TAMPERE UNIVERSITY OF TECHNOLOGY

ASE-9476 FIS: Databases and SQL

Outline

- Databases introduction
  - ✓ FIS and databases
  - ✓ Terms and concepts
  - ✓ Hardware and software
  - ✓ Users
  - ✓ General architecture
  - ✓ Benefits and limitations
- Database models
  - ✓ Hierarchical
  - ✓ Network
  - ✓ Object-oriented
  - ✓ Relational
- Example

FIS and databases

- Databases (DBs) is the primary way of sharing and exchanging information in FIS;
- DBs provide reliable mechanism for sharing large volumes of data and for integrating data from different sources
- Database management systems (DBMS) provide sophisticated capabilities to store, retrieve, and manage data

FIS shared information

- Material stocks
- Production inventory
- Work in progress
- Process results history and quality data
- Product data
- Process data
- Orders and schedules
- Shipping information
- Accounting information
- Customer information
- Supplier information

Databases terms and concepts

- Database system - a computer based system to record and maintain information.
- Information can be anything of significance to the database users.
- A database system has four major components:
  - ✓ data - information held in an integrated, shared database
  - ✓ hardware
  - ✓ software
  - ✓ users

Hardware and software

- Hardware
  - ✓ Consists of secondary storage on which the data lies.
  - ✓ Also consists of a processor, control units, etc.
  - ✓ the data is assumed to be too big to be held completely in the processor's memory.
- Software
  - ✓ The DBMS (database management system) software allows one or many persons to access the data.
  - ✓ allows the user to deal with data in an abstract (logical) way.

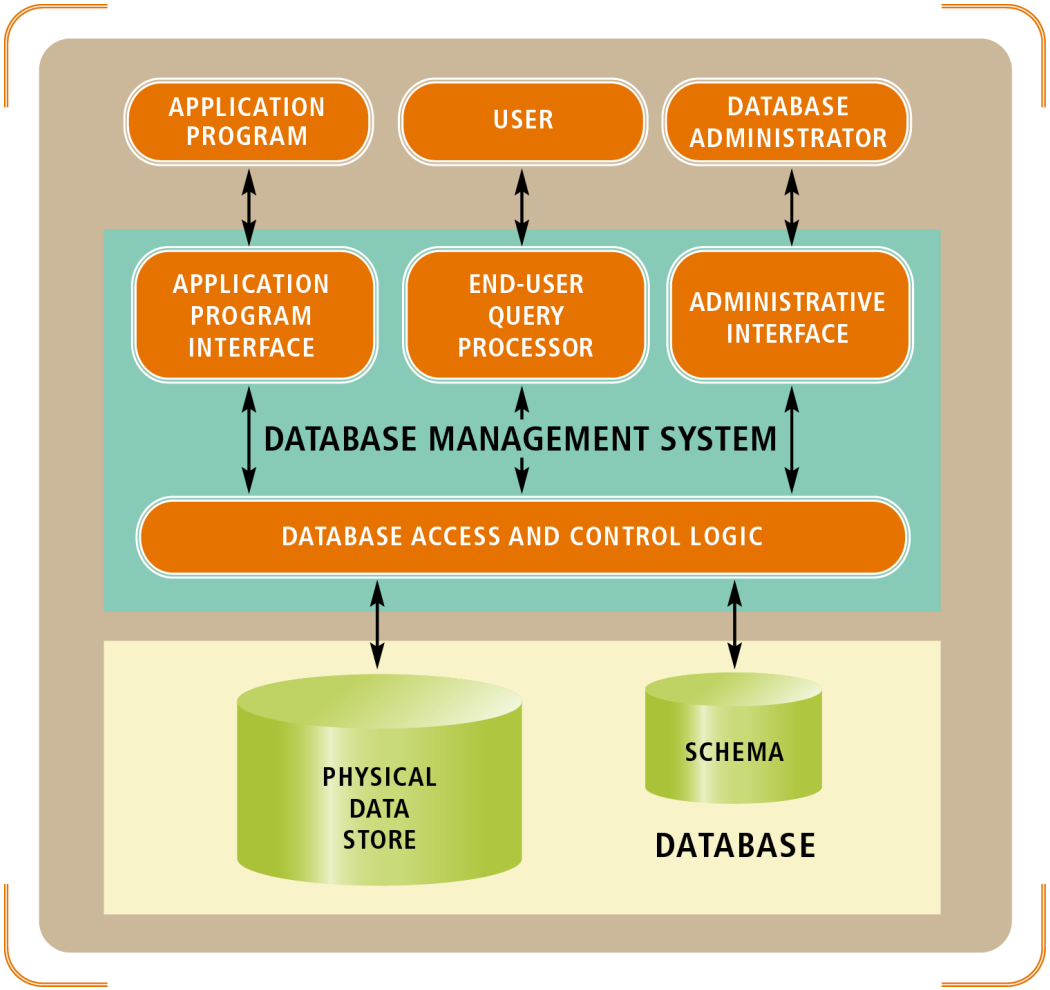
DBMS

- The database management system (DBMS) is the software that:
  - ✓ handles all access to the database
  - ✓ is responsible for applying the authorisation checks and validation procedures
- Conceptually what happens is:
  - ✓ A user issues an access request, using some particular DML.
  - ✓ The DBMS intercepts the request and interprets it.
  - ✓ The DBMS inspects in turn the external schema, the external/conceptual mapping, the conceptual schema, the conceptual internal mapping, and the storage structure definition.
  - ✓ The DBMS performs the necessary operations on the stored database.

Users

- There are three broad classes of user:
  - ✓ the application programmer, responsible for writing programs in some high-level language such as COBOL, C++, etc.
  - ✓ the end-user, who accesses the database via a query language
  - ✓ the database administrator (DBA), who controls all operations on the database

Database architecture



Benefits and limitations of databases (Relational)

Benefits

- Integration of heterogeneous systems without compatibility issues
- Standard SQL interface improves reusability and interoperability of systems
- Data is synchronized and up to date in all systems
- System integration simplified to defining common database formats

Limitations

- Expensive
- Potential bottleneck
- No time synchronization between writing and reading
- Unsafe, data can be read by the wrong system
- Difficult to agree on a table format
- Difficult to extent a table with new fields
- SQL commands can be large and difficult to manage

Database models

- Database models have evolved since their introduction in the 1960s
  - ✓ Hierarchical
  - ✓ Network
  - ✓ Object-oriented
  - ✓ Relational
- Most existing and newly developed systems use the relational or object-oriented approach

Hierarchical model

- Data is organized in a tree-like structure
- Data is stored as records, which are connected to one another through links
- Advantages: Hierarchical database can be accessed and updated rapidly
- Disadvantages:
  - child in the tree may have only one parent
  - relationships or linkages between children are not permitted
  - for retrieving data the whole tree has to be traversed

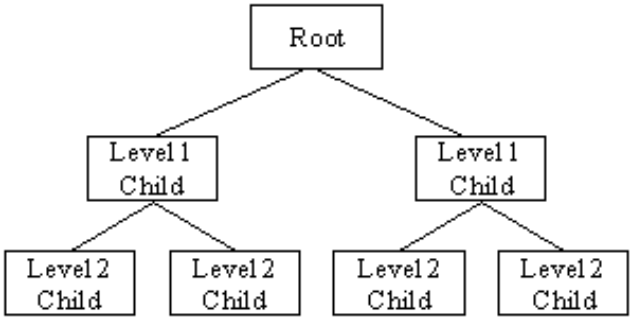
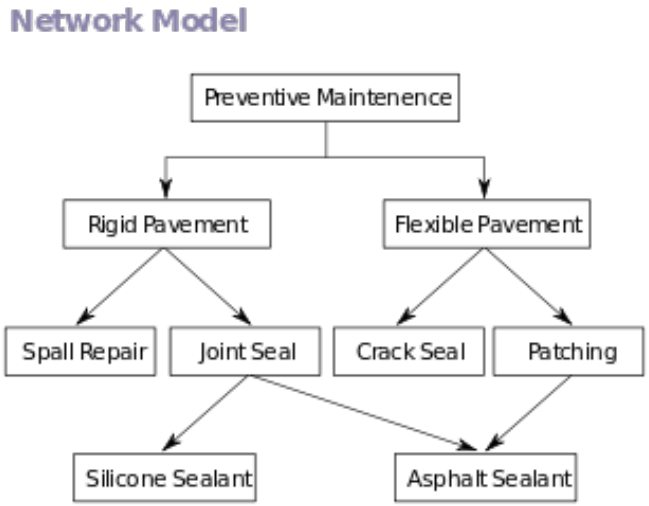


