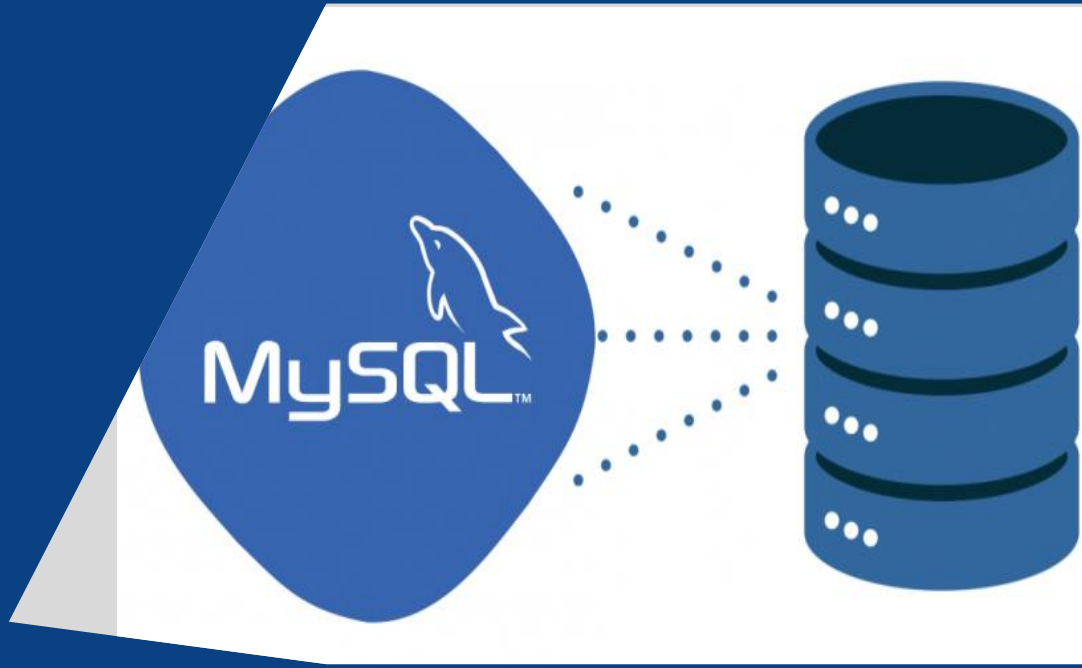
THE UNIVERSITY OF MELBOURNE

Physical Modelling. MySQL Data Types



Database Systems & Information Modelling
INFO90002

Week 4 – Physical Modelling

Dr Tanya Linden
David Eccles

Convert from Logical to Physical Design

Inputs

- Normalised relations
  - (Next topic)
- Attribute definitions
- Response time expectations
- Data security needs
- Backup / recovery needs
- Integrity expectations
- DBMS technology used

Leads to

Decisions

- Attribute data types
- Physical record descriptions (Oracle primary)
  - These don't always match the logical design
- File organisations
- Indexes and database architectures
- Query optimisation

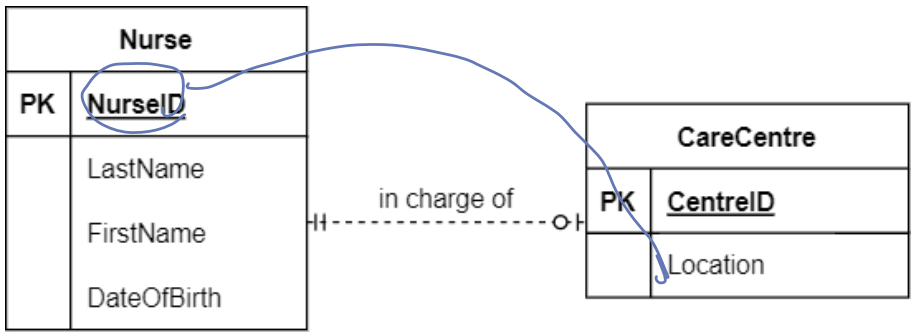
The physical record is **a group of fields stored in adjacent memory locations and retrieved together as a unit**.
The design of the physical record can also affect the speed of access to the record and the amount of disk space needed to store the record data.

