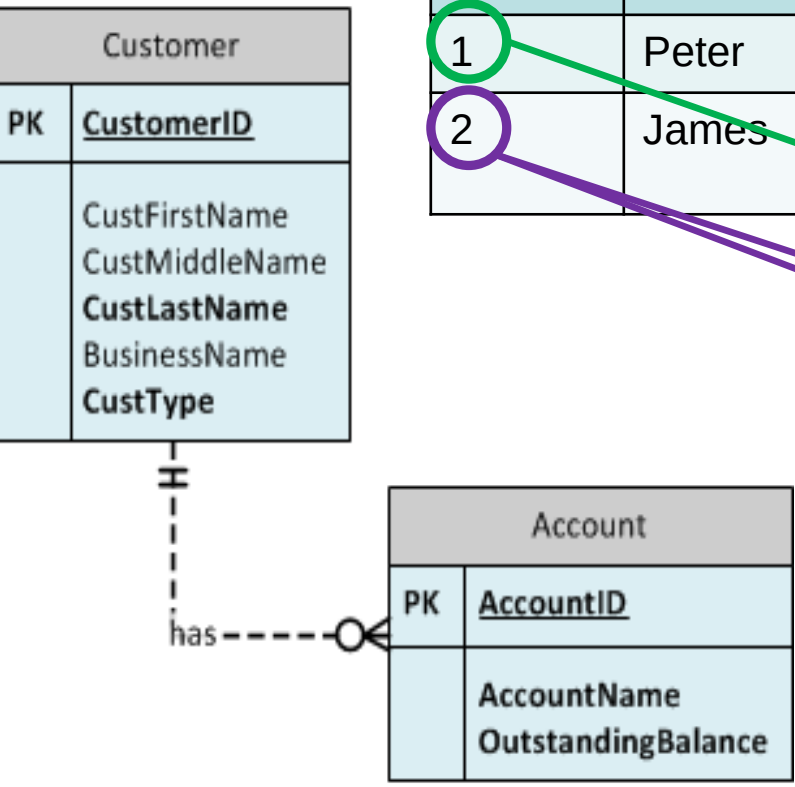
- Business rules are assertions that constrain entities
- Can impact structure and behaviour of the database
- Business rules can be assertions about attributes
  - "Quantity bought must be between 1 and 200." (assertion)
    - quantity bought (attribute)
- Or business rules can be assertions about entities
  - "A customer sets up at least one account." (assertion)
    - customer (entity)
    - account (entity)
- The latter kind of business rules are represented in our data models as *relationships between entities*.

Keys

- Keys or Identifiers are used to identify individual entity instances
  - Primary Key
    - (set of) columns, the values in which uniquely identify each instance
    - no column can be removed from the key without losing uniqueness
  - Candidate Key
    - the set of possible primary keys (choose one to be the PK)
  - Surrogate Key
    - system-assigned serial number (used if natural PK is unavailable or unsuitable)
  - Composite Key
    - a key which is made up of more than one attribute
      - e.g. for the entity “airline flight” we might use the composite key
        - » FlightNumber + FlightDate
  - Foreign Key
    - the key used to link to a primary key in another table
      - helps us to join tables in a Select statement
- Primary Keys are
  - unique
  - never null
  - do not change their value

Two entities with 1-M relationship

- Example: "A customer can set up several accounts."
  - The tables get linked through a foreign key

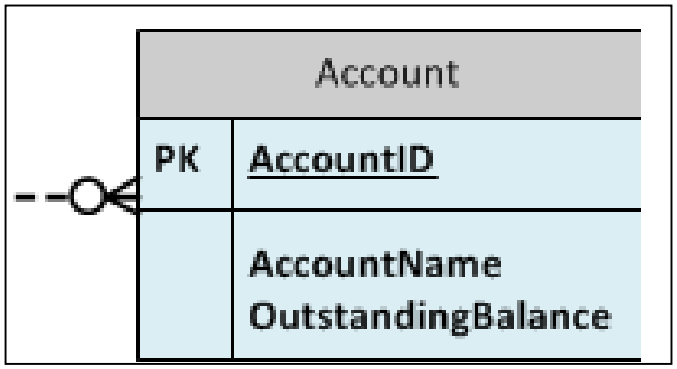


CustID	Customer FirstName	CustMiddle Name	CustLast Name	Business Name	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company

