

# Creating and Altering Structures and Data

DT228/DT282/2

Dr Emma Murphy

Databases 1, Week 2



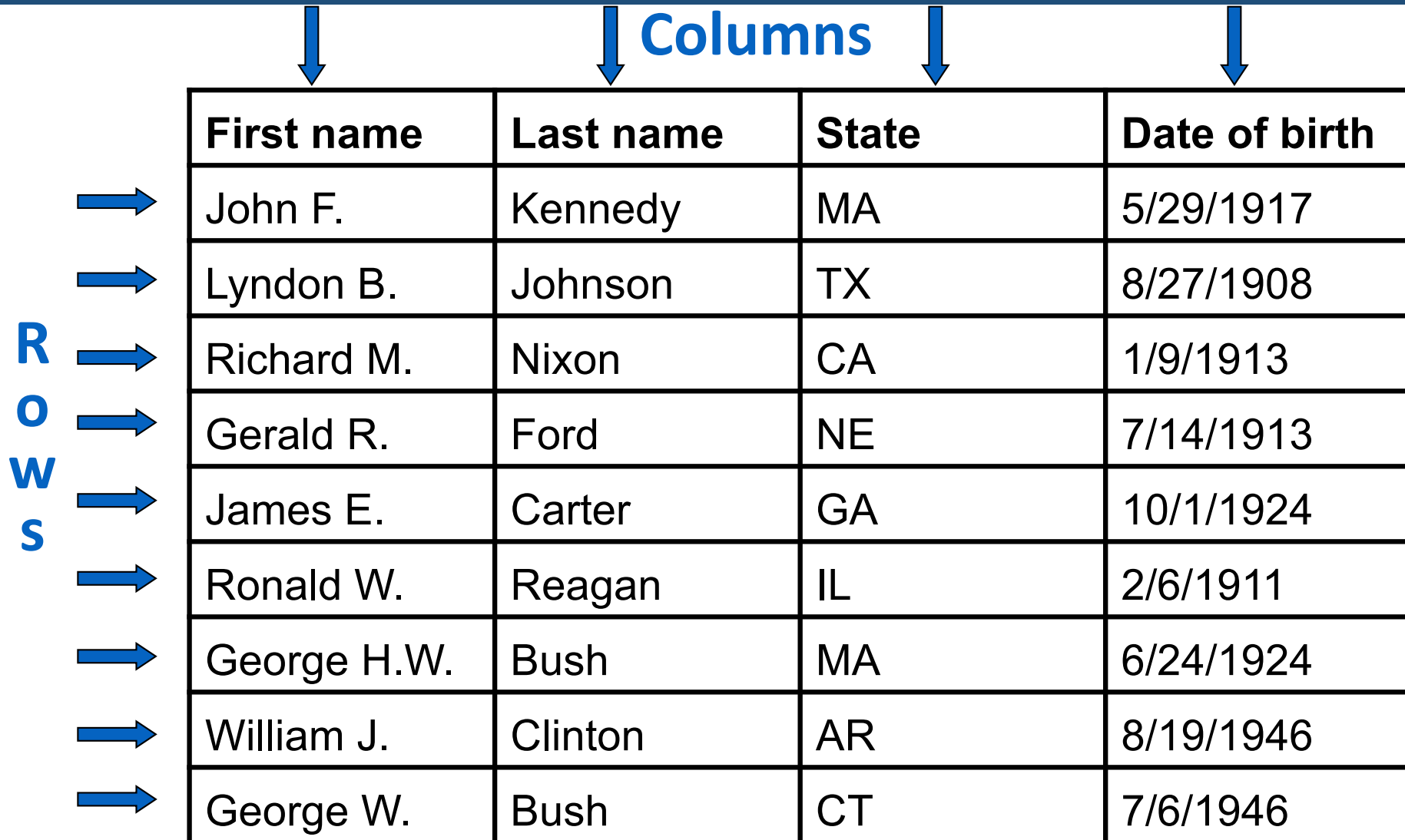
# What do we know so far?

- Database
  - An *organized, machine-readable* collection of *symbols* which can be interpreted as a true account of some enterprise
  - It is also machine-updatable, so the database is also a collection of *variables*
    - Variables have a *datatype* which determines the type of data that can be stored in it
  - Its organization/structure depends on the model being used
    - We are using the *relational* model
  - A database is typically available to a community of users, with possibly varying requirements.

# What do we know so far?

- Relational Database (Physical)
  - A collection of related records called *rows* (consisting of variables called *columns*), organized into *tables*
  - Tables are connected based on common variables
    - Records are related based on having common values for these common variables

# A database table



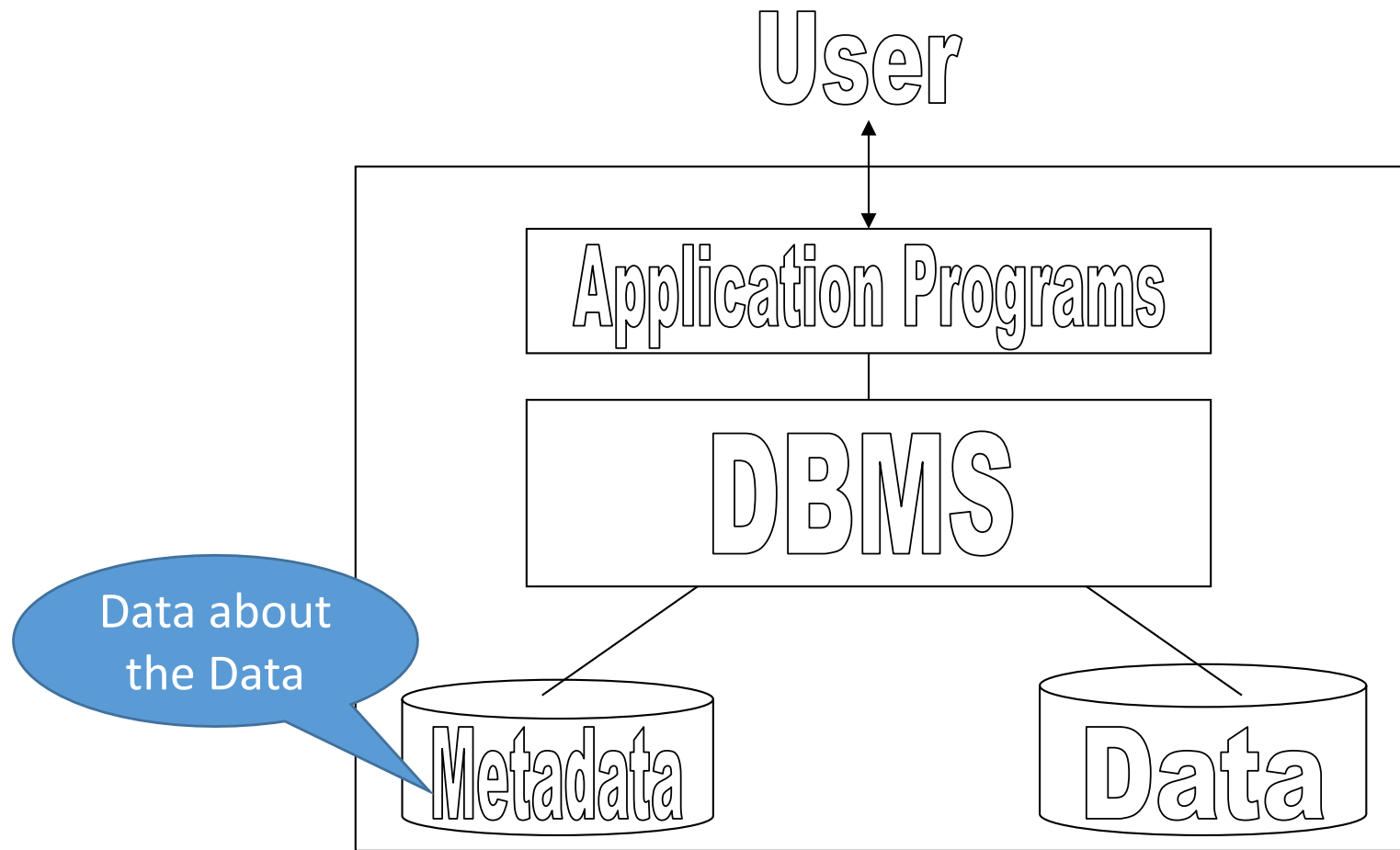
The diagram illustrates a database table with four columns and nine rows. The columns are labeled 'First name', 'Last name', 'State', and 'Date of birth'. The rows contain data for various US presidents. Blue arrows point to the column headers from the word 'Columns' and to the row headers from the word 'ROWS'.