One-to-One Relationship

In one to many

→ FK goes to many side

- Rule: Make the PK from the *one* side an FK on the other side
- But we have 2 “one” sides. Which one?



If you put the CentreID to nurse, there are a lot of null value.

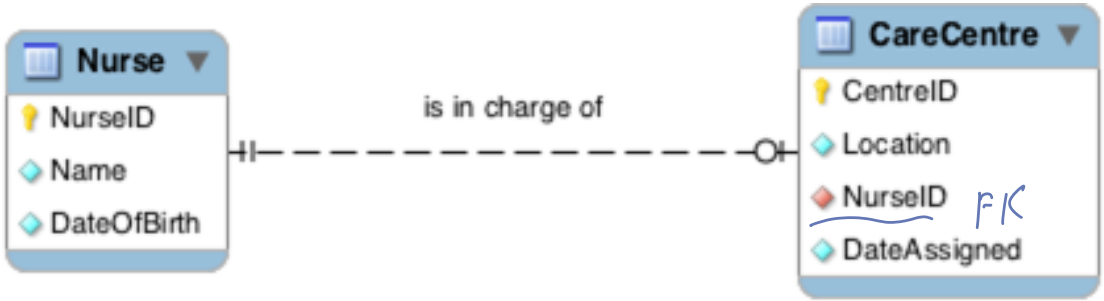
what will be

- Need to decide whether to put the foreign key inside Nurse or CareCentre (in which case you would have the Date_Assigned in the same location)
  - Where would the least NULL values be?
  - The rule is the OPTIONAL side of the relationship gets the foreign key

One to One Relationship – Logical and Physical Design

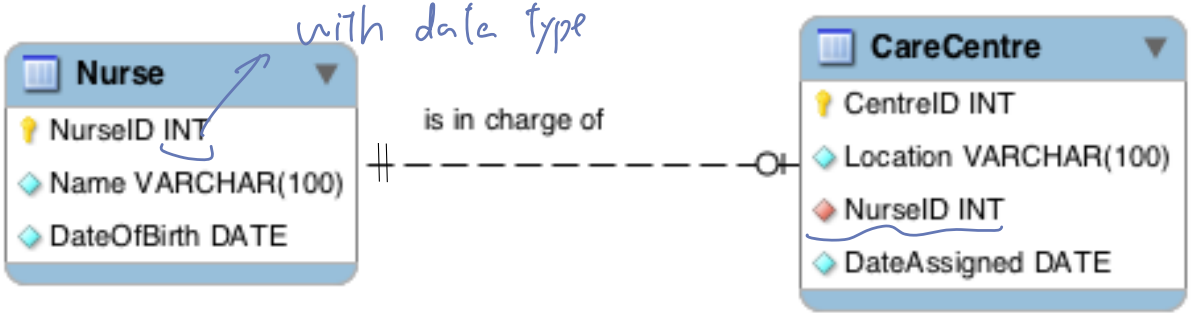
- **Logical**
  - Nurse (NurseID, Name, DateOfBirth)
  - CareCentre (CentreID, Location, *NurseID*, DateAssigned)