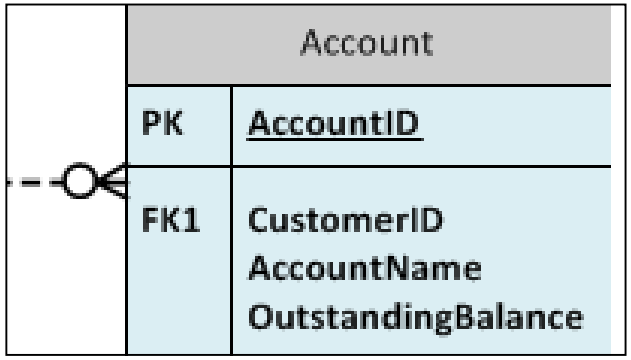
AccountID	AccountName	Outstanding Balance	CustID
01	Peter Smith	245.25	1
05	JJ Ent.	552.39	2
06	JJ Ent. Mgr	10.25	2

Conceptual to logical design – with FK

- Conceptual Design



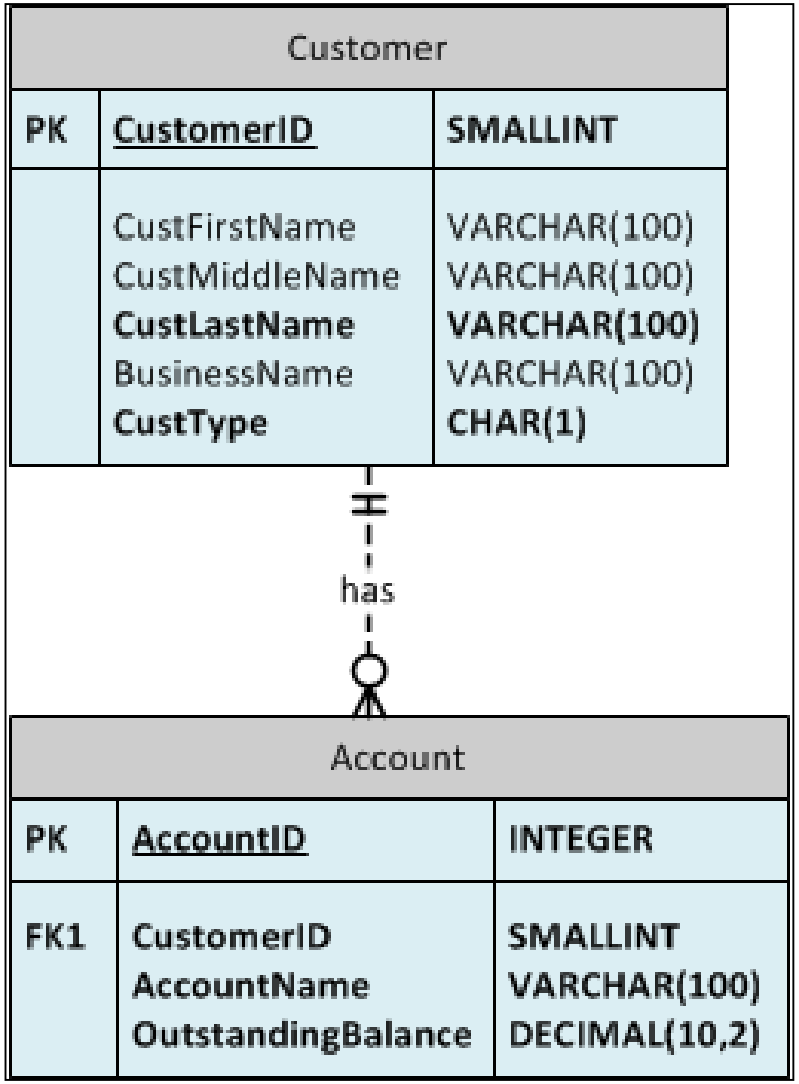
- Logical Design



– Add foreign keys at crows feet end of relationships
  - FK1 – CustomerID
    – This is the link to the customer table
    – Every CustomerID in Account must be present in Customer
      » Referential integrity

Physical design - with FK

- Attribute data types



FK must have the same data type as the PK it refers to.