	First name	Last name	State	Date of birth
→	John F.	Kennedy	MA	5/29/1917
→	Lyndon B.	Johnson	TX	8/27/1908
→	Richard M.	Nixon	CA	1/9/1913
→	Gerald R.	Ford	NE	7/14/1913
→	James E.	Carter	GA	10/1/1924
→	Ronald W.	Reagan	IL	2/6/1911
→	George H.W.	Bush	MA	6/24/1924
→	William J.	Clinton	AR	8/19/1946
→	George W.	Bush	CT	7/6/1946

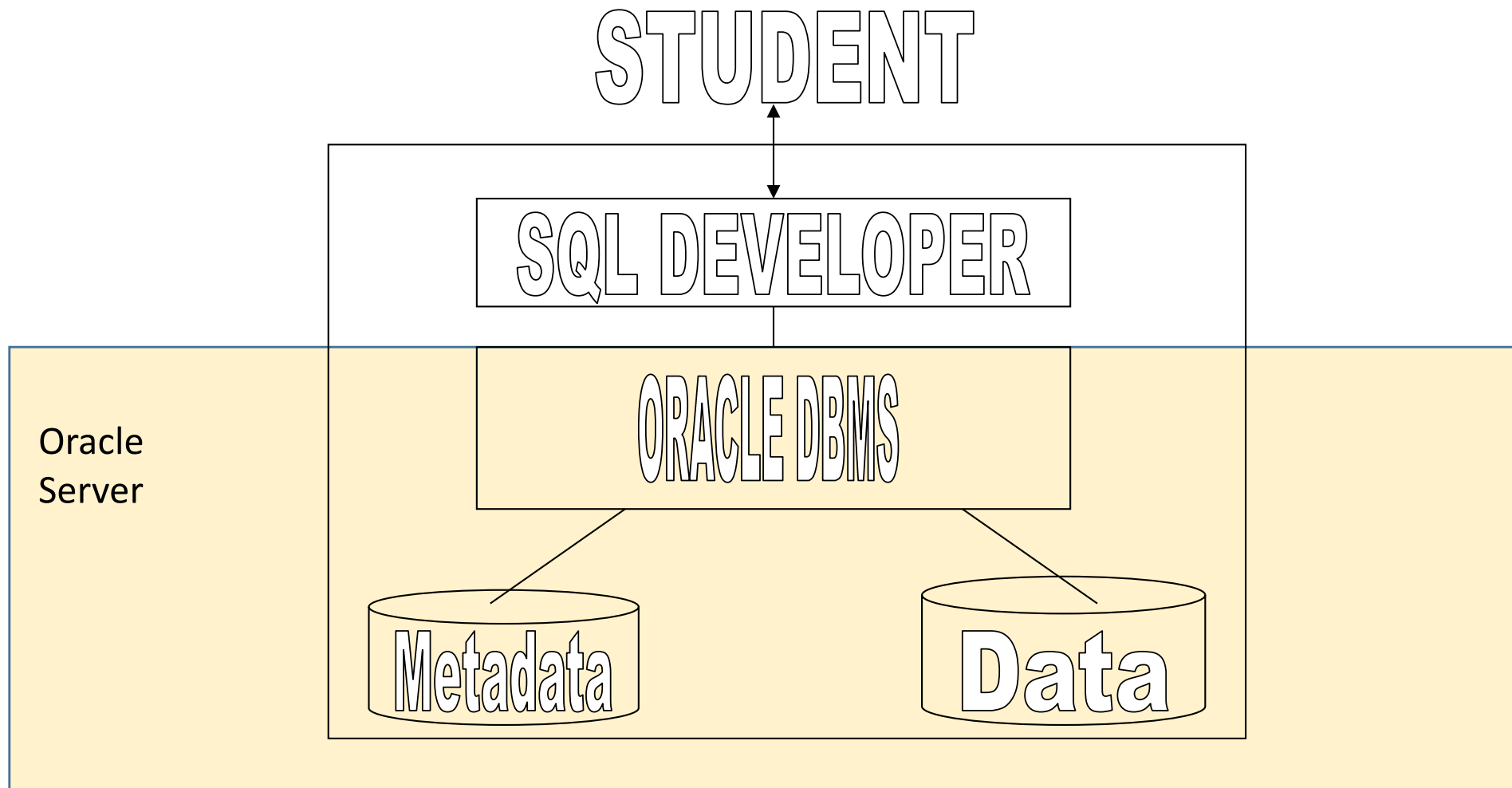
# What do we know so far?