FK in Italic



- **Physical**

with data type



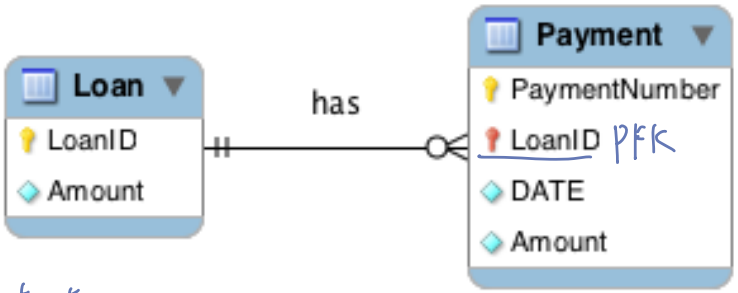
Summary of Binary Relationships

- **One-to-Many**
  - Primary key on the one side becomes a foreign key on the many
- **Many-to-Many**
  - Create an Associative Entity (a new relation) with the primary keys of the two entities it relates to as the combined primary key
- **One-to-One**
  - Need to decide where to put the foreign key
  - The primary key on the mandatory side becomes a foreign key on the optional side
  - If two optional or two mandatory, pick one arbitrarily

, we still need to consider after put the two fks into the new created associated entity, is primary keys unique? If not, add another attribute ↓ to make them primary keys.

Strong and Weak Entity - Identifying Relationship

- How to map an Identifying relationship
  - Map it the same way: Foreign Key goes into the relationship at the crow's foot end.
  - Only Difference is: The Foreign Key becomes **part of the Primary Key**



Without the diagramme, we can see by the text

  - Logical Design
    - Loan (LoanID, Amount)
    - Payment (PaymentNumber, *LoanID*, Date, Amount)
  - Physical Design – as per normal one-to-many

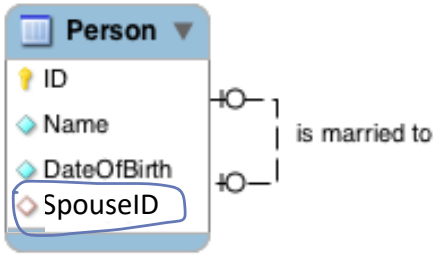
PFK

this one have underline and it's italic

In MySQL Workbench v8.0.x PFKs are displayed like this
They should be denoted with a RED KEY:
This is a cosmetic problem and there is a fix
published on the LMS for those who want the red key

Unary: One-to-One

Conceptual / Logical Design:



Logical Relation:

Person (ID, Name, DateOfBirth, *SpouseID*)

reference ID
FK

Implementation:

```
CREATE TABLE Person (
  ID INT NOT NULL,
  Name VARCHAR(15) NOT NULL,
  DateOfBirth DATE NOT NULL,
  SpouseID INT,
  PRIMARY KEY (ID),
  FOREIGN KEY (SpouseID) REFERENCES Person(ID)
  ON DELETE RESTRICT
  ON UPDATE CASCADE);
```

people can be not married

but it doesn't have not null ?

ID	Name	DOB	SpouseID
1	Ann	1969-06-12	3
2	Fred	1971-05-09	NULL
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	NULL

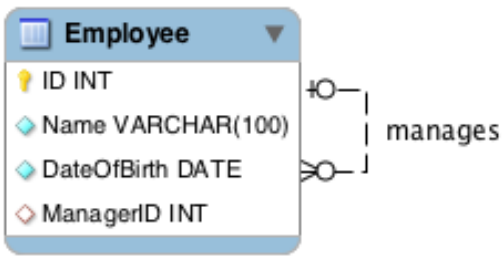
If Chon change ID, spouse ID from ID have to change

Unary: One-to-Many

Logical Relation:

Employee (<u>ID</u>, Name, DateOfBirth, *ManagerID*)

Physical Design:



Implementation:

```
CREATE TABLE Employee(
  ID INT NOT NULL,
  Name VARCHAR(12) NOT NULL,
  DateOfBirth DATE NOT NULL,
  ManagerID INT ,
  PRIMARY KEY (ID),
  FOREIGN KEY (ManagerID) REFERENCES Employee(ID)
  ON DELETE RESTRICT
  ON UPDATE CASCADE);
```