Image source: http://www.c-sharpcorner.com/UploadFile/65fc13/types-of-database-management-systems/

Network model

- The schema is a generalized graph of record types connected by relationship types
- The database itself is a generalized graph of record occurrences connected by relationships
- Advantages: allows more natural modelling of relationships between entities
- Disadvantages: relational models are still more flexible and network model advantage in speed was compensated by advances of hardware components



Object-oriented databases

- Introduction
- Design
- Attributes and relationship declarations
  - ✓ One-to-many relationship
  - ✓ Many-to-many relationship
  - ✓ Generalization

Object-oriented databases

- Object Database Management System (ODBMS)
  - ✓ Stores data as objects
  - ✓ Prototypes introduced in 1980s
  - ✓ Technology of choice for newly designed systems
    - In a way quite tightly connected to applications and might not be that easy to design for shared databases and legacy applications
- Object Definition Language (ODL)
  - ✓ Standard developed by the ODMG for defining the structure and content of an object database
- Java Data Objects (JDO)
  - ✓ Java-specific object database standard

Designing object databases

- Determine which classes require persistent storage
- Represent persistent classes
- Represent associations among persistent classes
- Choose appropriate data types and value restrictions (if necessary) for each field

Attribute and relationship declarations

- Attributes are (usually) elements with a type that does not involve classes.

```
attribute <type> <name>;
```

- Relationships connect an object to one or more other objects of one class.

```
relationship <type> <name> inverse
<relationship>;
```

Representing classes

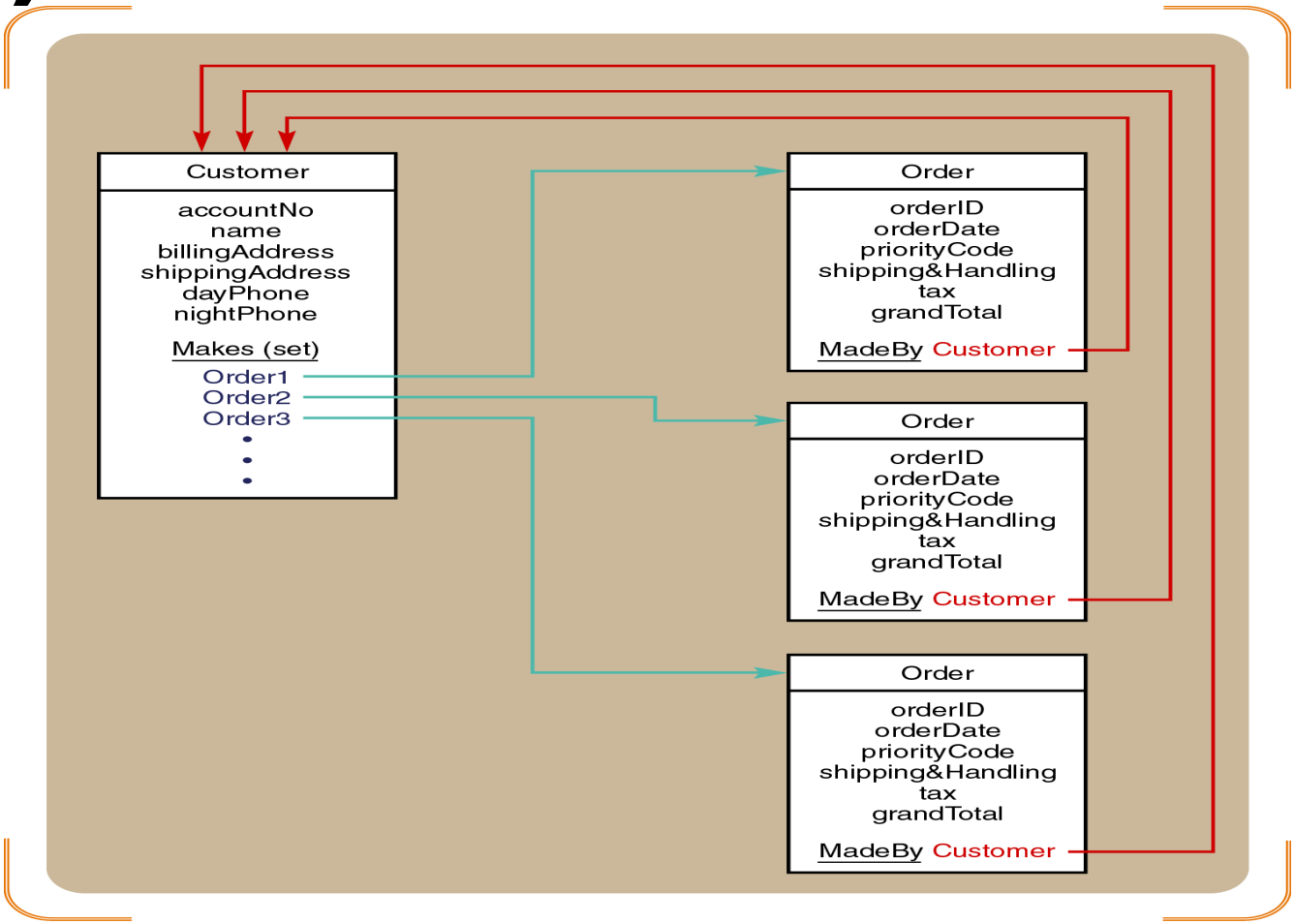
- An object database schema requires a class definition for each persistent class
  - ✓ ODL class definitions derive from the domain model class diagram

```
Class Customer {
    attribute string accountNo
    attribute string name
    attribute string billingAddress
    attribute string shippingAddress
    attribute string dayTelephoneNumber
    attribute string nightTelephoneNumber
}
```

Representing relationships

- ODBMSs represent associations by storing the object identifier of one object within related objects
  - ✓ Use attributes with object identifiers to find related objects (called navigation)
- Designers represent associations indirectly by declaring associations between objects
  - ✓ Use keywords “relationship” and “inverse”
  - ✓ ODBMS automatically creates object identifiers for declared associations

One-to-many relationship (1/2)



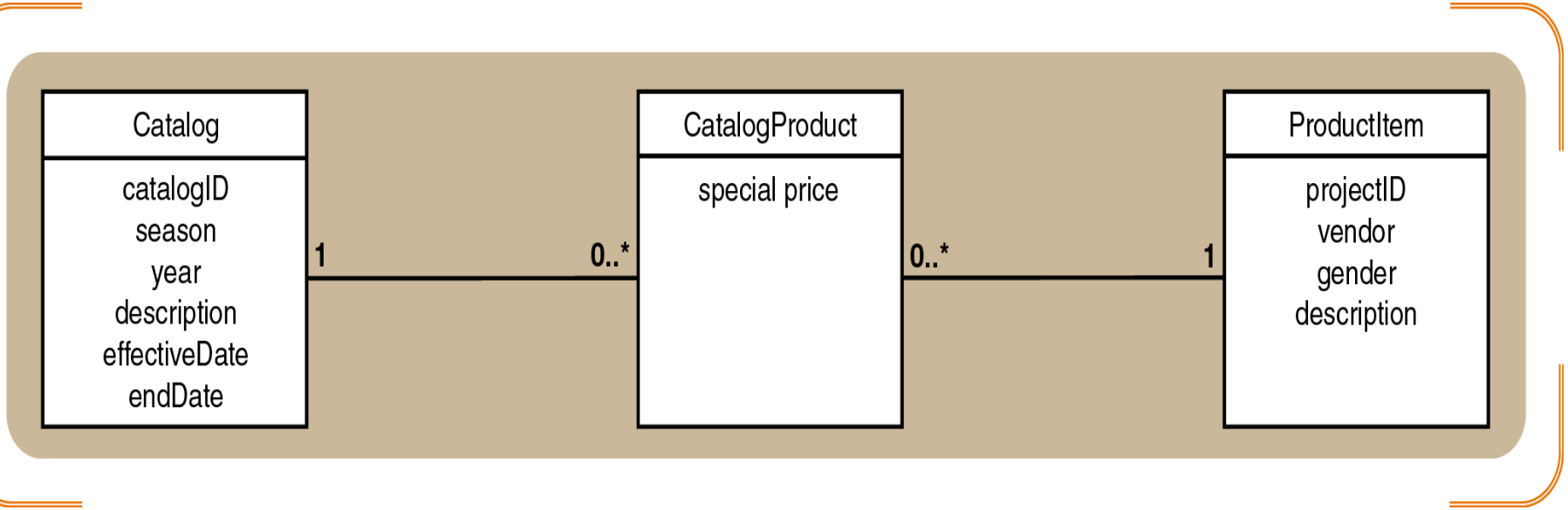
- A one-to-many association represented with attributes containing object identifiers