- Database Management System (DBMS)
  - System software that allows us to **manage** the content/information stored in a database

# Database Application



# How are we interacting with the Oracle Database at TU Dublin?



# How do we instruct the DBMS?

- SQL (Structured Query Language)
  - Includes:
    - Data Manipulation Language (DML)
    - Data Definition Language (DDL)
- Before we can manipulate data, someone needs to
  - Define the data structure
  - Populate the data
    - Adhering to the constraints (outlined in the meta-data)



# Get access to a database

- How to get access to the database?
  - You need to know the name of the database or schema
  - You need to have privilege view or create data structures within it
  - You need to create a connection to the database

# Oracle Architecture

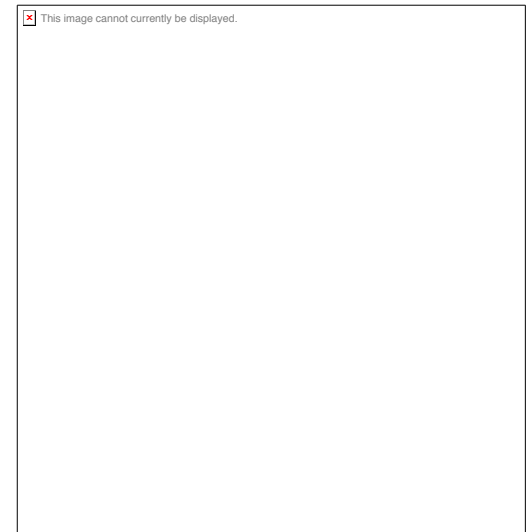
- An Oracle database consists of:
  - A large amount of stored data on disk (in physical files)
  - An Oracle instance
    - A set of memory structures that manage database files
    - That allows users to request services and manipulate the physical database (our DBMS)
- Without the instance, the Oracle database is not usable
  - It is like having a book and not being able to read.

# Client / server architecture

- The database application and the database are separated into two parts:
  - a front-end or **client** portion, and
  - a back-end or **server** portion

# DIT Client/Server Architecture

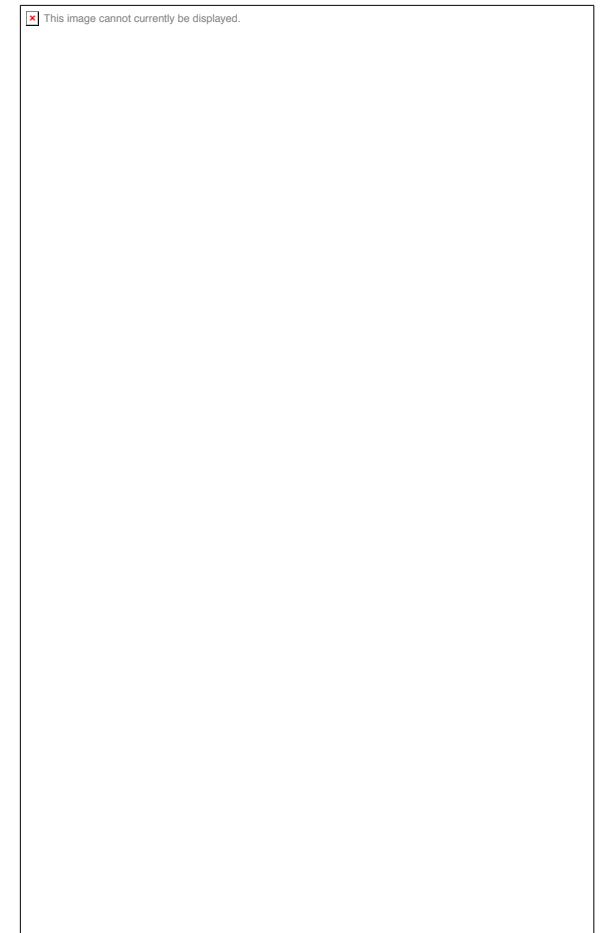
- The data is held in a database on a **server**
- We access and manipulate the data using an application on the **client**.
  - Our **client** application is **SQL Developer**.
    - Either installed on the lab PC or your laptop.



# TU Dublin Client/Server Architecture

- The client can access the database server through an application that sits on the server and can interact with that server.
- We identify the application we want to use when we create a connection.
- The client sends requests to the application server, which services them and returns a result.

- **The service we connect to DIT Oracle with is  
PDB1.sub03101424270.tudublinstudent.oraclevcn.com**



# TU Dublin Oracle Architecture

- Each of you will have an Oracle account which allows you to create objects (tables, queries, views, reports) within the TU Dublin Oracle database
- Each user is allocated an amount of space they can use
  - Tablespace – logical space used for storage
    - Users are allocated to particular tablespaces

# Creating a connection

- Within SQL developer
  - Create a connection
  - Provide details of the server
  - Indicate the application service you want to use to facilitate the connection
  - Provide your username and password
    - Note: these are case sensitive
- We will set this up in the labs

# Creating a connection

Schema/User name : Provided in spreadsheet

Password : Provided in spreadsheet

Connection Type : Basic

Role : default

Host Name : 132.145.215.49

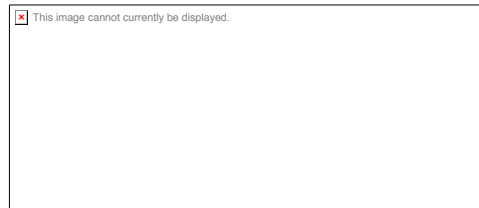
Port : 1521

ServiceName: [PDB1.sub03101424270.tudublinstudent.oraclevcn.com](https://PDB1.sub03101424270.tudublinstudent.oraclevcn.com)



