
LUT
University
LAND
OF THE
CURIOUS

CT60A4304 - BASICS OF DATABASE SYSTEMS

RELATIONAL MODEL AND SQL

Lecture

Jiri Musto, D.Sc.

TABLE OF CONTENTS

RELATIONAL MODEL: STRUCTURE

- Each row is a tuple
- The order of rows or columns does not matter
- Each row has to be distinguishable
- Attributes can be atomic or multivalued
  - A relation with a multivalued attribute is unnormalized

Attribute(s) <*A,a*>
A feature of a relation

Header *H*
The names of all attributes

Tuple(s)
The set of attributes named by the header

id	partNmb	projNmb	count
1	2	5	12
2	5	8	51
3	3	11	62
4	1	2	12

Relation(s) <*H,h*>
The set of tuples and a header

RELATIONAL MODEL: KEYS

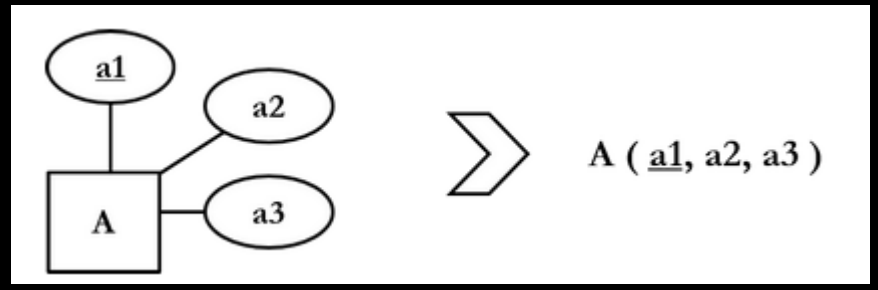
- Superkey:
  - A (set of) attribute(s) that uniquely identifies an entity
  - There can be multiple superkeys
- Candidate key:
  - Minimal super key
- Primary key:
  - The candidate key chosen for identification
  - Candidate keys not chosen as primary key can be called ‘alternate keys’
- Key attribute:
  - Atomic subset of the attribute(s) of primary key (may be the same as primary key)
- Foreign key:
  - The primary key of another relation

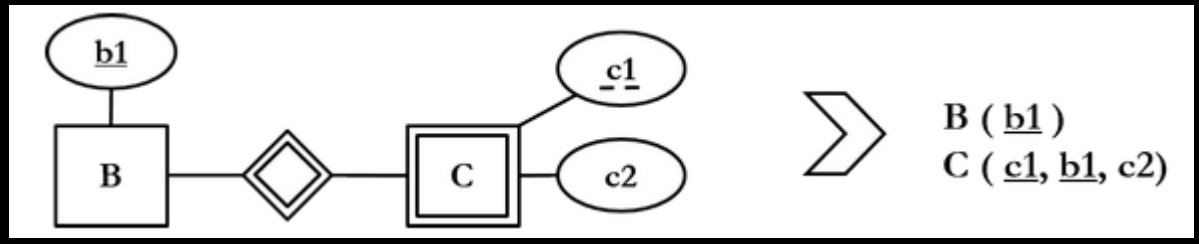
TRANFORM ER MODEL TO A RELATIONAL MODEL

- There are four different steps for transforming an ER model (or another conceptual model) to a relational model
  1. Transforming entities
  2. Transforming attributes
  3. Transforming relationships
  4. Transforming abstractions
- Each step has its own rules for transformation

TRANSFORMING ENTITIES

- Rule 1: Each strong entity is made into a relation. Key attributes are used to create the primary key
- Rule 2: Each weak entity is made into a relation similarly to Rule 1. The primary key is formed using the key attributes of the weak and strong entity

