Tables
Keys
Relationships

Tables and Keys

- Tables are the only containers in a relational database
- Must be well designed when creating a database
- Table columns
  - Some act as unique identifiers
    - Define the intent and purpose of each row
  - Some hold supporting data
  - Some act as external references that link rows in one table to rows in another table

Sample table

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

Primary Keys (PK)

- Primary Key – one or more columns that uniquely identify each row of a table
- Their values must be unique
- Other columns support data that is directly relevant to the primary key
- Values for a primary key can be
  - Actual unique data, such as Student Id or SSN
  - Arbitrary numbers automatically generated to uniquely identify customers or employees

- Each entry in an SQLite table has a unique integer key called the "rowid".

Primary Keys (cont.)

- Extremely important in the design and use of a database
- A primary key can be identified in the CREATE TABLE command

Example:

CREATE TABLE employee (employee_Id INTEGER PRIMARY KEY NOT NULL, name TEXT NOT NULL);

- When creating documentation, we can refer to a primary key as **PK**

Primary Keys (cont.)

Here we have a second table in the database to store orders placed by customers

Each row in the ORDERS table represents a single order placed by a single customer

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

ORDERS

OrderID	CustomerID	Amount	Date
1	3	27.50	02-Apr-2000
2	1	12.99	15-Apr-2000
3	2	74.00	19-Apr-2000
4	4	6.99	01-May-2000

Each order in the Orders table refers to a customer from the Customers table.

Foreign Keys (FK)

- Primary keys are crucial to joining tables together
- We can create a reference to a specific row in the table containing the a primary key by having a copy of the primary key in another table
  - This column will behave as a Foreign Key
- Unlike a table's own primary key, foreign keys are not required to be unique
  - Multiple values (multiple rows) in one table may refer to the same row in another table
  - This is known as: one-to-many relationship

CUSTOMERS

CustomerID	Name	Address	City
1	Julie Smith	25 Oak Street	Airport West
2	Alan Wong	1/47 Haines Avenue	Box Hill
3	Michelle Arthur	357 North Road	Yarraville

ORDERS

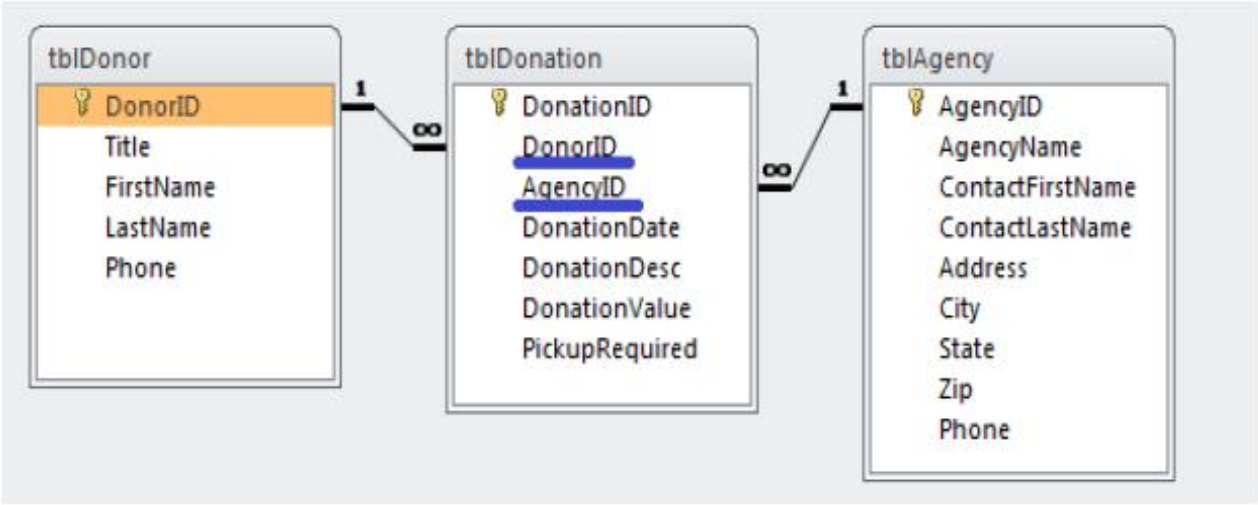
OrderID	CustomerID	Amount	Date	Books Ordered
1	3	27.50	02-Apr-2000	0-672-31697-8
2	1	12.99	15-Apr-2000	0-672-31745-1, 0-672-31509-2
3	2	74.00	19-Apr-2000	0-672-31697-8
4	3	6.99	01-May-2000	0-672-31745-1, 0-672-31509-2, 0-672-31697-8

Foreign Keys constraints

- Are used to keep foreign key references in sync
  - Prevent having “dangling references” also called ‘orphan records’
  - All foreign key values must match a row value from the columns of the reference table (the primary table)

  For example: the Customer and Rep tables both have the REP_NUM column. This column is the PK in the Rep table, and the FK in the Customer table. What happens if we delete a record from the Rep table (get rid of a sales rep)?

Foreign Keys (FK) - Example



Common Structures and Relationships

- One-to-One Relationships
  - Most basic kind of inter-table relationship
  - Establishes a reference from a single row in one table to a single row in another table
  - Commonly represented by having a foreign key in one table reference a primary key in another table
  - Commonly used to create ‘detail tables’
    - Detail tables are helpful when extended data only applies to a limited number of records

One-to-One Relationships (2)

Example:

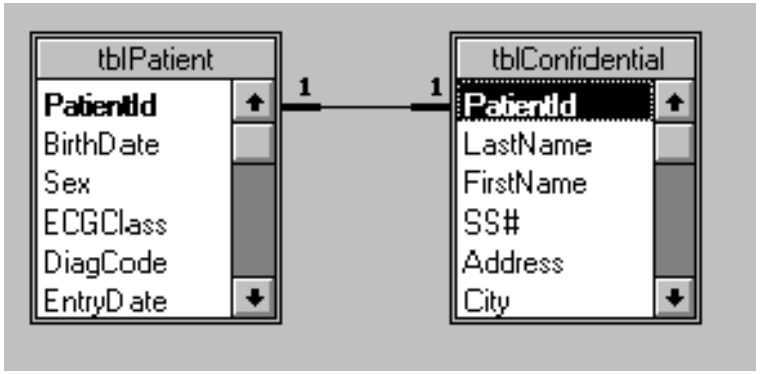
A **sales_items** table with common information (price, inventory, weight, etc) for all available items.

And a detail table with type specific data for each item, such a **cd_info** table for CD's, **dvd_info** table for DVD's.

Each individual row in sales_item would be referenced by only one row in the detail tables (**cd_info** and **dvd_info)**

One-to-One Relationships (3)

May be necessary in a database when we need to split a table into two or more tables because of security or performance concerns

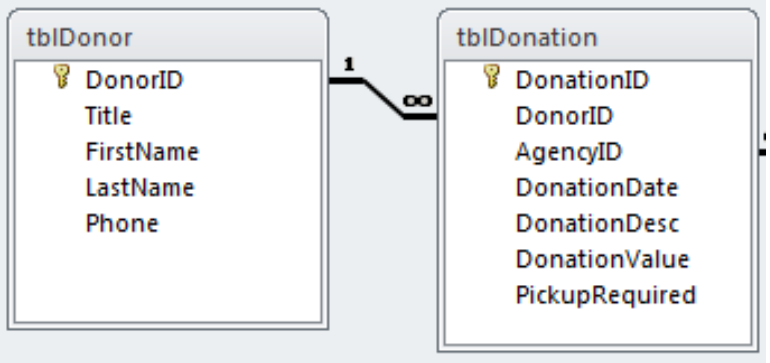


One-to-Many Relationships

- Establish a link between a single row in one table to multiple rows in another table
- For example: One sales rep in the REP table can have one or more customers in the Customer table
- The FK in then Customer table (REP_NUM) allows for duplicates
  - Can appear in more than one record in the Customer table
- The FK is always in the ‘many’ side

One-to-Many Relationships (2)

- Probably the most commonly used type of relationship
- Example: A table with Customer information where a customer could have multiple addresses. One could design a table a column for each ‘possible’ address, or, one could design two tables, where the second table could have one or more addresses per customer



One-to-Many Relationships

- Here is an example of a one-to-many relationship: music albums and the songs
- Each album has a list of songs associated with that album

```
CREATE TABLE albums( album_id INTEGER NOT NULL PRIMARY KEY,
album_name TEXT);

CREATE TABLE tracks(track_id INTEGER NOT NULL PRIMARY KEY,
track_name TEXT,track_number INTEGER,track_length INTEGER,
album_id INTEGER NOT NULL  REFERENCES albums );
```

One-to-Many Relationships

- Each album and track has a unique ID
- Each track has the ID as a foreign key (FK)

We enter the following record into the **albums** and **tracks** tables:

INSERT INTO **albums** VALUES (1, "The Indigo Album");

INSERT INTO tracks VALUES (1, "Metal Onion",1, 137,1);