One-to-many relationship (2/2)

- Partial ODL class definition

```
Class Customer {
    attribute string accountNo
    …
    relationship set<Order> Makes
      inverse Order::MadeBy
}
Class Order {
    attribute string orderID
    …
    relationship Customer MadeBy
      inverse Customer::Makes
}
```

Many-to-many relationship (1/2)



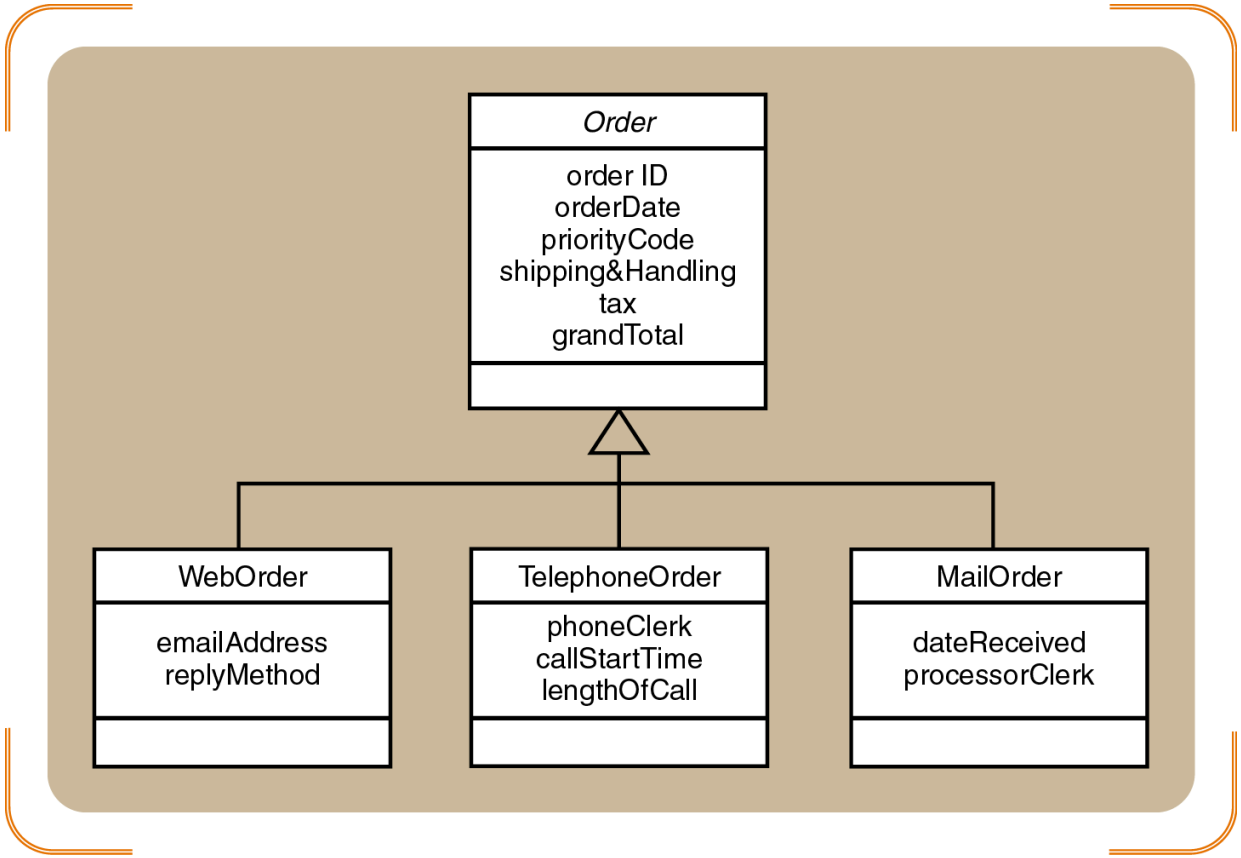
- A many-to-many association represented with two one-to-many associations

Many-to-many relationship (2/2)

- ODL uses multivalued attributes and an association class

```
Class Catalog {
    …
    relationship set<CatalogProduct> Contains1
      inverse CatalogProduct::AppearsIn1
}
Class ProductItem {
    …
    relationship set<CatalogProduct> Contains2
      inverse CatalogProduct::AppearsIn2
}
Class CatalogProduct {
    attribute real specialPrice
    …
    relationship Catalog AppearsIn1
      inverse Catalog::Contains1
    relationship ProductItem AppearsIn2
      inverse ProductItem::Contains2
}
```

Generalization associations (1/2)



- A generalization hierarchy

Generalization associations (2/2)

- ODL uses keyword extends

```
Class Order {
    attribute string orderID
    …
}
Class WebOrder extends Order {
    attribute string emailAddress
    …
}
Class TelephoneOrder extends Order {
    attribute string phoneClerk
    …
}
…
```

Object-oriented databases

- Strengths
  - ✓ rich type system
  - ✓ better at modelling complex objects
  - ✓ better performance on certain data structures
  - ✓ no impedance mismatch
    - Database becomes ”persistence extension”
- Weaknesses
  - ✓ procedural navigation
  - ✓ querying breaks encapsulation (or is limited)
  - ✓ no mathematical foundation

Relational databases

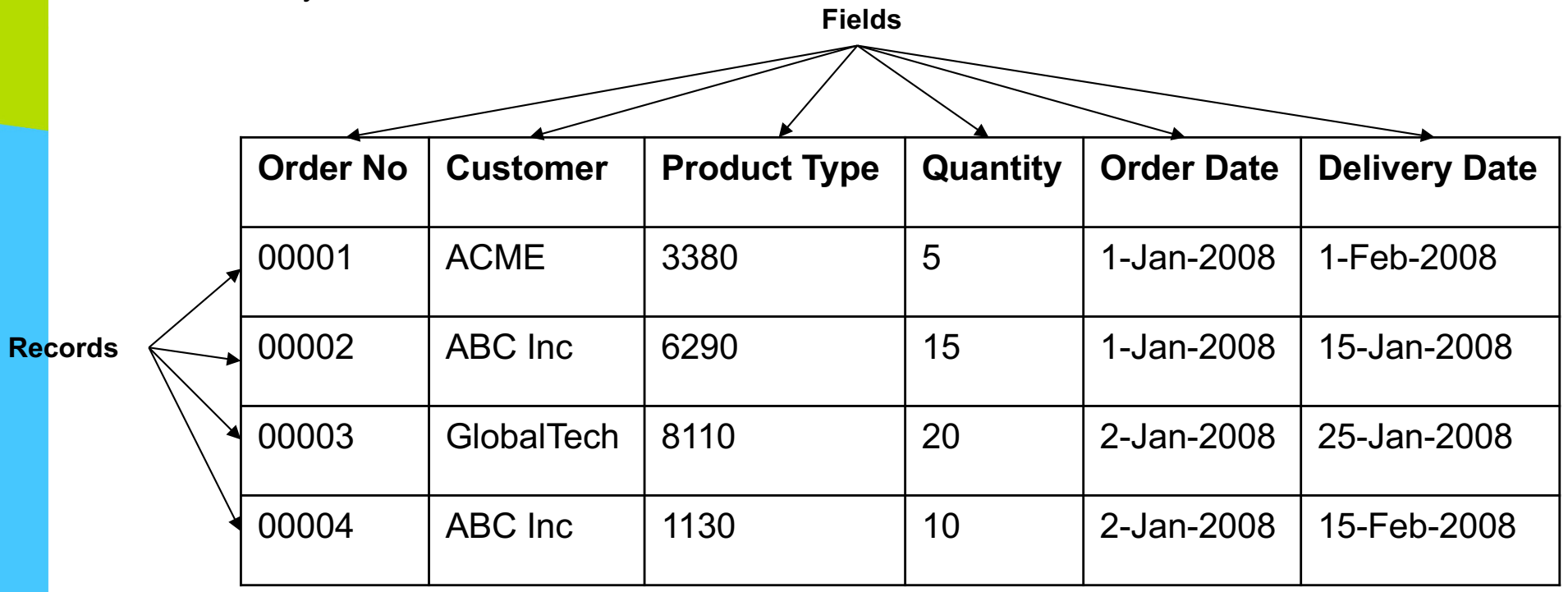
- Introduction
- SQL with examples

Relational model

- Collection of tables (also called relations)
- Table:
  - ✓ Collection of rows (also called tuples or records).
  - ✓ Each row in a table contains a set of columns (also called fields or attributes).
  - ✓ Each column has a type:
    - String: VARCHAR(20)
    - Integer: INTEGER
    - Floating-point: FLOAT, DOUBLE
    - Date/time DATE, TIME, DATETIME
    - Several others...
  - ✓ Primary key: A primary key uniquely identifies a row in a table.
  - ✓ A Foreign key duplicates the primary key in another table.
  - ✓ Keys may be natural or invented
- Schema: the structure of the database, including for each table
  - ✓ The table name
  - ✓ The names and types of its columns
  - ✓ Various optional additional information (constraints, etc.)