Now we can create the tables

```
CREATE TABLE Customer (
   CustomerID         smallint                      auto_increment,
   CustFirstName      varchar(100),
   CustMiddleName     varchar(100),
   CustLastName       varchar(100)                  NOT NULL,
   BusinessName       varchar(200),
   CustType           enum('Personal','Company')    NOT NULL,
   PRIMARY KEY  (CustomerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Account (
   AccountID             smallint        auto_increment,
   AccountName           varchar(100)    NOT NULL,
   OutstandingBalance    DECIMAL(10,2)   NOT NULL,
   CustomerID            smallint        NOT NULL,
   PRIMARY KEY  (AccountID),
   FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
         ON DELETE RESTRICT
         ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Referential Actions how foreign keys guarantee referential integrity.

Insert - with FK

Current Database

CustID	CustomerFirstName	CustMiddle Name	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company

AccountID	AccountName	OutstandingBalance	CustID

Insert a row…

```
INSERT INTO ACCOUNT VALUES (DEFAULT, 'My New Account', 0, 5);
```

What happens?

Error Code: 1452. Cannot add or update a child row: a foreign key constraint fails (`db_seanbm/account`, CONSTRAINT `account_ibfk_1` FOREIGN KEY (`CustomerID`) REFERENCES `customer` (`CustomerID`))

Insert - with FK

- Run the Inserts…