ID	Name	DateOfBirth	ManagerID
1	Ann	1969-06-12	NULL
2	Fred	1971-05-09	1
3	Chon	1982-02-10	1
4	Nancy	1991-01-01	1

doesn't have manager

Unary Relationships

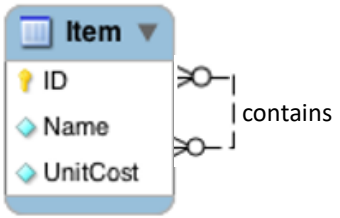
- A unary relationship is **when both participants in the relationship are the same entity**
- Operate in the same way as binary relationships
  - **One-to-One**
    - Put a Foreign key in the relation
  - **One-to-Many**
    - Put a Foreign key in the relation
  - **Many-to-Many**
    - Generate an Associative Entity
    - Put two Foreign keys in the Associative Entity
      - Need 2 different names for the Foreign keys
      - Both Foreign keys become the *combined* key of the Associative Entity

Unary: Many-to-Many

Scenario:

An Item is a composite of components

- An item contains components/parts
- is itself may be a component in many items



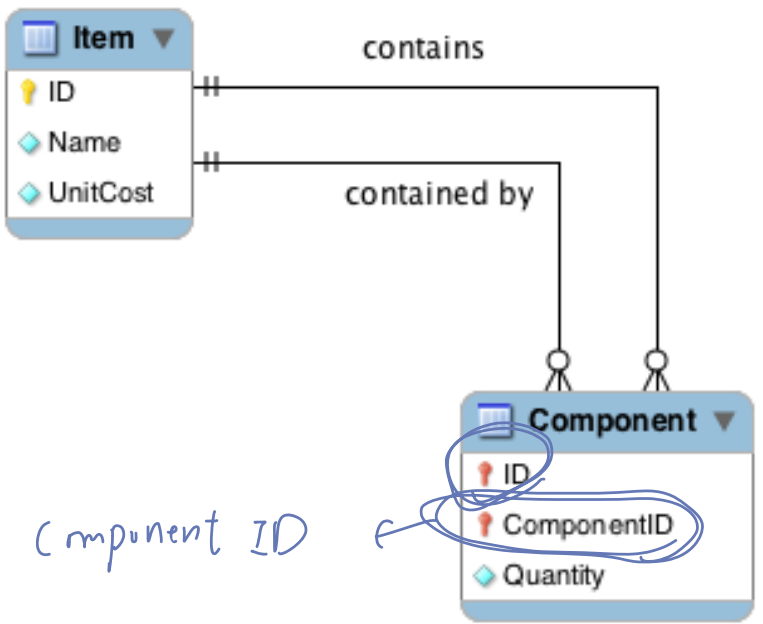
Logical Design:

- Create Associative Entity like usual
- Generate logical model

  Item (ID, Name, UnitCost)

  Component (*ID*, *ComponentID*, Quantity)

  PFK PFK



Unary: Many-to-Many Implementation

Implementation:

```
CREATE TABLE Item (
  ID         SMALLINT,
  Name       VARCHAR(100)    NOT NULL,
  UnitCost   DECIMAL(6,2)    NOT NULL,
  PRIMARY KEY (ID));
```

```
CREATE TABLE Component (
ID           SMALLINT,
ComponentID  SMALLINT,
Quantity     SMALLINT NOT NULL,
PRIMARY KEY   (ID, ComponentID),
FOREIGN KEY (ID) REFERENCES Item(ID)
  ON DELETE RESTRICT
  ON UPDATE CASCADE,
FOREIGN KEY (ComponentID) REFERENCES Item(ID)
  ON DELETE RESTRICT
  ON UPDATE CASCADE)
```

have two primary foreign keys.