Table example

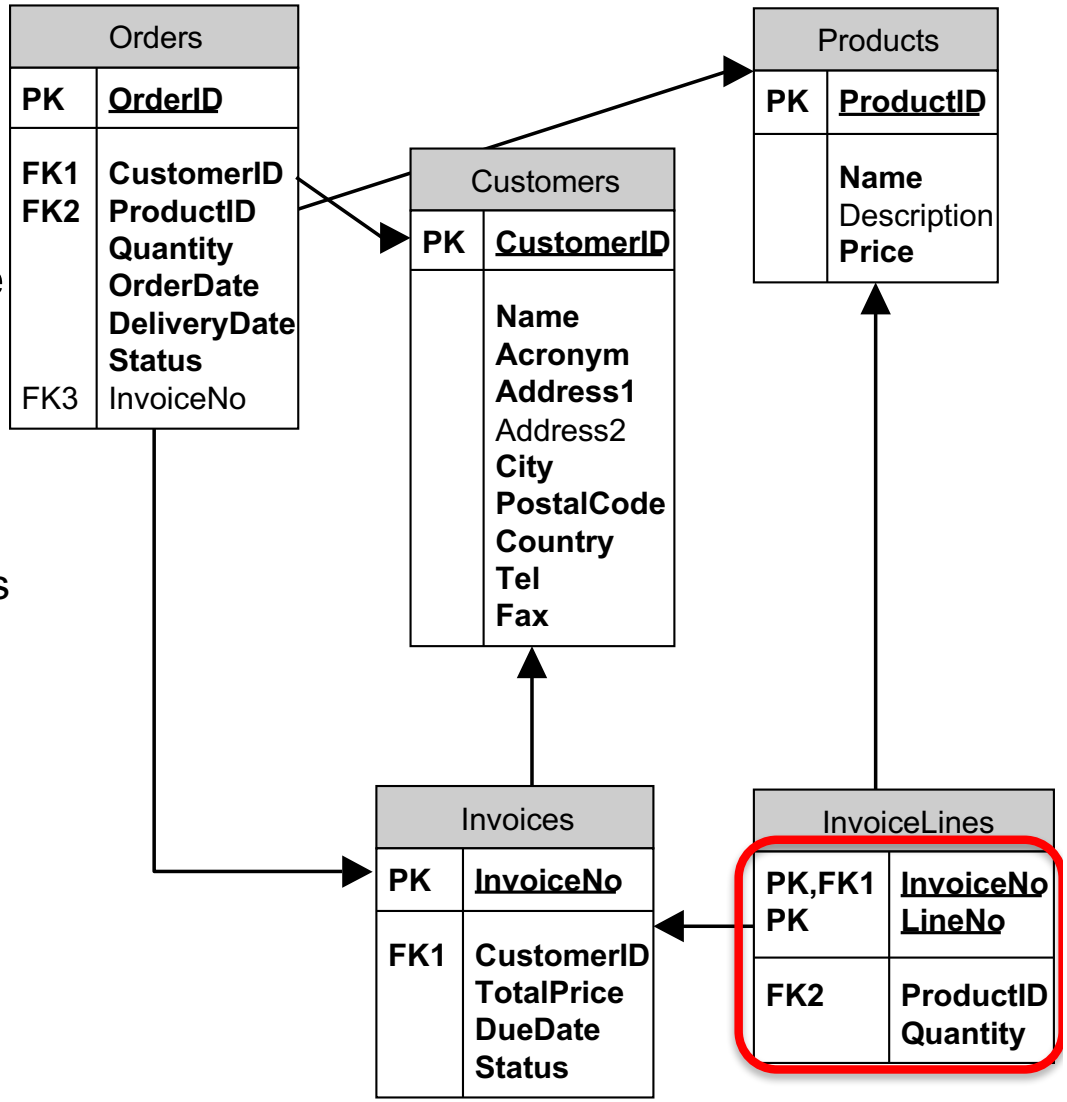
- Data organized in Tables.
- Each tables has a structure, composed of a number of fields.
- Each entry in a table is called a record.



Order No	Customer	Product Type	Quantity	Order Date	Delivery Date
00001	ACME	3380	5	1-Jan-2008	1-Feb-2008
00002	ABC Inc	6290	15	1-Jan-2008	15-Jan-2008
00003	GlobalTech	8110	20	2-Jan-2008	25-Jan-2008
00004	ABC Inc	1130	10	2-Jan-2008	15-Feb-2008

Relational model example

- Relational Databases establish links between tables
- The fields used to link tables are called keys.
- Primary (PK) and Foreign Keys (FK):
  - Link from Orders table to Products table through ProductID field (the key)
  - Link Orders table to Customers table through CustomerID field (the key)
  - Advantage: Info for each product/customers exists only once in the DB



Querying DB with Structured Query Language (SQL)

- SQL is a language/protocol used to add, modify and retrieve data from a relational database
- Most common commands:
  - ✓ SELECT
  - ✓ INSERT
  - ✓ UPDATE
  - ✓ DELETE

SQL SELECT

```
select [distinct] <column(s)>
from <table>
[ where <condition> ]
[ order by <column(s) [asc|desc]> ]
```

- Basic SELECT structure:
  - ✓ Return the selected fields for all records in the database

```
SELECT [field names] FROM [table name]
```

- Example:

```
SELECT OrderID, CustomerID, InvoiceNo FROM Orders
```

Orders							
OrderID	**CustomerID**	**ProductID**	**Quantity**	**Order Date**	**Delivery Date**	**Status**	**InvoiceNo**
00001	010203	3380	5	2008-01-01	2008-02-01	IN PROGRESS	-
00002	020406	6290	15	2008-01-01	2008-01-15	DELIVERED	001-02345
00003	030609	8110	20	2008-01-02	2008-01-25	COMPLETE	001-02346

Query Result:

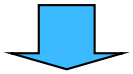
OrderID	CustomerID	InvoiceNo
00001	010203	-
00002	020406	001-02345
00003	030609	001-02346

SQL SELECT with WHERE clause

- The WHERE clause limits the number of records returned
- Example:

```
SELECT OrderID, CustomerID, InvoiceNo
FROM Orders
WHERE ProductID="6290"
```

Orders							
OrderID	**CustomerID**	**ProductID**	**Quantity**	**Order Date**	**Delivery Date**	**Status**	**InvoiceNo**
00001	010203	3380	5	2008-01-01	2008-02-01	IN PROGRESS	-
00002	020406	6290	15	2008-01-01	2008-01-15	DELIVERED	001-02345
00003	030609	8110	20	2008-01-02	2008-01-25	COMPLETE	001-02346



Query Result:

OrderID	CustomerID	InvoiceNo
00002	020406	001-02345

SQL SELECT from multiple tables

- SELECT can be used to collect related data from several tables

```
SELECT [table name].[field name] FROM [table name]
```

- Example:

```
SELECT Customers.Name,Invoices.TotalPrice
FROM Orders,Customers,Invoices
WHERE (Orders.ProductID=6290)
AND (Orders.CustomerID=Customers.CustomerID)
AND (Orders.InvoiceNo=Invoices.InvoiceNo)
```

SQL SELECT from multiple tables

```
SELECT Customers.Name,Invoices.TotalPrice
FROM Orders,Customers,Invoices
WHERE (Orders.ProductID=6290)
AND (Orders.CustomerID=Customers.CustomerID)
AND (Orders.InvoiceNo=Invoices.InvoiceNo)
```