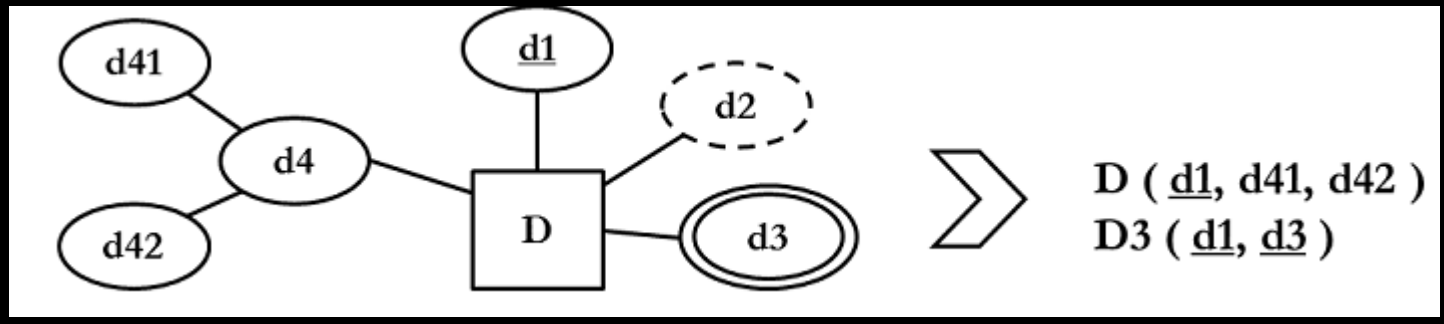




https://tim.jyu.fi/view/kurssit/tktl/itka204/kurssimoniste

TRANSFORMING ATTRIBUTES

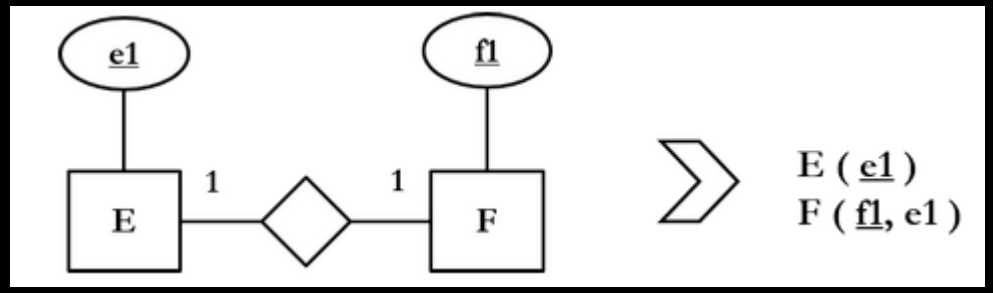
- Rule 3: Each derived attribute is discarded
- Rule 4: Each composite attribute is converted into simple attributes. Composite attribute is discarded
- Rule 5: Each multivalued attribute is transformed into a connecting relation. The primary key for the connecting relation will be the multivalued attribute as well as the key attribute the parent relation.

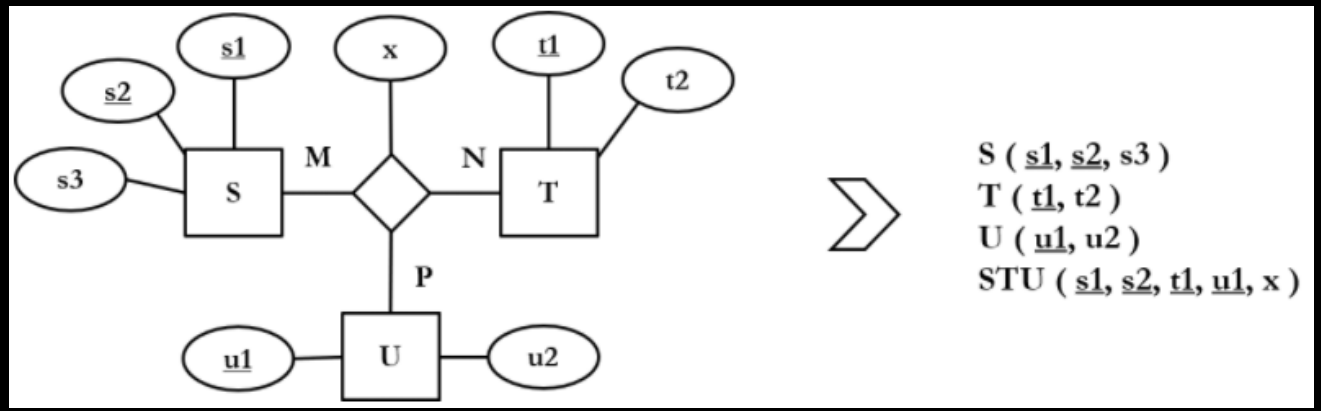


https://tim.jyu.fi/view/kurssit/tktl/itka204/kurssimoniste

TRANSFORMING RELATIONSHIPS

- Rule 6: In each one-to-one relationships, one of the relations will have a foreign key
- Rule 7: In each one-to-many relationships, the foreign key is place on the side of ‘many’
- Rule 8: In each many-to-many relationships, a linking/interim relation is formed that stores foreign keys to both relations as well as attributes related to the relationship
- Rule 9: Each n-degree relationships are transformed using Rule 8.





https://tim.jyu.fi/view/kurssit/tktl/itka204/kurssimoniste

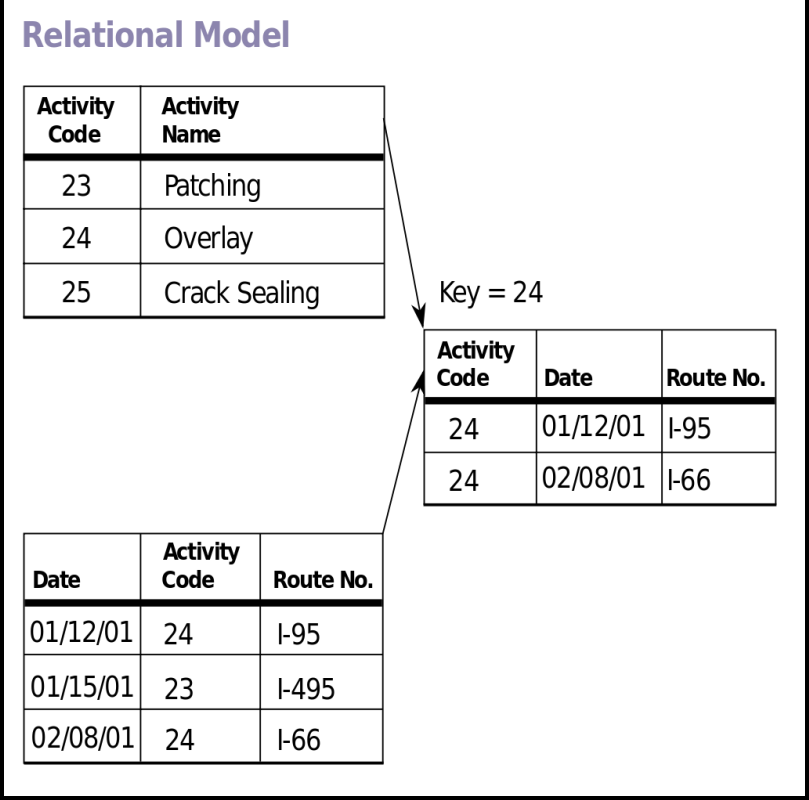
TRANSFORMING ABSTRACTIONS

- Four different ways:

1. Each parent and child entity is made into a relation.
2. Each child entity is made into a relation. The attributes of the parent and child are chosen for the relation.
3. Make one relation that has all the attributes of the parent and child relation as well as an additional attribute to imply the roles of tuples in the relation
4. Make one relation as in #3 but instead of an extra attribute, use Boolean flags to imply the roles of the tuples