INSERT INTO tracks VALUES (2, "Smooth Snake",2, 212,1);

INSERT INTO tracks VALUES (3, "Turn A",3, 255,1);

INSERT INTO **albums** VALUES (2, "Morning Jazz");

INSERT INTO tracks VALUES (4, "In the Bed",1,214,2);

INSERT INTO tracks VALUES (5, "Water All Around",2, 194,2);

INSERT INTO tracks VALUES (6, "Time Soars",3, 265,2);

INSERT INTO tracks VALUES (7, "Liquid Awareness",4, 175,2);

One-to-Many Relationships

We can get a list of the album name from the albums table and track name from the tracks table

select album_name, track_name from albums, tracks;

album_name	**track_name**
The Indigo Album	Metal Onion
The Indigo Album	Smooth Snake
The Indigo Album	Turn A
The Indigo Album	In the Bed
The Indigo Album	Water All Around
The Indigo Album	Time Soars
The Indigo Album	Liquid Awareness

One-to-Many Relationships

(continued from previous slide)

album_name	track_name
Morning Jazz	Metal Onion
Morning Jazz	Smooth Snake
Morning Jazz	Turn A
Morning Jazz	In the Bed
Morning Jazz	Water All Around
Morning Jazz	Time Soars
Morning Jazz	Liquid Awareness

(is this accurate?)

One-to-Many Relationships

We can get a list of track and their associated album by **joining** the tables:

```
.headers on
.mode column
.width 30,30,30
SELECT album_name, track_name, track_number
FROM albums, tracks
WHERE tracks.album_id = albums.album_id
ORDER BY album_name, track_number;
```

One-to-Many Relationships

album_name	track_name	track_number
Morning Jazz	In the Bed	1
Morning Jazz	Water All Around	2
Morning Jazz	Time Soars	3
Morning Jazz	Liquid Awareness	4
The Indigo Album	Metal Onion	1
The Indigo Album	Smooth Snake	2
The Indigo Album	Turn A	3

One-to-Many Relationships

We can get a list of track and their associated album by **joining** the tables using the **WHERE** clause:

```
SELECT album_name, track_name, track_number
FROM albums, tracks
WHERE albums.album_id = tracks.album_id
ORDER BY album_name, track_number;
```

album_name	track_name	track_number
Morning Jazz	In the Bed	1
Morning Jazz	Water All Around	2
Morning Jazz	Time Soars	3
Morning Jazz	Liquid Awareness	4
The Indigo Album	Metal Onion	1
The Indigo Album	Smooth Snake	2
The Indigo Album	Turn A	3

One-to-Many Relationships

We can group the tracks based off their album, and then aggregate the track data together. The following statement adds the track length for each album id, uses the alias ‘runtime’ , and counts the number of records for each group

```
SELECT album_name, sum(track_length) AS runtime, count(*) AS tracks
FROM albums, tracks
WHERE   albums.album_id = tracks.album_id
GROUP BY tracks.album_id;
```

album_name	runtime	tracks
The Indigo Album	604	3
Morning Jazz	848	4

One-to-Many Relationships

We can list data from the customer and rep tables based on their common field/column:

select customer_num, customer_name, rep.rep_num, last_name, first_name from customer, rep WHERE customer.rep_num = rep.rep_num;

CUSTOMER_NUM	CUSTOMER_NAME	REP_NUM	LAST_NAME	FIRST_NAME
148	Al's Appliance and	20	Kaiser	Valerie
282	Brookings Direct	35	Hull	Richard
356	Ferguson's	65	Perez	Juan
408	The Everything Shop	35	Hull	Richard
462	Bargains Galore	65	Perez	Juan
524	Kline's	20	Kaiser	Valerie
608	Johnson's Department	65	Perez	Juan
687	Lee's Sport and Appl	35	Hull	Richard
725	Deerfield's Four Sea	35	Hull	Richard
842	All Season	20	Kaiser	Valerie

Many-to-Many Relationships

- Associates one row in the first table to many rows in the second table wile simultaneously allowing individual rows in the second table to be linked to multiple rows in the first table
- A many-to-many relationship is really two one-to-many relationships build across each other

  Typical example: people and groups

  – One person can belong to many different groups

  – While each group is made up of many different people

  We may need to find

  – all the groups a person belongs to

  – all the people in a group

Many-to-Many Relationships

- We cannot directly represent a many-to-many relationship with only two tables
- We need a link table/or bridge table that sits between the two ‘many’ tables
- Each many-to-many relationship requires a unique bridge table