Orders

OrderID	CustomerID	ProductID	Quantity	Order Date	Delivery Date	Status	InvoiceNo
00001	010203	3380	5	2008-01-01	2008-02-01	IN PROGRESS	-
00002	020406	6290	15	2008-01-01	2008-01-15	DELIVERED	001-02345
00003	030609	8110	20	2008-01-02	2008-01-25	COMPLETE	001-02346

Customers

CustomerID	Name	Acronym	Address1	Address2	…
010203	ACME	ACM	ACME Ave 12	-	…
020406	ABC Inc	ABC	123 ABC Road	North Side	…
030609	GlobalTech Corp	GLB	PO Box 1101	-	…

Invoices

InvoiceNo	CustomerID	Total Price	Due Date	Status
001-02345	ABC Inc	6,495.00	2008-02-15	PAID
001-02346	GlobalTech Corp	55,970.00	2008-02-25	AWAITING PAYMENT
…				

Query Result:

ABC Inc	6,495.00

SQL INSERT

- The INSERT command is used to add new records.
- Basic INSERT structure:

```
INSERT INTO [table name] VALUES (value1, value2, ...)
```

- It is also possible to specify the columns where the values should be inserted:

```
INSERT INTO [table name] (field1, field2, ...)
VALUES (value1, value2, ...)
```

- For example, we can add a new product to the Products table with the following statement:

```
INSERT INTO Products (ProductID, Name, Price)
VALUES (5120, 'Office Workstation', 1199.00)
```

SQL UPDATE

- The UPDATE command is used to modify a record
- It uses the WHERE clause to select the record(s) to modify
- The general structure is:

```
UPDATE [table name]
SET [field]=[new value]
WHERE [condition]
```

- For example:

```
UPDATE Products
SET Price='1,500.00'
WHERE ProductID="6290"
```

SQL DELETE

- The DELETE statement is used to remove a record
- Uses the WHERE clause to select a record(s).
- Structure:

```
DELETE FROM [table name] WHERE [condition]
```

- For example:

```
DELETE FROM Products
WHERE ProductID="6290"
```

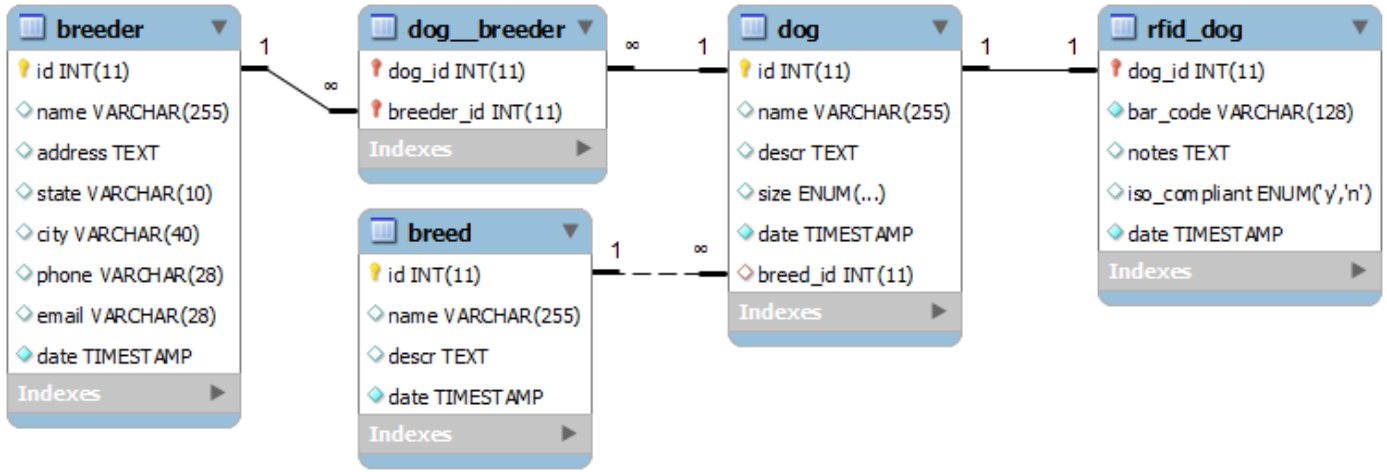
Enhanced Entity Relationship (EER) model

- Entity Relationship (ER) Model – represents an object
  - ✓ Physical – person, car
  - ✓ Conceptual – school, company
- ER model is based on the perception of the real world as a collection of objects with attributes
- Attributes – describe the entity
  - ✓ Single, Multi-value
  - ✓ Composite, Simple
  - ✓ Derived, Stored
- EER shows complex relationships between objects in a database (multimedia, geographical)

Example

- EER model
- Creating database according to model
  - ✓ Adding data
  - ✓ Querying database

EER model example

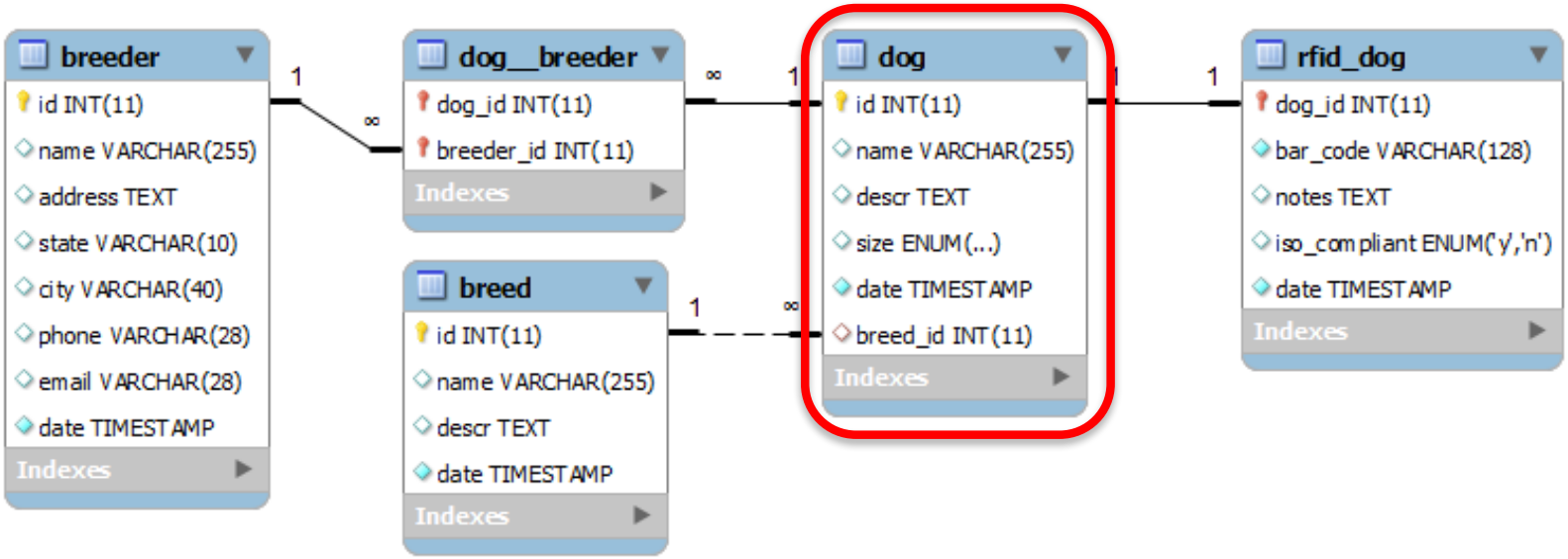


- ”Key” sign is denoted for PK (note two PK for dog__breeder table, which at the same time are FKs)
- Red is the indication of FK
- Filled diamond means that field cannot be empty
- Filled lines denote identifying relationships
  - Existence of a row in a child table depends on a row in a parent table
- Dotted lines denote non-identyfing relationships