```
INSERT INTO ACCOUNT VALUES (DEFAULT, 'Peter Smith', 245.25, 1);
INSERT INTO ACCOUNT VALUES (DEFAULT, 'JJ ENt.', 552.39, 2);
INSERT INTO ACCOUNT VALUES (DEFAULT, 'JJ ENt. Mgr', 10.25, 2);
```

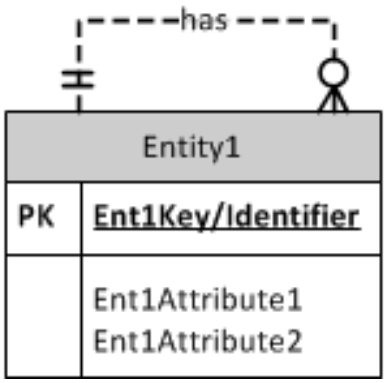
CustID	CustomerFirstName	CustMiddle Name	CustLastName	BusinessName	CustType
1	Peter		Smith		Personal
2	James		Jones	JJ Enterprises	Company
3	Akin		Smithies	Bay Wart	Company

AccountID	AccountName	OutstandingBalance	CustID
01	Peter Smith	245.25	1
02	JJ Ent.	552.39	2
03	JJ Ent. Mgr	10.25	2

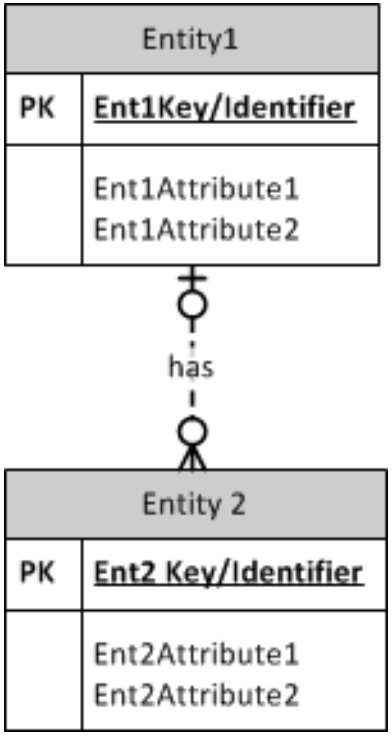
Relationship Degree

How many entities take part in the relationship?

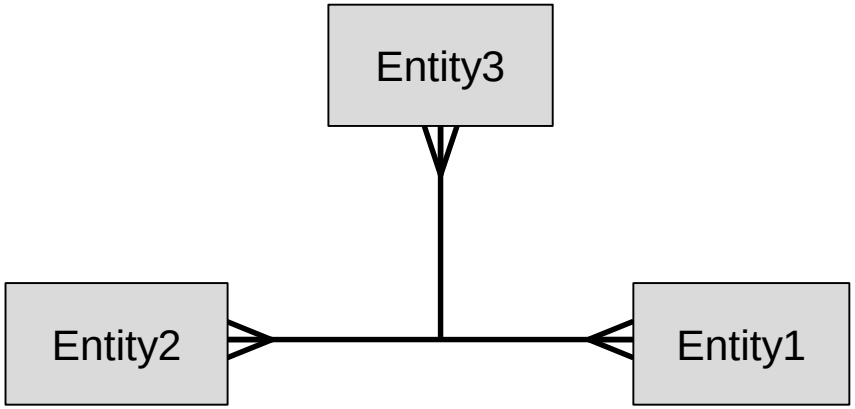
Unary (1)



Binary (2)

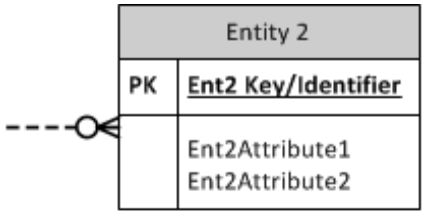


Ternary (3)

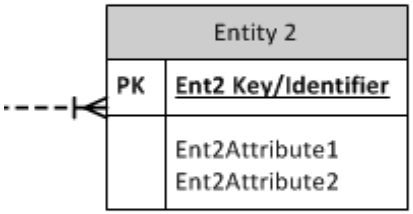


Relationship Cardinality

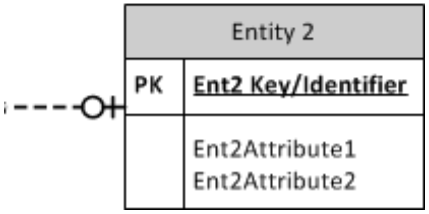
- One to One
  - Each entity in one set is related to 0 or 1 in the other.
- One to Many
  - Each entity in one set is related to many in the other.
- Many to Many
  - Each entity in either set can be related to many in the other set
  - These require an extra step to implement in a relational database.



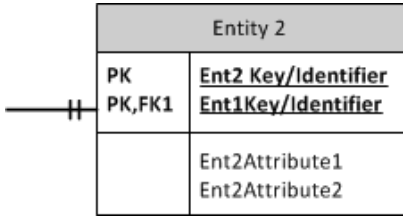
Optional Many



Mandatory Many



Optional One

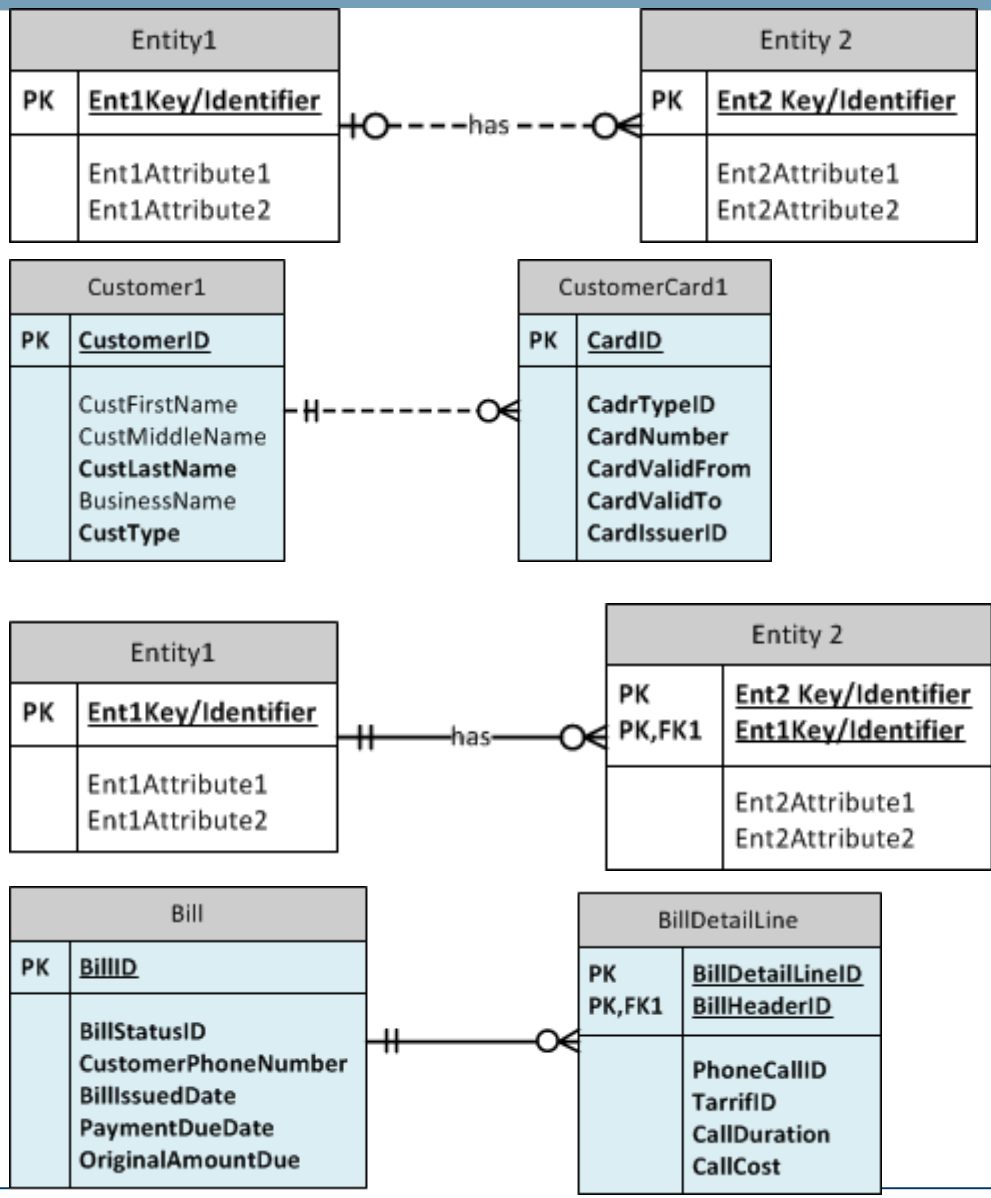


Mandatory One

Strong and Weak entities

- Strong Entity
  - entity 2's PK is independent of the PKs of other entities

Weak Entity
- entity 2's PK depends on (includes) the PK of entity 1



Structured Query Language (SQL)

What is SQL

- SQL – or SEQUEL – is a language used to create, access and maintain relational databases
- Based on relational algebra and relational calculus
- SQL (DML) supports CRUD (Create, Read, Update, Delete)
  - Insert, Select, Update, Delete commands
- You can see the latest SQL 2011 standard at
  - http://www.jtc1sc32.org/doc/N2151-2200/32N2153T-text_for_ballot-FDIS_9075-1.pdf
- SQL is a widely used standard and there are resources online:
  - http://en.wikipedia.org/wiki/SQL
  - http://en.wikipedia.org/wiki/Category:SQL_keywords
  - https://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html
  - https://www.w3schools.com/sql/

Brief history

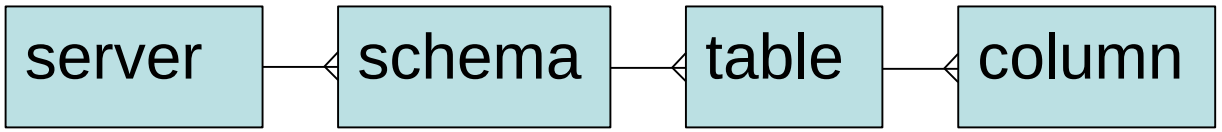
1974	IBM develops SEQUEL (renamed to SQL) based on Codd research
1979…	Oracle, IBM etc release RDBMS with SQL language
1986	1st SQL Standard (ANSI)
1989	2nd SQL Standard (ANSI) – includes referential integrity
1992	3rd SQL Standard (ISO) – most widely conformed to by vendors
1997…	dynamic websites enabled by SQL
1999	SQL-1999 – 4th SQL Standard (ISO) – Object support, recursion, procedures and flow control
2003	SQL-2003 – 5th SQL Standard (ISO) – XML support, auto number
2006	SQL-2006 – 6th SQL Standard (ISO) – Defines SQL use with XML
2008	SQL-2008 – 7th SQL Standard (ISO) – FETCH command added
2008	HTML 5 with SQLite built in
2011	SQL-2011 – 8th SQL Standard (ISO) – temporal databases

How we use SQL

- during Implementation of the database
  - Implement tables from physical design using DDL Create Table
- during Production
  - use Select commands to read the data from the tables
  - use DML Insert, Delete, Update commands to update data
  - use DML Alter, Drop commands to update the database structure

MySQL

- We are using the MySQL implementation of SQL
  - If you are using other DBMS (such as ORACLE or SQLServer) you will need to check their implementation of SQL.
    - differences can range from valid keywords to data types
- The university's MySQL server = version 5.7.9
- You can get the latest version of MySQL (5.7) from
    - http://dev.mysql.com/downloads/