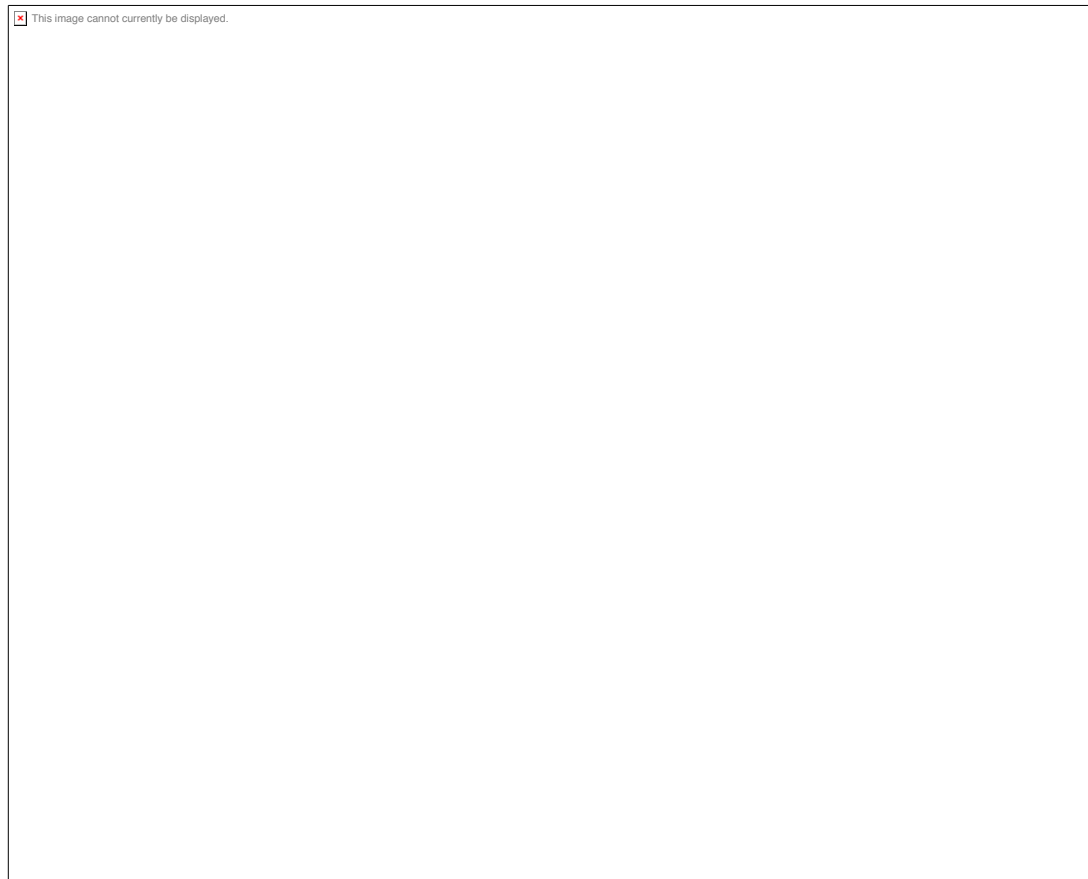
# Creating a connection

- When you have opened SQL Developer you will you will need to create a connection to the database.
- Click on the green plus icon on the top left of the tool



# Creating a connection

A Connections window will open.  
Enter your Database Connection details

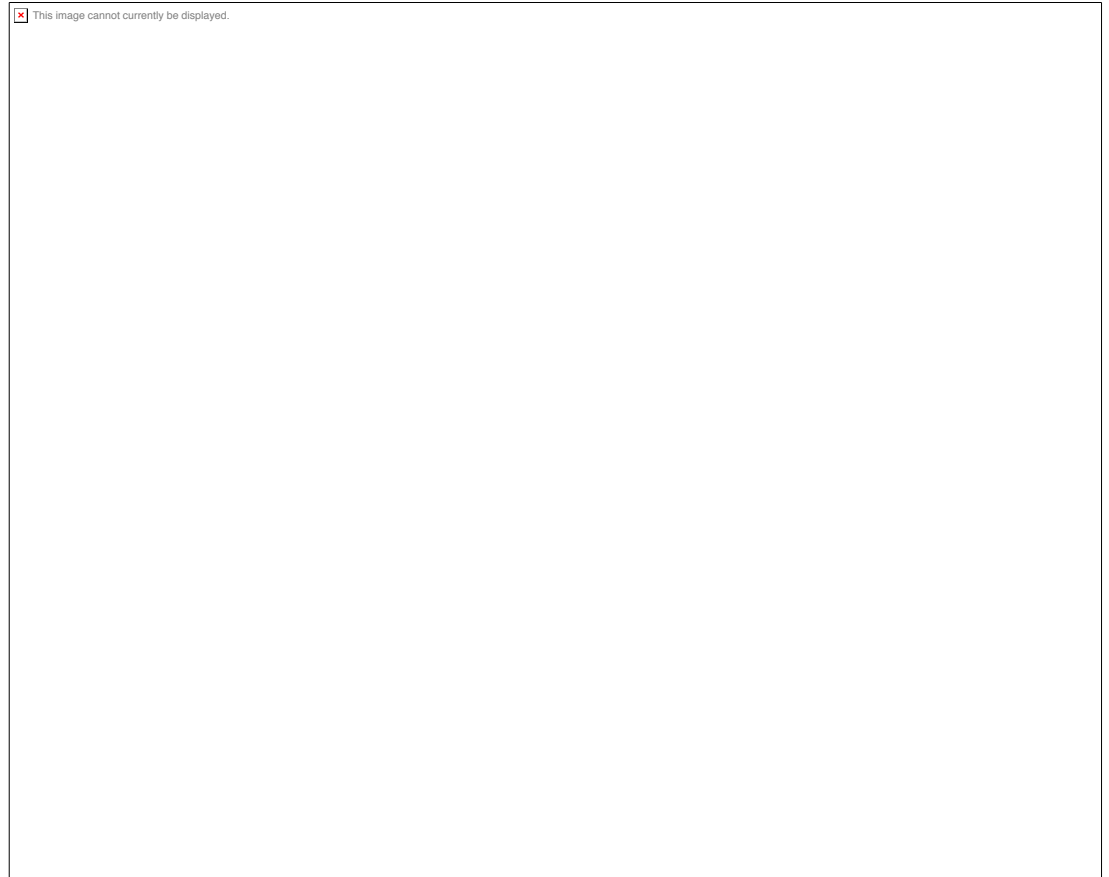


# Creating a connection

Click on the Test button to see if you have entered the details correctly. If you get an error, then you have not entered the details correctly.

Click the Save button to save the connection details.

Finally, click the Connect button to make the database connection and open a SQL Worksheet



# Creating a connection

- Details of usernames and passwords required for the connection are available here:
- [https://drive.google.com/open?id=13nfriL32ppeYOKzIqhNmHM\\_muidJesmr](https://drive.google.com/open?id=13nfriL32ppeYOKzIqhNmHM_muidJesmr)

# Creating Tables

# Table

- Fundamental structure in the relational model
- Organised collection of symbols
  - Organised into rows and columns

studentId	firstname	programme
S1	Anne	C1
S1	Anne	C2
S2	Boris	C1
S3	Cindy	C3

# Table

- Fundamental structure in the relational model
- Organised collection of variables which can be populated by symbols
  - studentId is a variable, firstname is a variable, programme is a variable
  - The table itself is a variable

studentId	firstname	programme
S1	Anne	C1
S1	Anne	C2
S2	Boris	C1
S3	Cindy	C3

# What Is a DBMS?

- A DBMS responds to *imperatives* ('statements') given by application programs, custom-written or general-purpose, executing on behalf of users.
- Imperatives are written in the database language of the DBMS (e.g., SQL).
- Responses include completion codes, messages and results of queries.



# SQL: Data Definition Language (DDL) Commands

- CREATE TABLE: used to create a table.
- ALTER TABLE: modifies a table after it was created.
- DROP TABLE: removes a table from a database.

# Creating a Table

- A table is an object that can store data in an Oracle database.
- You will create a table for each 'thing' you want to store data about.
- When you create a table, you must specify:
  - the table name,
  - the name of each column,
  - the data type of each column,
  - and the size of each column
  - any constraints on the data that each column can contain.

# Example IMDB

- Suppose we want to model a very basic version of IMDB (Internet Movie Database)
- We want to store data about movies and actors?
- We need TWO basic tables (There will be others)



MOVIE

ACTOR

# Example

- What data do we need to store about them?

Movie

movieId, movieTitle, Year, Director, Budget, Profit

Actor

actorId, actorName

# Example

- What piece of data is unique ?