MySQL

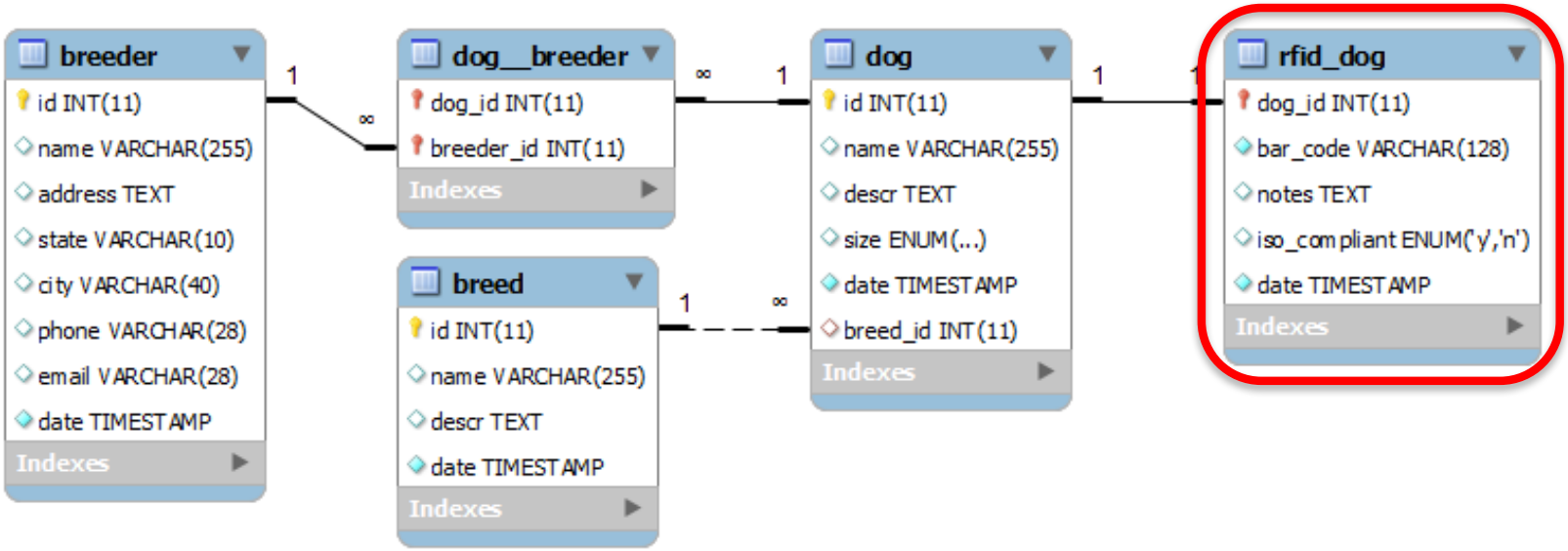
- MySQL is an open source database (https://www.mysql.com/products/)
- Community editions provide free server, connectors, MySQL Workbench and many more
- MySQL Workbench is the visual tool for database design and management, facilitating work with database and giving powerful tools for visualization and querying

Creating tables and adding data



Creating table ’dog’ and inserting some data there

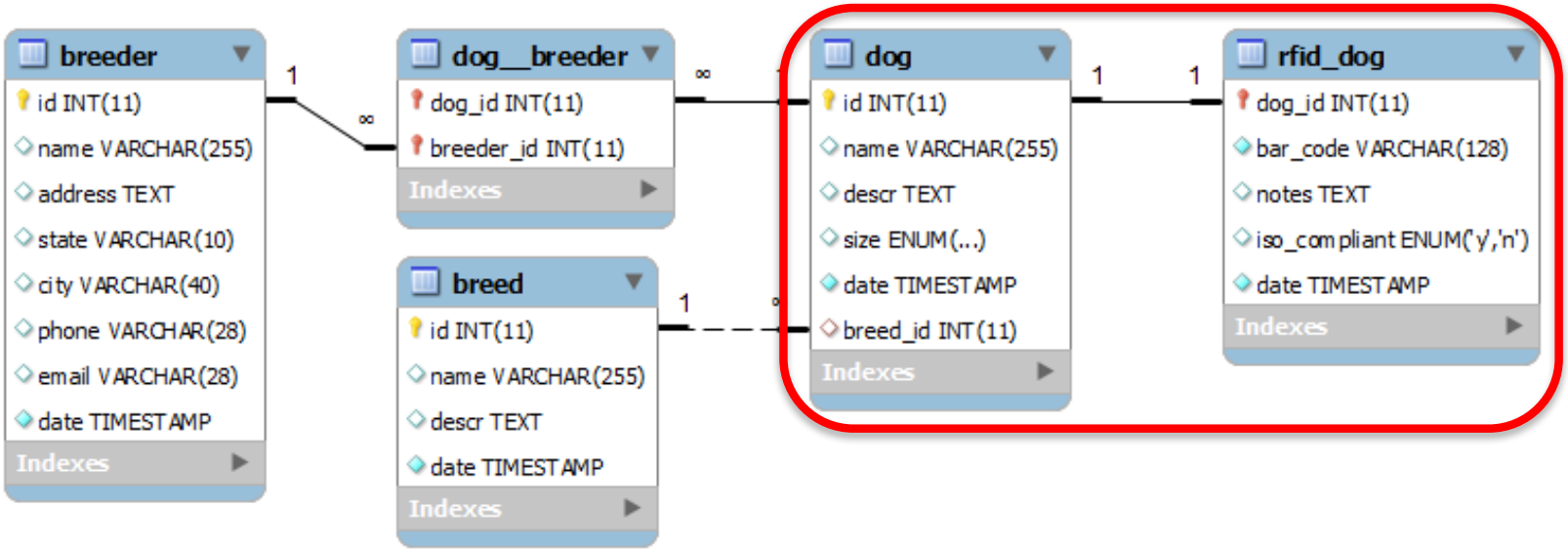
```
CREATE TABLE dog(
    id int(11) NOT NULL auto_increment,
    name varchar(255),
    descr text,
    size enum('small','medium','large'),
    date timestamp(6),
    PRIMARY KEY (id));

INSERT INTO dog (name,descr,size,date) VALUES('Max','Its
distinctive appearance and deep foghorn voice make it
stand out in a crowd.','medium',NOW());

INSERT INTO dog (name,descr,size,date) VALUES('Jake','It
loves human companionship and being part of the
group.','medium',NOW());

INSERT INTO dog (name,descr,size,date)
VALUES('Buster','Short-legged but surprisingly strong and
agile.','small',NOW());
```


breeder
id INT(11)
name VARCHAR(255)
address TEXT
state VARCHAR(10)
city VARCHAR(40)
phone VARCHAR(28)
email VARCHAR(28)
date TIMESTAMP
Indexes
1
∞
dog__breeder
dog_id INT(11)
breeder_id INT(11)
Indexes
breed
id INT(11)
name VARCHAR(255)
descr TEXT
date TIMESTAMP
Indexes
∞
1
dog
id INT(11)
name VARCHAR(255)
descr TEXT
size ENUM(...)
date TIMESTAMP
breed_id INT(11)
Indexes
1
rfid_dog
dog_id INT(11)
bar_code VARCHAR(128)
notes TEXT
iso_compliant ENUM('y','n')
date TIMESTAMP
Indexes

Creating table ’rfid_dog’ and inserting some data there

```
CREATE TABLE rfid_dog(
    dog_id int(11) NOT NULL,
    bar_code varchar(128) NOT NULL,
    notes text,
    iso_compliant enum('y','n') DEFAULT 'n',
    date timestamp(6),
    PRIMARY KEY (dog_id));

INSERT INTO rfid_dog
(dog_id,bar_code,notes,iso_compliant,date)
VALUES('1','234k343401123423023022','This is a RFID tag for
the Max','y',NOW());

INSERT INTO rfid_dog
(dog_id,bar_code,notes,iso_compliant,date)
VALUES('2','0938363892029039 7d829','This is a RFID tag for
the Jake','y',NOW());

INSERT INTO rfid_dog
(dog_id,bar_code,notes,iso_compliant,date)
VALUES('3','30id8383837210jndal20','This is a RFID tag for
the Buster','y',NOW());
```