    - Community edition = FOSS
  - Get syntax help for MySQL SQL statements at
    - http://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html
- Explore your server with these commands:
  - show schemas; (alternatively, 'show databases')
  - show tables;
  - describe table;



The SQL Language

- Consists of:
  - Data Definition Language (DDL)
    - to define and set up the database
    - CREATE, ALTER, DROP
      - also TRUNCATE, RENAME
  - Data Manipulation Language (DML)
    - to manipulate and read data in tables
    - SELECT, INSERT, DELETE, UPDATE
      - MySQL also provides others…. eg REPLACE
  - Data Control Language (DCL)
    - to control access to the database
      - GRANT, REVOKE
  - Other Commands
    - administer the database
    - transaction control

The SELECT Statement in detail

SELECT [ALL | DISTINCT] ***select_expr*** **[,** ***select_expr*** **...]**

List the columns (and expressions) that are returned from the query

[FROM ***table_references***

Indicate the table(s) or view(s) from where the data is obtained

[WHERE ***where_condition*****]**

Indicate the conditions on whether a particular row will be in the result

[GROUP BY {*col_name* **|** ***expr*** **} [ASC | DESC], ...]**

Indicate categorisation of results

[HAVING ***where_condition*****]**

Indicate the conditions under which a particular category (group) is included in the result

[ORDER BY {*col_name* **|** ***expr*** **|** ***position*****} [ASC | DESC], ...]**

Sort the result based on the criteria

[LIMIT {[*offset*,**] row_count | row_count OFFSET** ***offset*****}]**

Limit which rows are returned by their return order (ie 5 rows, 5 rows from row 2)

]

Select entire contents of table

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
1 •  SELECT * FROM Customer;
2
```