Actor

actorId, actorName

Movie

movieId, movieTitle, Year, Director, Budget,  
Profit

**Movie: MovieID**

**Actor: ActorID**

**These are the primary keys**

# Create Table Statement

- You must have:
  - CREATE TABLE privilege
  - Access to a storage area
- Anything in [ ] is optional

```
CREATE TABLE [schema.]table  
              (column datatype [DEFAULT expr] [, ...]);
```

- You specify:
  - Table name
  - Column name, column data type, and column size, any constraints

# Data Types

Data Type	Description
<b>VARCHAR2</b> ( <i>size</i> )	Variable-length character data
<b>CHAR</b> ( <i>size</i> )	Fixed-length character data
<b>NUMBER</b> ( <i>p</i> , <i>s</i> )	Variable-length numeric data
<b>DATE</b>	
<b>LONG</b>	Variable-length character data (up to 2 GB)
<b>CLOB</b>	Character data (up to 4 GB)
<b>RAW</b> and <b>LONG RAW</b>	Raw binary data
<b>BLOB</b>	Binary data (up to 4 GB)
<b>BFILE</b>	Binary data stored in an external file (up to 4 GB)
<b>ROWID</b>	A base-64 number system representing the unique address of a row in its table

# Example

- What are the datatypes for our data?

Movie

movieId, movieTitle , Year, Director, Budget, Profit

movieId Number(4), movieTitle Varchar2(50),  
Year Number(4), Director Varchar2(50), Budget  
Number(11,2), Profit (11,2)

Actor

actorId, actorName

actorId Number (4), actorName Varchar2 (50)



# The CREATE Statement

- The table name
  - Call your table a name that is short and sensible.
  - Reflect the names in your design.

# Naming Tables

- Table names and column names:
  - Must begin with a letter
  - Must be 1–30 characters long
  - Must contain characters between A–Z, a–z, 0–9, \_, \$, and # (no spaces or hyphens (-) or quotation marks)
  - Must not duplicate the name of another object owned by the same user
  - Must not be an Oracle server reserved word

# Create Our two basic tables

```
CREATE TABLE Movie(  
    movieID NUMBER(4),  
    movieTitle VARCHAR2(50),  
    year number(4),  
    director VARCHAR2(50),  
    budget number(11,2),  
    profit number(11,2),  
    PRIMARY KEY (movieID)  
);
```

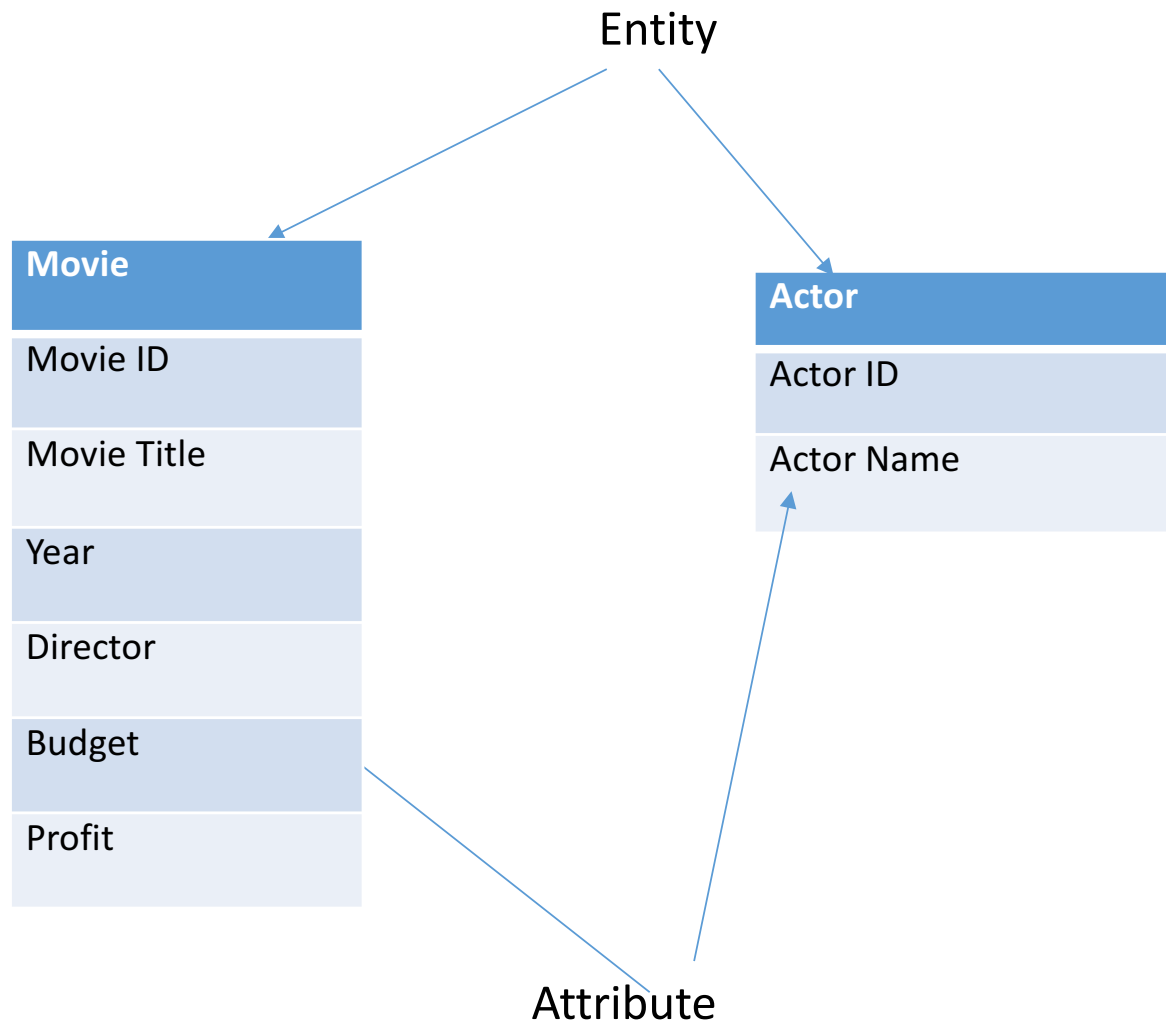
# Create Our two basic tables

```
CREATE TABLE Actor(  
    actorID NUMBER(4),  
    actorName VARCHAR2(50),  
    PRIMARY KEY (actorID)  
);
```

# Key Constraints – Primary Key

- Is a column or a set of columns that uniquely identify a specific instance of the thing a table represents
  - In Database terminology we refer to the 'thing' as an **ENTITY**
  - Each entity has a number of *attributes* which are the columns in the physical model
- Every entity in the data model must have a primary key whose values uniquely identify instances of the entity.
- Primary key enforces **entity integrity** by uniquely identifying entity instances.

# Key Constraints – Primary Key



# Key Constraints – Primary Key

- To qualify as a primary key for a column must have the following properties:
  - it must have a *non-null* value for each instance of the entity
  - the value must be unique for each instance of an entity
  - the values must not change or become null during the life of each entity instance.

# Key Constraints – Primary Key

- Let's assume that for each employee in an organization there are three candidate keys:
  - Employee ID, PRSI Number, and Name.
- Name is the least desirable candidate.
  - Even for a small company it would not be unusual for two people to have exactly the same name, more unusual for a large company.
  - There is the possibility that an employee's name could change because of marriage.
- Employee ID would be a good candidate as long as each employee were assigned a unique identifier at the time of hire.
- PRSI number could also work since every employee is required to have one before being hired.



# Defining Primary Key

```
CREATE TABLE Actor(  
    actorID NUMBER(4) PRIMARY KEY,  
    actorName VARCHAR2(50)  
);
```



# Defining Primary Key

- Defining primary key at Table-level:

```
CREATE TABLE Actor(  
    actorID NUMBER(4),  
    actorName VARCHAR2(50),  
PRIMARY KEY (actorID)  
);
```

# IMDb Example

- We have now created data structures
  - Two basic tables
- Each table represents on thing we want to store data about (Movie and Actor)
- Each table has a set of columns which defines the type of data we want to store about the thing the table represents
- Each table has a primary key defined through which we can retrieve specific rows of data
- However we do not have a mechanism to store data about a movie cast i.e. the actors who appeared in a movie

# IMDb Example

- Where should we store data about the movie cast?
- We want to minimise the amount of data stored and we do not want redundant data

# IMDb Example

- Title: Jumanji: Welcome to the Jungle
- Year: 2017
- Director: Jake Kasdan
- Budget: 90 million
- Profit: 962,077,546
- Some Cast Members:
  - Dwayne Johnson... Spencer
  - Kevin Hart ... Fridge
  - Jack Black ... Bethany
  - Karen Gillan ... Martha

# IMDb Example

- What else has Dwayne Johnson appeared in?
- [https://www.imdb.com/name/nm0425005/?ref =  
ttfc\\_fc\\_cl\\_t1](https://www.imdb.com/name/nm0425005/?ref=ttfc_fc_cl_t1)

# IMDb Example

- Suppose we include the movieID and name of the character played on the ACTOR table?
  - Our primary key is now not unique
- We would have multiple rows of data for each actor
  - One for every movie they have appeared in
    - We would need a row in the actor table
    - ActorID and actorName would be repeated on each row
      - This is wasteful and if we need to make change we have to change them all

# IMDb Example

- Suppose we include the actorID and name of the character played on the MOVIE table?
- Our primary key is now not correct
- We would have multiple rows of data for each movie
  - One for every actor that appeared in the movie
    - We would be storing the movieid, movie title, year, director, budget and profit on each row
      - This is wasteful and if we need to make change we have to change them all



# How to overcome this?

- Introduce another table
- Cast
- It has the attributes movieId, actorId and role name

```
CREATE TABLE Cast (  
    movieID NUMBER(4),  
    actorID NUMBER(4),  
    rolePlayed VARCHAR2(50)  
);
```

What should our primary key be?

# Compound Primary Key

- Suppose we decide that the combination of movieID and actorID is unique?
- We need a **COMPOUND** Primary Key

```
CREATE TABLE Cast (  
    movieID NUMBER(4),  
    actorID NUMBER(4),  
    rolePlayed VARCHAR2(50),  
    PRIMARY KEY (movieID, actorID)  
);
```

COMPOUND PRIMARY KEYS CAN ONLY BE CREATED AT TABLE LEVEL

# So as a structure we now have

- Three tables

**Movie**

**Actor**

**Cast**

**Cast** has a common data element with **Movie(movieid)**

**Cast** has a common data element with **Actor (actorid)**

We need to make sure this is expressed in the metadata to make sure we protect the integrity of our data.

# How do we define relationships between the tables?

- So what is a foreign key again?
- A column of one table that is the same as the primary key of another
- So we need to include something in our Create statement for **Cast** to indicate which attribute is the foreign key

# Foreign Keys

- We know that Cast has two foreign keys: actorID and movieID

Movie
Movie ID
Movie Title
Year
Director
Budget
Profit

Cast
Actor ID
Movie ID
Role Played

Actor
Actor ID
Actor Name

# Defining Foreign Key

To define a foreign key between Table A and Table B

When creating Table A include the following constraint as a line in the Create Statement:

```
CONSTRAINT <name of constraint> FOREIGN KEY (name  
of the column in Table A) REFERENCES (name of  
Table B) (name of column in Table B)
```

- **FOREIGN KEY:** Defines the column in the child table at the table-constraint level
- **REFERENCES:** Identifies the table and column in the parent table

# Our example

```
CREATE TABLE Cast (  
  movieID NUMBER(4),  
  actorID NUMBER(4),  
  rolePlayed VARCHAR2(50),  
  
  CONSTRAINT cast_actor_fk FOREIGN KEY  
  (actorID) REFERENCES actor (actorID),  
  CONSTRAINT cast_movie_fk FOREIGN KEY  
  (movieID) REFERENCES movie (movieID),  
  
  PRIMARY KEY (movieID, actorID));
```

# Key Constraints - Foreign Key

- Foreign keys provide a method for maintaining integrity in the data (called referential integrity) and for navigating between different instances of an entity.
- Every relationship between tables must be supported by a foreign key.
- Foreign keys attributes are indicated by the notation (FK) beside them on database models.




# Table create order

- We need to create the tables in an order
  - Movie
  - Actor
  - Cast
- WHY?

# Table create order

- First
  - Create tables that have no dependencies
- Second
  - Create tables that depend on those tables but only have one dependency
- Third
  - Create tables with multiple dependencies
- The SQL needed to create the tables is in DBI-W2-Lecture.DDL (Data Definition)

# So what is our physical model?

 This image cannot currently be displayed.

# Dropping a Table

- All data and structure in the table are deleted.
- Any pending TRANSACTION are committed.
- All indexes are dropped.
- All constraints are dropped.

```
DROP TABLE dept80;  
Table dropped.
```

# Dropping Tables

- When we create a set of SQL in a script to create and insert data we include statements to drop the tables at the start of that script
  - This will make sure our script clears out any old versions of our tables
  - If no tables exist then we will get the message

```
SQL Error: ORA-00942: table or view does not exist  
00942. 00000 - 'table or view does not exist'
```

\*Cause:

\*Action:

This is not a problem unless we are dropping something we expect to exist

# Lab class

- We are going to take a look at the spec for our lab class.
- What pieces of data do we have?
- What tables do we need?

# Peter's Pets

- Peter sells small animals (cats and small dogs), rodents (hamsters, guinea pigs and white mice) and also keeps a range of reptiles (lizards and snakes) to members of the public.
- In order to manage all the information on animal sales in Peter's Pets a database containing about animals, customers and sales of animals is needed.
- Peter wants to store for each customer their uniqueID, their name and their email address; for each animal its uniqueID and its name; and when an animal is sold, record the customer to whom it is sold and the animal sold.
- What are the tables? What are the columns and types?

# Possible Solution

- Animal (animalID, animalName)
  - Customer (custID, custName, custEmail)
  - AnimalSale (animalID, custID)
- 
- Is this the correct order to do the creates in?



# Inserting Data

# What you need to know

- The format of the table
  - i.e. the columns and their constraints.
- You can do either:
  - A full INSERT
    - This inserts a value for every column in the table.
  - A partial INSERT
    - This inserts some values, but accepts the default values for other columns.

# What you need to know

- NOTE
  - All CHAR, VARCHAR2 and DATE fields must have their values surrounded **by single quotes**.

# To add a row we need to :

```
INSERT INTO table (column1, column2, ... column_n ) VALUES  
(expression1, expression2, ... expression_n );
```

**insert into movie (movieid, movietitle, year, director,  
budget, profit)  
values(1, 'Jumanji: Welcome to the Jungle', 2017, 'Jake  
Kasden', 90000000.00, 962077546.00);**

**NOTE: If we do not include a column list then ORACLE assumes you are inserting data in the column order it expects (we may not always know this).**

# To add a row we need to :

```
INSERT INTO table (column1, column2, ... column_n )  
VALUES (expression1, expression2, ... expression_n );
```

**insert into movie (movieid, movietitle, year, director)  
values(3, 'Baywatch', 2018, 'Seth Gordon');**

**NOTE: If we use a column list we can insert the data in the order we want**

# Some errors we need to be aware of

```
insert into movie (movieid, movietitle, year, director, budget, profit)  
values(1, 'Jumanji: Welcome to the Jungle', 2017, 'Jake Kasden',  
90000000, 962077546);
```

**Error report -**

**SQL Error: ORA-00001: unique constraint (SYSTEM.SYS\_C0032346) violated  
00001. 00000 - 'unique constraint (%s.%s) violated'**

- **We've already inserted a row with a movie ID of 1.**
- **If we try to insert a second row with a movie ID with the value 1...**
- **we are trying to add a duplicate key (not allowed)**

**\*Cause: An UPDATE or INSERT statement attempted to insert a duplicate key.**

**For Trusted Oracle configured in DBMS MAC mode, you may see  
this message if a duplicate entry exists at a different level.**

**\*Action: Either remove the unique restriction or do not insert the key**

# Order to insert data

- Follow the order of create
  - If we don't and we try to insert something into a dependent table
  - Where there is no value in the related table for the foreign key then we will get the following error

```
insert into cast(actorid, movieid, roleplayed) values (5, 5,  
'Test Role')
```

Error report -

```
SQL Error: ORA-02291: integrity constraint  
(DLAWLESS.CAST_MOVIE_FK) violated - parent key not found  
02291. 00000 - "integrity constraint (%s.%s) violated - parent  
key not found"
```

\*Cause: A foreign key value has no matching primary key value.

\*Action: Delete the foreign key or add a matching primary key.

# Insert the correct number of values

- Suppose we try this insert

```
INSERT INTO movie (movieid, movietitle, year,  
director, budget, profit) VALUES(12, 'Jumanji:  
Welcome to the Jungle', 2017, 'Jake Kasden',  
90000000.00)
```

Error at Command Line : 3 Column : 73

Error report -

SQL Error: ORA-00947: not enough values

00947. 00000 - "not enough values"

\*Cause:

\*Action:

We need to provide a value for every column Oracle is expecting a value for – here we have left out the value for the profit column.

There is an equivalent error – Too many values



# TRANSACTION and SQL

- The SELECT statement reads data from the database
  - BUT doesn't change anything.
- The other three DML (data manipulation language) statements (INSERT, UPDATE, DELETE) we will use DO change the data.
- Because they change the data, the DBMS (Database Management System) goes into defensive mode when it receives one of these.
- What does this mean?
  - It will not allow other users of the database make changes to this data until the statements are complete.

# TRANSACTION in SQL

- As soon as Oracle sees an INSERT, UPDATE or DELETE, it starts a TRANSACTION
  - It LOCKS all rows that are subjected to
    - INSERT
    - UPDATE or DELETE
- The transaction continues until it receives either a
  - COMMIT
    - This makes all of the operations permanent
  - OR
  - ROLLBACK
    - This is like 'undo'. It rolls back all operations carried out by the current session since the last COMMIT

# COMMITs

- Uncommitted data cannot be seen by other sessions
- If I open a table in one SQL connection and add rows
- Then I open a second SQL connection to the same schema
  - I won't be able to see the inserts in the SECOND connection, until they are COMMITed in the FIRST.
- If I delete rows in the other one, the first one will still see them until the DELETEDs are committed

# TRANSACTION and locking

- When an INSERT, UPDATE or DELETE takes place:
  - It starts a TRANSACTION - (or logical unit of work) a sequence of SQL statements that ORACLE treats as a single unit.
  - It locks the row that is being INSERTed / UPDATED / DELETED
- Other sessions cannot see the changes that are being made
- The lock holds until the transaction session issues
  - COMMIT or
  - ROLLBACK
- A transaction ends with a:
  - commit,
  - rollback ,
  - exit, or any DDL statement which issues an implicit commit.

# The Transaction Control Commands

- Commit makes all changes since the beginning of a transaction permanent
  - Other users cannot see the results of the transaction
    - Until it has been committed.
- Rollback rolls back (undoes) all changes since the beginning of a transaction
  - Other users will be unaware of any changes.

# Our Example

- DBI-W2-Lecture-Part1.SQL

# Lab class (based on what we've done so far)

- You will be given data to insert
- What will the inserts look like (given our tables)?
- What order will the inserts be in ?
- What else do we need to include?

Alter



# Making changes to our structures

- We have director as a column (varchar2(50)) on our movie table
- We are duplicating data here
  - When we are duplicating data – generally we created another table
  - The Director table
- What do we need to do?

# Create the director table

```
CREATE TABLE Director (directorID NUMBER(4),  
directorName VARCHAR2(50), PRIMARY KEY  
(directorID));
```

# Insert some data

- So we have three directors in our data:
  - Jake Kasden
  - Rawson Marshall Thurber
  - Seth Gordon
- What are the insert statements?

# Is our movie table correct?

- How do we change it?
  - We can use the ALTER statement

# ALTER TABLE Statement

- Use the ALTER TABLE statement to:
  - Add a new column
  - Modify an existing column
  - Define a default value for the new column
  - Drop a column
  - Add a constraint to a table

# Lets think about it

- We have data already in the director column
- It is character data
  - It cannot be changed to number data just with a Modify
  - We need to do the ALTER in three stages
  - Step 1: Drop the director column on the Movie table
  - Step 2: Add a column to Movie of type Number (4)
  - Step 3: Update the data in Movie

# Drop column(s) in a table

- To drop or remove a column in an existing table:

```
ALTER TABLE table_name  
    DROP COLUMN column_name;
```

- For example:

```
ALTER TABLE Movie  
    DROP COLUMN director;
```

# Adding column(s) to a table

- To add a column to an existing table:

```
ALTER TABLE table_name  
    ADD column_name column-definition;
```

```
ALTER TABLE Movie  
    ADD directorID Number(4);
```



# Now we need to protect the Movie table

- We need to make sure you cannot insert data into the Movie table for directors that are not in the Director table
- How do we do this?

# ALTERING constraints

- To use ALTER to change constraints, you may:
  - Add a constraint
  - Drop a constraint
  - Enable a constraint
    - This happens automatically when the constraint is added.
  - Disable a constraint
    - May need to do this sometimes to manage inserts/updates without enforcing table order

# Altering to add constraints

```
ALTER TABLE table_name  
ADD CONSTRAINT constraint_name CHECK  
(column_name condition);
```

- E.g.

```
ALTER TABLE Movie ADD CONSTRAINT  
movie_direct_fk FOREIGN KEY  
(director) REFERENCES director  
(directorid);
```

# DDL

- DBI-W2-Alter.DDL

# Now we need to insert data into the director table

```
INSERT INTO director (directorid,  
directorname) values (1, 'Jake Kasden');
```

```
INSERT INTO director (directorid,  
directorname) values (2, 'Rawson Marshall  
Thurber');
```

```
INSERT INTO director (directorid,  
directorname) values (3, 'Seth Gordon');
```

# How do we Update the director table?

# Updating Existing Table Records

- UPDATE:
  - Updates field values in one or more records in a table
  - Only one table may be updated at a time

**UPDATE** tablename

**SET** field1= new\_value1, field2 =  
new\_value2, ...

**WHERE** search condition;

# UPDATE

UPDATE <table>

SET col1 = val1

[,col2 = val2...]

[WHERE

<condition>]

- All rows where the condition is true have the columns set to the given values
- If no condition is given all rows are changed so BE CAREFUL
- Values are constants or can be computed from columns



# Updating the Movie Table

```
Update Movie set directorid=1 where  
movieid=1;
```

```
Update Movie set directorid=2 where  
movieid=2;
```

```
Update Movie set directorid=3 where  
movieid=3;
```

```
COMMIT;
```

# Lab Class

- You will be asked to create a new table Species after you have created the other tables
  - You will need to create the Species table
  - Add a column to the Animal table
  - Add a foreign key to the Animal table
  - Insert data into the Species Table
  - Update the Animal table with species data

# Accessing Data

# Accessing the data in a databases

- Where the data is in the database?
  - You need to know the table structure
  - What the data means?
  - How each row is identified uniquely?
- What items of data you need?
  - What parts of each row in the table is needed
  - You need to know the constraints. E.g
    - 'I only want the items purchased in the last month'
    - 'I only want to bill for items not paid for'

# Accessing the data in a database

- What format it is in?
  - What datatypes are used?
- What you want to know or calculate from the data?
  - Are you looking at unit prices and an amount used ? Do you need to calculate a total?
- How it should be presented?
  - Do you want to give the data a different name?
  - Do you want to present it in a different format? E.g. rounding up to the nearest euro?

# The SELECT Statement

- This is the most powerful and complex of the SQL statements.
- Its structure is reasonably simple:

```
SELECT columns FROM tables  
WHERE condition;
```

There are other rules that can be added later.

# Find me everything

- Select statement format:

**SELECT** **columns** **FROM** **tables** **WHERE** **conditions**;

- To select all rows and columns from a table called MOVIE

**SELECT** **\*** **FROM** **Movie**;

\* denotes all available columns

# Find me only specific pieces of information

- Select statement format:

```
SELECT  columns  
FROM  tables WHERE conditions;
```

- To select only the column called title from MOVIE

By picking one single column we have taken a **PROJECTION** from the table

```
SELECT  movietitle FROM  MOVIE;
```





# The Where Clause

- Allows use to restrict output based on data matching some criteria

# Basic Operators to build expressions

Operator	Explanation and Example	Syntax
Comparisons	<, =, >, <=, >=, <> are applicable to all of the built-in types.	<b><i>Expression</i></b> { <   =   <=   >=   <> } <b><i>Expression</i></b>
AND, OR, NOT	Evaluate any operand(s) that are boolean expressions (orig_airport = 'SFO') OR (dest_airport = 'GRU') -- returns true	{ <b><i>Expression AND Expression</i></b>   <b><i>Expression OR Expression</i></b>   <b><i>NOT Expression</i></b> }

# Example

- Find the title and budget of all movies with a Movie ID <3?
- And a director with a DirectorID <2?
- What happens if we change the AND to OR?
- Find the title and budgets of movies not directed by directorid 2?

# Boolean Expressions

Operator	Explanation and Example	Syntax
BETWEEN	<p>Tests whether the first operand is between the second and third operands. The second operand must be less than the third operand.</p> <p>Applicable only to types to which <code>&lt;=</code> and <code>&gt;=</code> can be applied.</p>	<p><b><i>Expression [ NOT ] BETWEEN Expression AND Expression</i></b></p>

# Exercise

- Find the titles of all movies with a budget between 100000000 and 900000000.00

# Boolean Expressions

Operator	Explanation and Example	Syntax
IN	<p>Operates on table subquery or list of values. Returns TRUE if the left expression's value is in the result of the table subquery or in the list of values. Table subquery can return multiple rows but must return a single column.</p> <p><b>SELECT * FROM employees WHERE job_id IN ('PU_CLERK','SH_CLERK');</b></p>	<p><b>{ <i>Expression</i> [ NOT ] IN <u><i>TableSubquery</i></u>   <i>Expression</i> [ NOT ] IN ( <i>Expression</i> [, <i>Expression</i> ]* )</b></p>

# Boolean Expressions

Operator	Explanation and Example	Syntax
LIKE	<p>Attempts to match a character expression to a character pattern, which is a character string that includes one or more wildcards.</p> <p>% matches any number (zero or more) of characters in the corresponding position in first character expression. <b>city LIKE 'S%' – all cities starting with S</b></p> <p>_ matches one character in the corresponding position in the character expression. Any other character matches only that character in the corresponding position in the character expression. <b>city LIKE '_sant' – all cities starting with any character followed by sant</b></p>	<b><i>CharacterExpression [ NOT ] LIKE CharacterExpression WithWildCard [ ESCAPE 'escapeCharacter']</i></b>

# Exercise

- Remember % is the wildcard symbol
- Find the names of all movies starting with B?
- Find the names of all movies with a letter a somewhere in their name?
- And does not start with a J?



# The WHERE Clause

- When we filter the data we use the WHERE CLAUSE
- By using a WHERE Clause we have implemented a **RESTRICTION** on the table
- In the WHERE Clause we use condition(s) using our operators to restrict

# Changing the output

- We can change the way the output will be presented to the user e.g.

```
select movietitle "Title", DirectorID "Director"  
from MOVIE  
where DIRECTORID BETWEEN 1 AND 3;
```

# The Techniques We Have Used

- Projection – filtering out unwanted columns.
- Renaming – using alternate titles
- Restriction – filtering out unwanted rows using conditions

# SQL

- DBI-W2-Lecture-PartII.SQL

# Lab Class

- You will build some simple select statements

# References

Lawless, Deirdre (2018) Databases 1 slides and resources, TU Dublin.