Query example



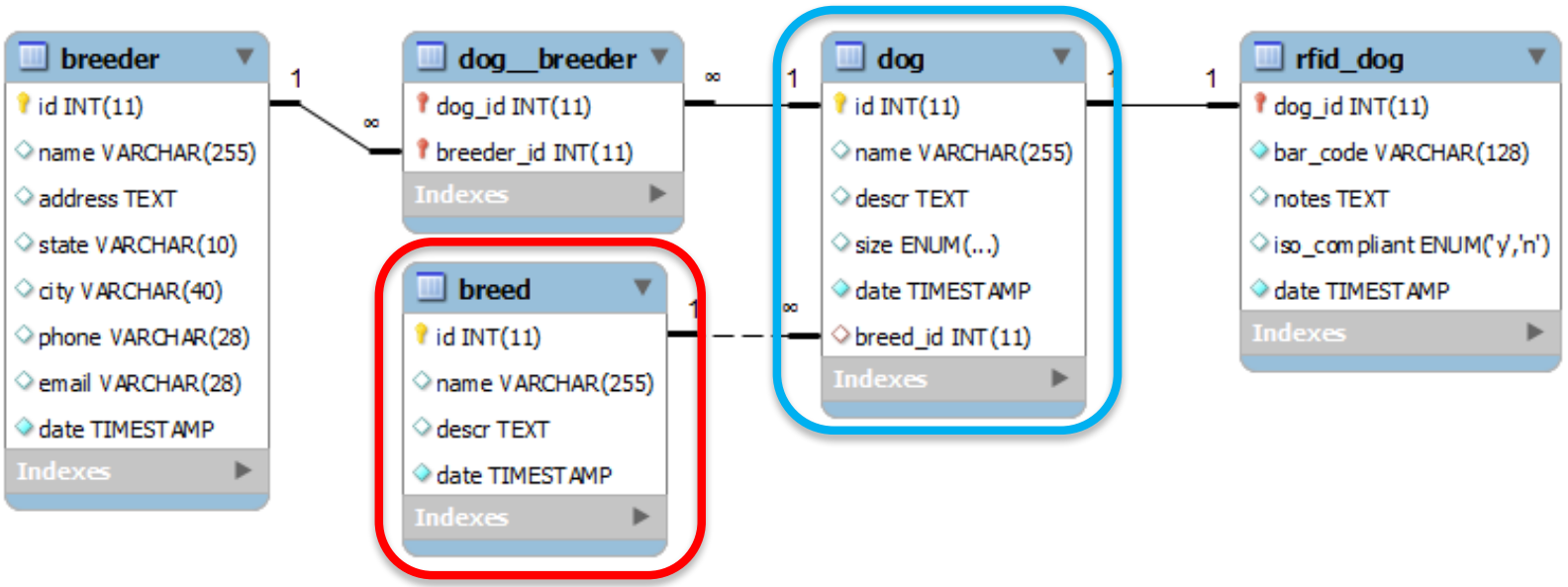
Query example for one-to-one relationship

- Query the list of dog ID, name, and corresponding RFID

```
SELECT dog.id, dog.name, rfid_dog.bar_code AS rfid
FROM dog,rfid_dog
WHERE dog.id = rfid_dog.dog_id
ORDER BY dog.name ASC;
```

- Result

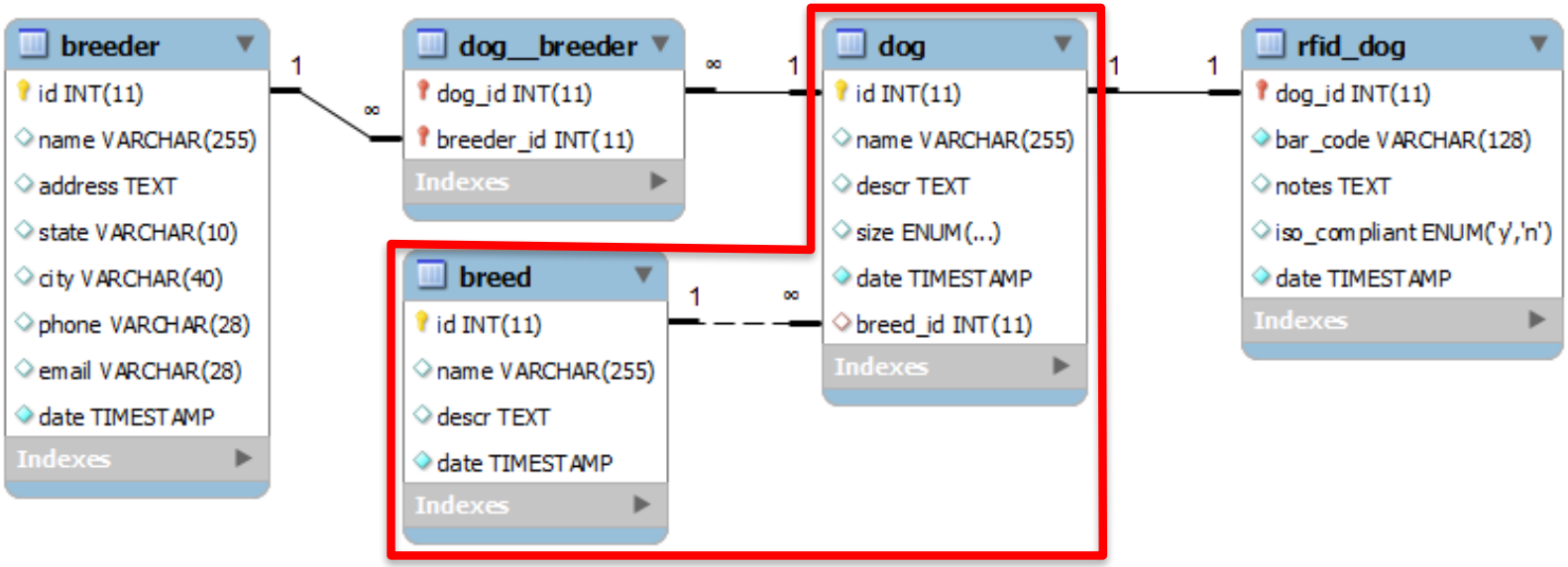
```
+----+--------+------------------------+
| id | name   | rfid                   |
+----+--------+------------------------+
|  3 | Buster | 30id8383837210jndal20  |
|  2 | Jake   | 0938363892029O397d829  |
|  1 | Max    | 234k3434011234232302l  |
+----+--------+------------------------+
```


breeder
id INT(11)
name VARCHAR(255)
address TEXT
state VARCHAR(10)
city VARCHAR(40)
phone VARCHAR(28)
email VARCHAR(28)
date TIMESTAMP
Indexes
dog__breeder
dog_id INT(11)
breeder_id INT(11)
Indexes
breed
id INT(11)
name VARCHAR(255)
descr TEXT
date TIMESTAMP
Indexes
dog
id INT(11)
name VARCHAR(255)
descr TEXT
size ENUM(...)
date TIMESTAMP
breed_id INT(11)
Indexes
rfid_dog
dog_id INT(11)
bar_code VARCHAR(128)
notes TEXT
iso_compliant ENUM('y','n')
date TIMESTAMP
Indexes
1
∞

Create table ’breed’, change table ’dog’

```
CREATE TABLE breed(
    id int(11) NOT NULL auto_increment,
    name varchar(255),
    descr text,
    date timestamp(6),
    PRIMARY KEY (id));

INSERT INTO breed (name,descr,date) VALUES('Hounds','One
of the oldest groups of dog originating thousands of years
ago.',NOW());

INSERT INTO breed (name,descr,date) VALUES('Terrier','A
tough, no-nonsense, rabbiting and badgering dog.',NOW());

ALTER TABLE dog
ADD breed_id int(11) AFTER date;

UPDATE dog SET breed_id = '1' WHERE id = '1';
UPDATE dog SET breed_id = '1' WHERE id = '2';
UPDATE dog SET breed_id = '2' WHERE id = '3';
```

Query example



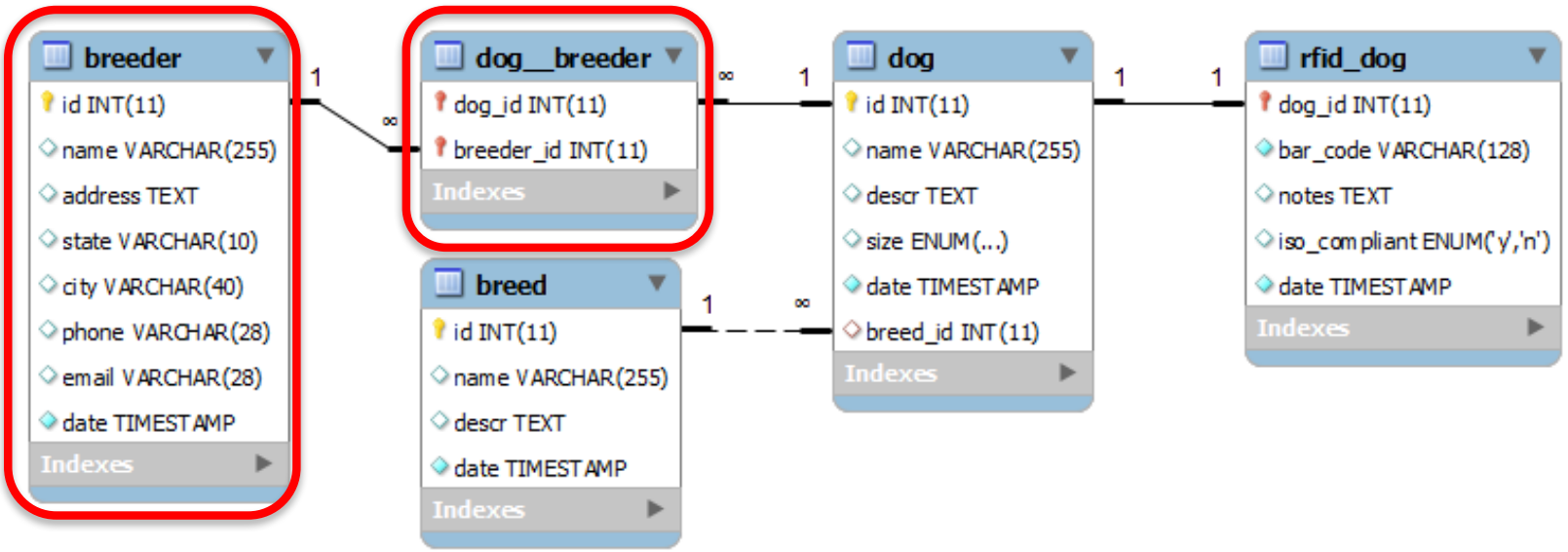
Query example for one-to-many relationships

- Get a list of dogs and their breed

```
SELECT dog.id,dog.name,breed.name AS breed
FROM dog,breed
WHERE dog.breed_id = breed.id
ORDER BY dog.name ASC;
```

- Result

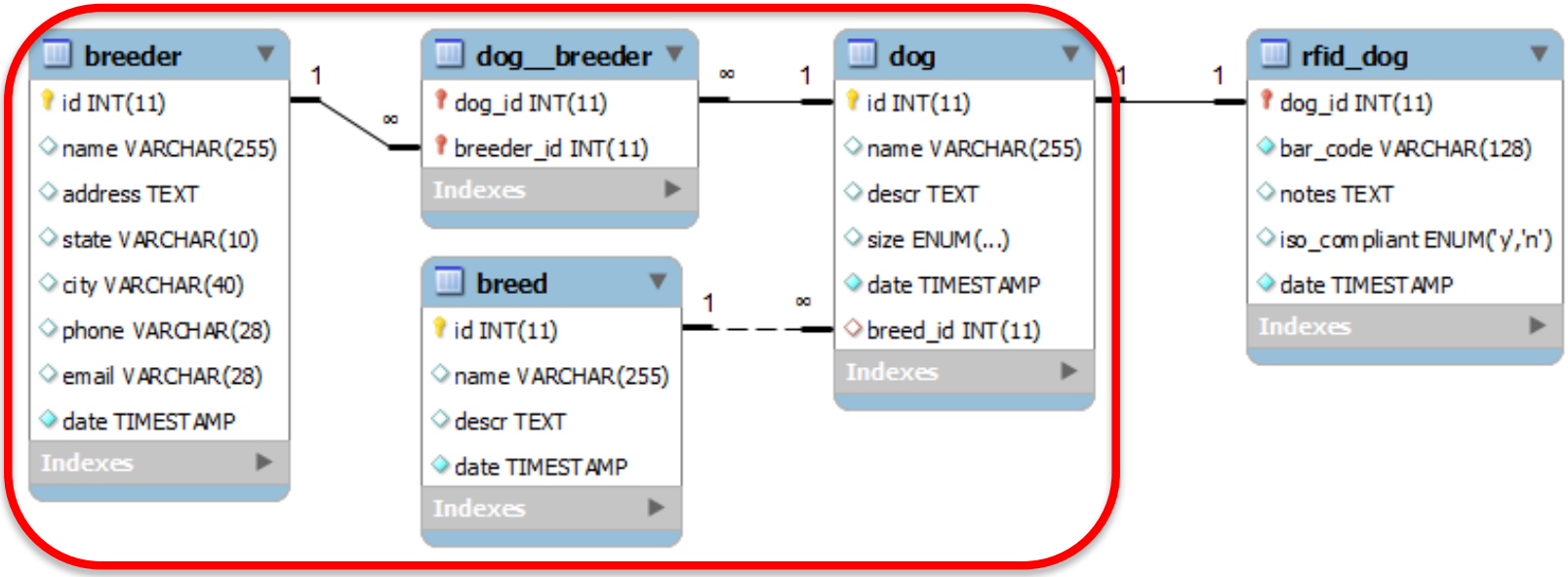
```
+----+--------+---------+
| id | name   | breed   |
+----+--------+---------+
|  3 | Buster | Terrier |
|  2 | Jake   | Hounds  |
|  1 | Max    | Hounds  |
+----+--------+---------+
```


breeder
id INT(11)
name VARCHAR(255)
address TEXT
state VARCHAR(10)
city VARCHAR(40)
phone VARCHAR(28)
email VARCHAR(28)
date TIMESTAMP
Indexes
1
∞
dog__breeder
dog_id INT(11)
breeder_id INT(11)
Indexes
breed
id INT(11)
name VARCHAR(255)
descr TEXT
date TIMESTAMP
Indexes
1
∞
∞
1
dog
id INT(11)
name VARCHAR(255)
descr TEXT
size ENUM(...)
date TIMESTAMP
breed_id INT(11)
Indexes
1
1
rfid_dog
dog_id INT(11)
bar_code VARCHAR(128)
notes TEXT
iso_compliant ENUM('y','n')
date TIMESTAMP
Indexes

Create tables ’breeder’ and ’dog_breeder’

```
CREATE TABLE breeder (
     id int(11) NOT NULL auto_increment,
     name varchar(255),
     address text,
     state varchar(10),
     city varchar(40),
     phone varchar(28),
     email varchar(28),
     date timestamp(6),
     PRIMARY KEY (id));

INSERT INTO breeder (name,address,state,city,phone,email,date) VALUES('Joe
Bloggs','23 Smith St','NSW','Sydney','02 7875 4545','joe@email.com',NOW());

INSERT INTO breeder (name,address,state,city,phone,email,date) VALUES('Tom
Smith','11 Tucker St','QLD','Brisbane','07 023 2343','tom@email.com',NOW());

CREATE TABLE dog__breeder(
     dog_id int(11) DEFAULT '0' NOT NULL,
     breeder_id int(11) DEFAULT '0' NOT NULL,
     PRIMARY KEY (dog_id, breeder_id));

INSERT INTO dog__breeder (dog_id,breeder_id) VALUES('1','1');
INSERT INTO dog__breeder (dog_id,breeder_id) VALUES('2','1');
INSERT INTO dog__breeder (dog_id,breeder_id) VALUES('1','2');
```

Query example



Query example for many-to-many relationship

- Get the breeder name, address, and the name of his/her dog with breed

```
SELECT breeder.name,breeder.address,dog.name AS dogName,
breed.name AS breed
FROM breeder,dog,breed,dog__breeder
WHERE dog__breeder.dog_id = dog.id
AND dog__breeder.breeder_id = breeder.id
AND dog.breed_id = breed.id
ORDER BY breeder.name ASC;
```

- Result:

```
+------------+--------------+---------+--------+
| name       | address      | dogName | breed  |
+------------+--------------+---------+--------+
| Joe Bloggs | 23 Smith St  | Max     | Hounds |
| Joe Bloggs | 23 Smith St  | Jake    | Hounds |
| Tom Smith  | 11 Tucker St | Max     | Hounds |
+------------+--------------+---------+--------+
```

Homework

- Set up MySQL environment with MySQL Workbench in your laptops
  - ✓ Will be needed for the next lecture
  - ✓ Will be needed for the second assignment
- Create tables from lecture slides
- Practice:
  - ✓ Try to add, modify, delete data
  - ✓ Try querying for different kind of information

Summary

- Databases is an important part of information exchange, sharing and storing
- Existing systems mostly use relational and object oriented databases
  - ✓ Although the contemporary hardware is fast, the large databases might become a bottleneck for the information exchange. Additional mechanisms for information sharing has to be considered

Resources

- Course FIS book
- Lecture slides from databases introductory course: http://db.grussell.org/resources/lectures.html
- Introduction to relational databases: http://web.stanford.edu/~ouster/cgi-bin/cs142-spring12/lecture.php?topic=sql
- Breeders and dogs initial example: http://www.anchor.com.au/hosting/support/CreatingAQuickMySQLRelationalDatabase