THE UNIVERSITY OF MELBOURNE

MySQL Data Types

Data Types	Length
1. CHAR(SIZE)	1. 0 to 255 string
2. VARCHAR(SIZE)	2. 0 to 65535 string
3. BINARY(SIZE)	3. 1 byte
4. BLOB(SIZE)	4. 65535 bytes
A. TINYBLOB	a. 255 bytes
B. LONGBLOB	b. 4294967295
5. TEXT(SIZE)	5. 65635 bytes
A. TINYTEXT	a. 255 char
B. MEDIUMTEXT	b. 16777215
C. LONGTEXT	c. 4294967
6. ENUM(VALUE,VALUE....)	6. 65535 va
7. SET(VALUE,VALUE...)	7. 64 valu

Data Types

Choosing data types

Column: smallest unit of data in database

Data types help DBMS to store and use data efficiently

choose correct datatype.

You should choose data types that:

- enforce data integrity (quality)
- can represent all possible values
- support all required data manipulations
- minimise storage space
- maximise performance (e.g. fixed or variable length)

The major data types categories are:

- Text/string or character
- Number
  - Integer
  - Decimal
  - Float, Double
- Date and time

CPU works faster in interger

post code → use character (use 4 bit) and don't manipulate

Character types (MySQL)

CHAR(M): A fixed-length string that is always right-padded with spaces to the specified length when stored on disc.
The range of M is 1 to 255.

CHAR: Synonym for CHAR(1).

VARCHAR(M): A variable-length string.
Only the characters inserted are stored – no padding.
The range of M is 1 to 65535 characters.

BLOB, TEXT: A binary or text object with a maximum length of 65535 (2^16) bytes (blob) or characters (text).
Not stored inline with row data.

LONGBLOB, LONGTEXT: A BLOB or TEXT column with a maximum length of 4,294,967,295 (2^32 - 1) characters.

ENUM ('value1','value2',…) up to 65,535 members.

Number types (MySQL)

Integers

- **TINYINT**: Signed (-128 to 127), Unsigned (0 to 255)
- **SMALLINT**: Signed (-32,768 to 32,767), Unsigned (0 to 65,535 – 2^16 or 64k)
- **MEDIUMINT**: Signed (-8388608 to 8388607), Unsigned (0 to 16777215 –16M)
- **INT / INTEGER**:
  Signed (-2,147,483,648 to 2,147,483,647),
  Unsigned (0 to 4,294,967,295 – 2^32 or 4G)
- **BIGINT**:
  Signed (-9223372036854775808 to 9223372036854775807),
  Unsigned (0 to 18,446,744,073,709,551,615 - 2^64)

- **BIT**: stores bit values (representation using 0s and 1s), e.g. `b'111'` represents 7
  BIT(M) enables storage of M-bit values. M can range from 1 to 64.
- **Don't** use the "(M)" number for integers.

Number types (MySQL)

Decimal(65,30)

Real numbers (fractions)

- **FLOAT**: single-precision floating point, allowable values: -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.

- **DOUBLE / REAL**:

  double-precision, allowable values: -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
- optional M = number of digits stored, D = number of decimals.
- Float and Double are often used for scientific data.

- **DECIMAL**[(M[,D])]: fixed-point type. Good for money values.
- M = precision (total number of digits stored), D = number of decimals (within M)

Boolean values – not directly supported in MySQL and represented as **TINYINT**. If an attribute is a Boolean type, MySQL outputs data as 1 for true or 0 for false

Date Time types

DATE	used for values with a date part but no time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format; 1000-01-01 to 9999-12-31
TIME	retrieves and displays TIME values in 'hh:mm:ss' format; -838:59:59 to 838:59:59 (time of day or elapsed time)
DATETIME	used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD hh:mm:ss' format; 1000-01-01 00:00:00 to 9999-12-31 23:59:59 Stored in local time
TIMESTAMP	1970-01-01 00:00:00 - '2038-01-19 03:14:07' used for values that contain both date and time parts. Stored in UTC, converted to local when retrieved, which allows to use a different time zone when you connect to MySQL Server.
YEAR	1901 to 2155

By default, the current time zone for each connection is the server's time.

If the time zone setting remains constant, you get back the same value you store.

If you store a TIMESTAMP value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored.