

MODULE 2

WEEK 1 REVIEW

DAY 1: RELATIONAL DATABASES


- Storing data so it can be used later is called **persisting** data.
- **Relational databases** allow data to be accessed and reassembled in many different ways by **Relating** the data rather than reassembling it.
- Real world problems are modeled into **Entities** such as users, posts comments, products, etc.
- A **Relational Database Management System (RDBMS)** helps manage a relational database.

TABLE (Entity)

COLUMN (Attribute)

Product

ROW
(Record or
Instance)



ID	BRAND	MODEL	DESCRIPTION	PRICE
1453672	DELL	Vostro 5590	Great laptop.	679.00
1453673	APPLE	MacBook Pro	13" 2020 2 Thunderbold 3 Ports	1299.00

DAY 1: DATABASE COLUMNS

Each column in a database has a data type.

ANSI SQL (Structured Query Language):

The standard for SQL that is defined by ANSI (American National Standards Institute).

ANSI SQL defines many common data types:.

Examples:

- char, varchar, nvarchar
- int, decimal, bigint
- boolean/bit
- Datetime

https://en.wikipedia.org/wiki/SQL#SQL_data_types

DAY 1: INTRO TO QUERIES

SQL stands for **structured query language** and is a **declarative programming language** to retrieve and update records from a database.

SQL consists of:

- **Data definition language** to define the data structures
- **Data manipulation language** to query and modify the data in a database
- **Data control language** to define access to a particular database

DAY 1: INTRO TO QUERIES

- The **SELECT** clause indicates what columns to get from a database table.
- The **FROM** clause indicates which table(s) to retrieve the data from.
- The **WHERE clause** is used to filter the result set using one or more criteria rules.

```
SELECT name FROM city WHERE countrycode = 'USA';
```

DAY 1: INTRO TO QUERIES

- Conditional clauses in the **WHERE clause** can include
 - **=, <>, !=, >, >=, <, <=**
 - **IN(values), NOT IN(values)**
 - **BETWEEN value AND value**
 - **IS NULL, IS NOT NULL**
 - **LIKE (with wildcard character)**

DAY 1: INTRO TO QUERIES

- The **DISTINCT clause** indicates that duplicate values should not be included
- The **AS clause** establishes an alias for a particular column name

```
SELECT DISTINCT name AS city_name FROM city  
WHERE countrycode = 'USA';
```


DAY 2: ORDERING DATA

A result set can be sorted using the **ORDER BY** syntax

- Sort columns must exist in the table being queried or can be aliased columns
- Multiple column names can be provided which assigns a priority sort.
- Each column in **ORDER BY** clause can be specified as ascending (**ASC**) or descending (**DESC**). If not specified the default is **ASC**.

```
SELECT name, continent, population FROM country  
ORDER BY continent, population DESC;
```

DAY 2: LIMITING RESULTS

We can reduce the size of our result set to N results. By using **LIMIT N** at the end of a query. Where **N** is the result size.

```
SELECT name FROM country LIMIT 10;
```

DAY 2: STRING OPERATIONS

We can concatenate the values across multiple columns into a single field using the `||` operator to concatenate strings.

```
SELECT name || ', ' || district FROM city WHERE  
countrycode='USA';
```

DAY 2: STRING OPERATIONS

We can use the **AS** keyword to give the concatenated column a name.

```
SELECT name || ', ' || district AS city_state FROM  
city WHERE countrycode='USA';
```

DAY 2: GROUPING RESULTS

GROUP BY statements group records into summary rows and return one record for each group.

- The **GROUP BY** clause can be used in conjunction with a **SELECT** statement and aggregate functions to collect data across multiple records.

```
SELECT continent, SUM(population) FROM country  
GROUP BY continent
```

DAY 2: AGGREGATE FUNCTIONS

- **AVG** returns the average value of a numeric column
- **SUM** returns the total sum of a numeric column
- **COUNT** returns the number of rows matching criteria
- **MIN** returns the smallest value of the selected column
- **MAX** returns the largest value of the selected column

DAY 2: COUNT(COLUMN) VS. COUNT(*)

It's important to be aware of the **difference between COUNT (*) and COUNTing a specific field.**

If you specify a column name to count, only the rows in the table that have a value for that column will be returned. For instance, in our world data, this query returns 192 rows, while changing it to use **COUNT (*)** instead returns 239. This is because only 192 rows have a value (that is do not have a **NULL** value) for **indepyear**.

```
SELECT COUNT(indepyear) FROM country;
```

DAY 2: SUBQUERIES

A **subquery** is referred to as an inner query and can provide the results of one query as input to another.

- Often used in the WHERE clause
- Can only return one item in the SELECT clause when using = in main query so you must be sure you will only get **one result** or use **IN** in your **WHERE** clause.

```
SELECT name, countrycode FROM city where  
countrycode IN (SELECT code FROM country WHERE  
continent='Europe') ;
```


DAY 2: SUBQUERIES

SQL is a declarative language and does *not* run from top to bottom and left to right. The order it runs is:

1. **FROM** clause - The database needs to know which table you're selecting from first of all
2. **WHERE** clause - The database then needs to know which rows you'll work with
3. **GROUP BY** clause - The database then groups those rows according to your GROUP BY clause
4. **SELECT** clause - The database then collapses those rows down and selects the columns that you want data from
5. **ORDER BY** clause - The database orders the rows in the order that you ask for
6. **LIMIT / TOP** clause - The database only returns the number of resulting rows that you want

DAY 3: KEYS

Primary Keys:

- Uniquely identify records in a table.
- Leveraged to allow us to define relationships between tables.

Natural Primary Keys:

- Use a piece of table data that is unique for each record.

Surrogate Primary Keys:

- Use a generated unique identifier when the data does not contain a natural one.

DAY 3: KEYS

Composite Primary Keys:

- A primary key made up of multiple fields.

Foreign Key:

- A field that references a primary key in another table.
- Allows enforcement of data integrity.

DAY 3: CARDINALITY

One-To-One (1:1)

- One row in table A relates to one row in table B.
- Example: Each record in Person table has one corresponding record in SSN (Social Security Number) table.

One-To-Many (1:N OR 1:M)

- One row in table A may relate to multiple rows in table B.
- Example: Each record in Address table may related to multiple records in Person table.

Many-To-Many (M:N or N:M)

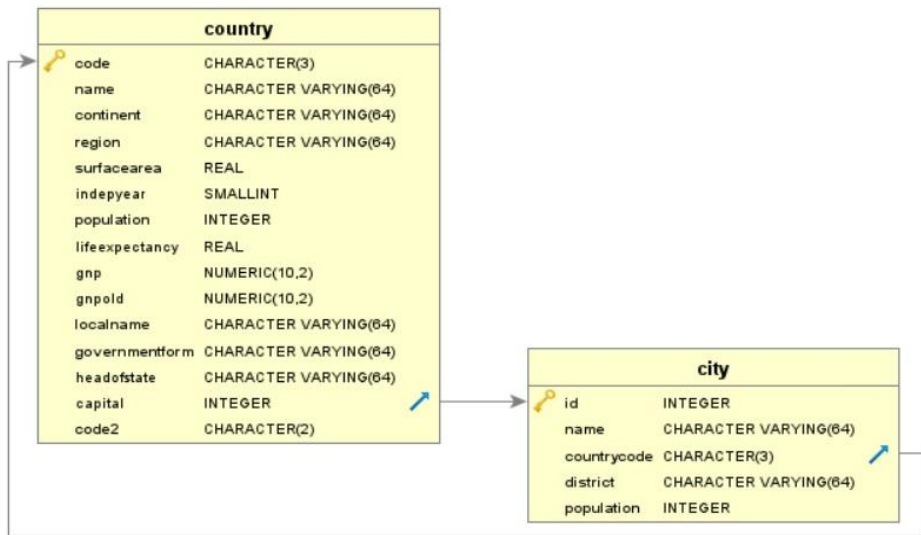
- Many rows in table A may relate to many rows in table B.
- Each record in Film table may relate to multiple records in Actor table and each record in Actor table may relate to multiple records in Film table.
- Implemented via join tables.

DAY 3: JOINS

Joins allow us to relate data between tables to query data in whatever ways makes sense.

We could relate data from the country and city tables to provide one set of data.

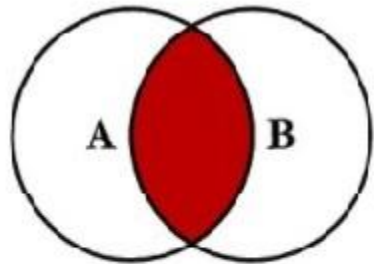
To get a city's country we would join the **city** table's **countrycode** field to the **country** table's **code** field.



DAY 3: JOINS

INNER JOINS

Inner joins allow us to query data that is the intersection of two tables.



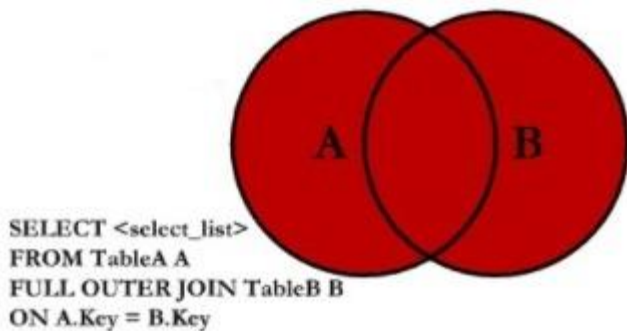
```
SELECT <select_list>  
FROM TableA A  
INNER JOIN TableB B  
ON A.Key = B.Key
```

DAY 3: JOINS

OUTER JOINS

When performing an Inner Join, rows from either table that are unmatched in the other table are not returned. In an outer join, unmatched rows in one or both tables can be returned. There are a few types of outer joins.

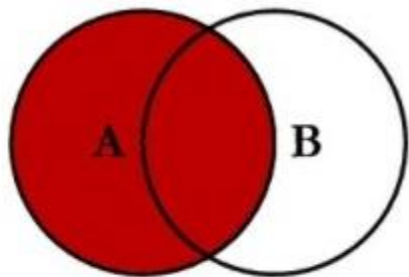
A Full Outer Join returns the data from both tables, including unmatched data.



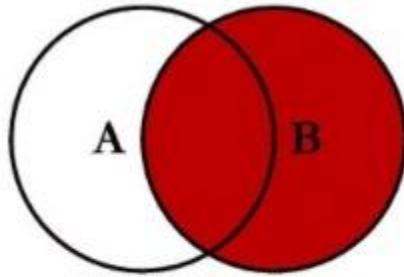
DAY 3: JOINS

LEFT AND RIGHT JOINS

Left and Right Outer Joins allow us to include unmatched data from either the “Left” or “Right” table data. Left and Right refer to the table’s position in the from/join statement. Left and Right Outer Joins are usually referred to as Left and Right Joins.



```
SELECT <select_list>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.Key = B.Key
```

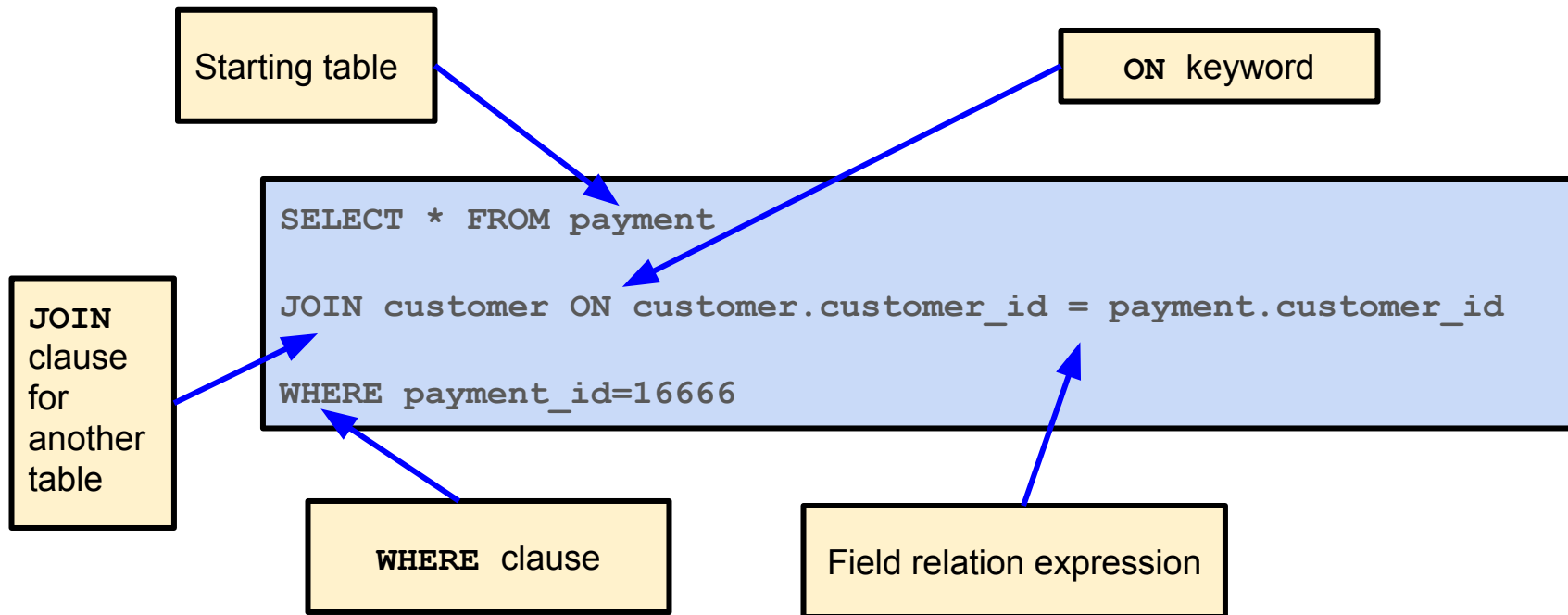


```
SELECT <select_list>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.Key = B.Key
```

- Join is shorthand for Inner Join, which is the default.
- Left and Right Joins are shorthand for Left and Right Outer Joins. This is because Inner Joins do not have the concept of Left and Right so saying Left or Right implies an Outer Join.

DAY 3: JOINS

ANATOMY OF A JOIN STATEMENT



DAY 3: UNIONS

A SQL Union:

- Combines the results of two or more queries into a single result set.
- The number of columns involved as well as the data types for those columns in each query **MUST BE THE SAME.**
- Duplicate rows are removed

```
SELECT first_name FROM actor  
  
UNION  
  
SELECT first_name FROM customer
```

DAY 4: INSERT

```
INSERT INTO table_name (column1, column2, ...,  
column_n) VALUES (value1, value2, ... value_n);
```

- This form specifies columns as well as their values.
- Columns can be in an order since the column names are specified.

```
INSERT INTO table_name VALUES (value1, value2, ...  
value_n);
```

- This form specifies only values.
- Values for all columns must present in the order the columns appear in the database

DAY 4: INSERT USING SELECT

```
INSERT INTO table_name (column1, column2,  
..., column_n) [SELECT STATEMENT]
```

Example:

```
INSERT INTO countrylanguage (countrycode, language, isofficial,  
percentage)  
SELECT countrycode, 'Klingon', false, .1 FROM countrylanguage  
WHERE language='English';
```

DAY 4: UPDATE

```
UPDATE table_name  
SET column1 = value1, column2 = value2, ...  
WHERE condition;
```

- Can use subquery in WHERE clause.

Example:

```
UPDATE client SET last_name = 'Butters' WHERE client_id =  
(SELECT client_id FROM client WHERE first_name='Johnny');
```

DAY 4: DELETE

```
DELETE FROM table_name  
WHERE column=value;
```

Don't forget the WHERE clause!!!

Doing so can have *disastrous* consequences.

DAY 4: CRUD OPERATIONS

Create: INSERT

Read: SELECT

Uppdate: UPDATE

Delete: DELETE

DAY 4: REFERENTIAL INTEGRITY

Referential integrity ensures that relationships between tables remain consistent.

- We enforce referential integrity and other rules by applying constraints to our tables.

DAY 4: TABLE CONSTRAINTS

A **constraint** is associated with a table and defines properties that the column data must comply with.

NOT NULL	Requires data in column.
UNIQUE	Requires data in column to be unique.
PRIMARY KEY	Allows FKs to establish a relationship, and enforces NOT NULL and UNIQUE.
FOREIGN KEY	Enforces valid PK values, and limits deletion of the PK row if FK row exists.
CHECK	Specifies acceptable values that can be entered in the column.
DEFAULT	Provides a default value for the column.

DAY 4: TRANSACTIONS

If we transfer money from one bank account to another and there's a failure depositing it after it withdraws, we wouldn't want our account to be out the money too and we would want the withdrawal from the original account to be reverted.

Even though there are two steps to the process (withdrawing and depositing), the process functions as a whole and must either complete successfully or be undone. In database-speak, this would be a **transaction**.

DAY 4: THE ACID TEST

The ACID test can be used to determine whether a series of actions should be written as a transaction. A scenario implemented as a transaction should have the following characteristics:

- **Atomicity**: Within a transaction, a series of database operations all occur or none occur.
- **Consistency**: The completed transaction leaves things remaining in a consistent state at the end. Any rules in place before the transaction still pass after the transaction.
- **Isolation**: Ensures that the concurrent execution of a transaction results as if the operations were executed serially.
- **Durability**: Once a transaction has been committed it will remain so, even during a power loss, crash, or an error.

DAY 4: TRANSACTIONS IN POSTGRES

- We can start a transaction by using **START TRANSACTION** before a set of SQL statements.
- Once we have completed all the statements successfully, we can **COMMIT** the transaction.
- If something goes wrong or we change our mind BEFORE committing, we can **ROLLBACK** the transaction.