Edit: Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType
1	Peter	NULL	Smith	NULL	Personal
2	James	NULL	Jones	JJ Enterprises	Company
3	Akin	NULL	Smithies	Bay Wart	Company
4	Julie	Anne	Smythe	Konks	Company
5	Jen	NULL	Smart	BRU	Company
6	Lim	NULL	Lam	NULL	Personal
7	Kim	NULL	Unila	Saps	Company
8	James	Jay	Jones	JJ's	Company
9	Keith	NULL	Samson	NULL	Personal
NULL	NULL	NULL	NULL	NULL	NULL

Select specific columns

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName, CustFirstName
    FROM Customer;
```

Export: Autosize:

CustLastName	CustFirstName
Smith	Peter
Jones	James
Smithies	Akin
Smythe	Julie
Smart	Jen
Lam	Lim
Unila	Kim
Jones	James
Samson	Keith

WHERE clause: select specific rows

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName FROM Customer
    WHERE CustLastName = "Smith";
```

Export | Autosize

CustLastName
Smith

WHERE clause with LIKE

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName FROM Customer
    WHERE CustLastName LIKE "Sm%";
```

Export

CustLastName
Smith
Smithies
Smythe
Smart

Select with ORDER BY

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName;
```

Export:

CustLastName	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company
Unila	Company

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName DESC;
```

Export:

CustLastName	CustType
Unila	Company
Smythe	Company
Smithies	Company
Smith	Personal
Smart	Company
Samson	Personal
Lam	Personal
Jones	Company
Jones	Company

Select with LIMIT

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName
    LIMIT 5;
```

Export:

CustLastName	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company

```
SELECT CustLastName, CustType
    FROM Customer
    ORDER BY CustLastName
    LIMIT 5
    OFFSET 3;
```

Export:

CustLastName	CustType
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company

CustLastName	CustType
Jones	Company
Jones	Company
Lam	Personal
Samson	Personal
Smart	Company
Smith	Personal
Smithies	Company
Smythe	Company
Unila	Company

(what happens if we Limit without Ordering?)

Select with GROUP BY

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustType, Count(CustomerID)
    FROM Customer
    GROUP BY CustType;
```

Export: Autosize:

CustType	Count(CustomerID)
Personal	3
Company	6

```
SELECT CustType, Count(CustomerID) AS Count
    FROM Customer
    GROUP BY CustType;
```

Export: Autosize:

CustType	Count
Personal	3
Company	6

Select with WHERE and GROUP BY

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
SELECT CustType, Count(CustomerID)
    FROM Customer
    WHERE CustLastName LIKE "Sm%"
    GROUP BY CustType;
```

Export

CustType	Count(CustomerID)
Personal	1
Company	3

Select with GROUP BY and HAVING

Customer	
PK	**CustomerID**
	CustFirstName CustMiddleName **CustLastName** BusinessName **CustType**

```
1  select custtype, count(CustomerId) from customer
2  where custlastname like 'Sm%'
3  group by custtype
4  having count(CustomerId) = 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

custtype	count(CustomerId)
Company	3

"Having" works on groups
the way "Where" works on individual rows

Inner join, Natural join

- Data about an entity is spread across 2 tables – so join them
- Inner/Equi join - Join rows where FK value = PK value

```
SELECT * FROM Customer INNER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2

- Natural Join gives the same result as Inner Join
  - requires PK and FK columns to have the same name

```
SELECT * FROM Customer NATURAL JOIN Account;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25

Outer join

- Outer join
  - Can be left or right (see difference below)
  - Includes records from left/right table that don't have a matching row

```
SELECT * FROM Customer LEFT OUTER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleNa	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2
3	Akin	NULL	Smithies	Bay Wart	Company	NULL	NULL	NULL	NULL

```
SELECT * FROM Customer RIGHT OUTER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2

Cartesian join

- What if there is no join condition?

```
SELECT * FROM Customer, Account;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	1	Peter Smith	245.25	1
3	Akin	NULL	Smithies	Bay Wart	Company	1	Peter Smith	245.25	1
1	Peter	NULL	Smith	NULL	Personal	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
3	Akin	NULL	Smithies	Bay Wart	Company	2	JJ ENt.	552.39	2
1	Peter	NULL	Smith	NULL	Personal	3	JJ ENt. Mgr	10.25	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2
3	Akin	NULL	Smithies	Bay Wart	Company	3	JJ ENt. Mgr	10.25	2

NOT CORRECT: lack of join conditions -> Cartesian product

(every row in Customer combined with every record in Account)