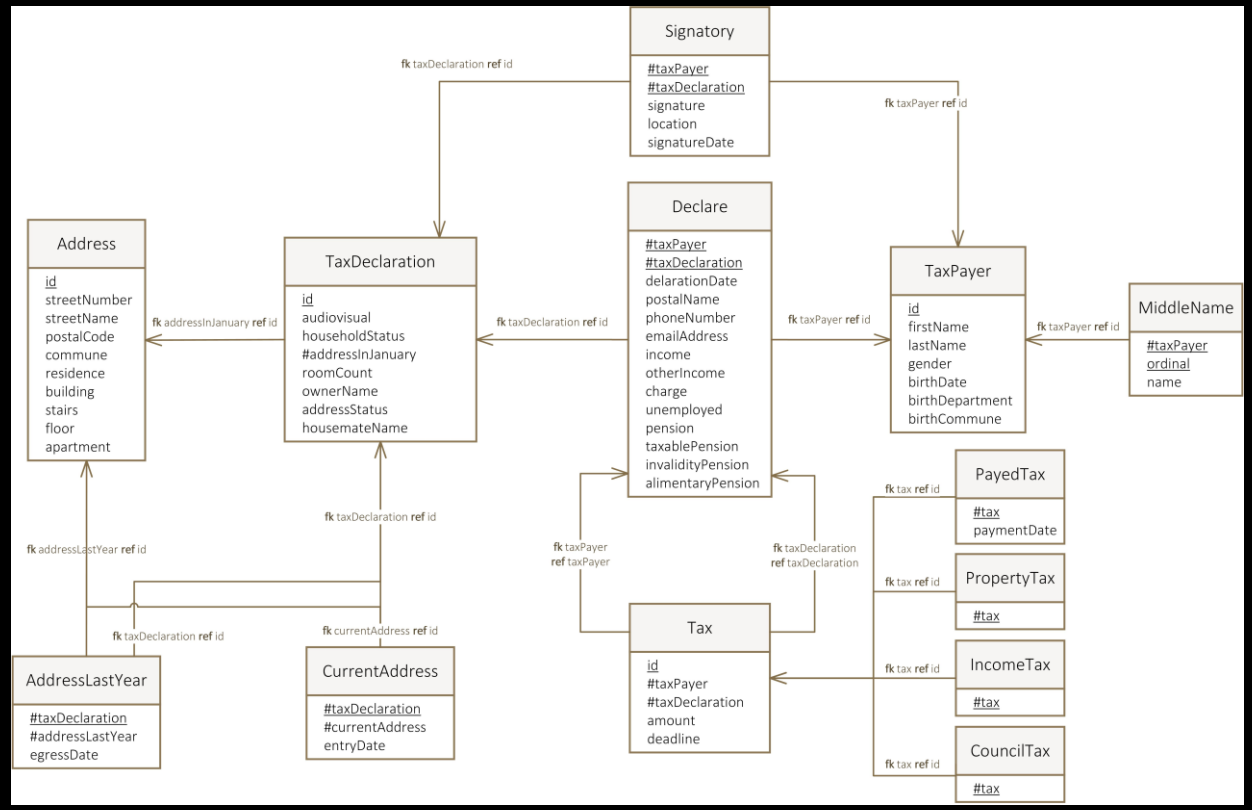
RELATIONAL MODEL EXAMPLES

Strict relational model



https://fi.wikipedia.org/wiki/Tiedosto:Relational_Model.svg

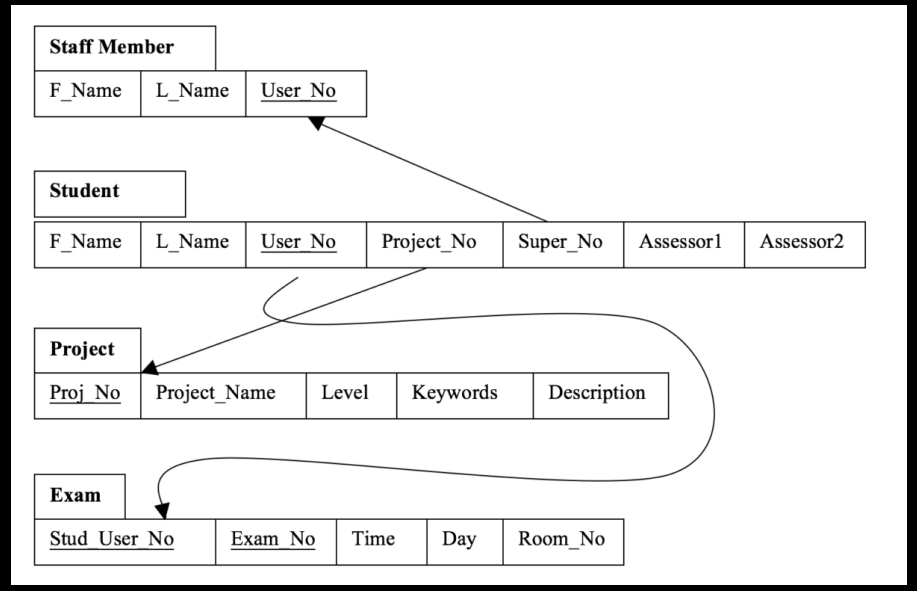
Relational database model / relational schema



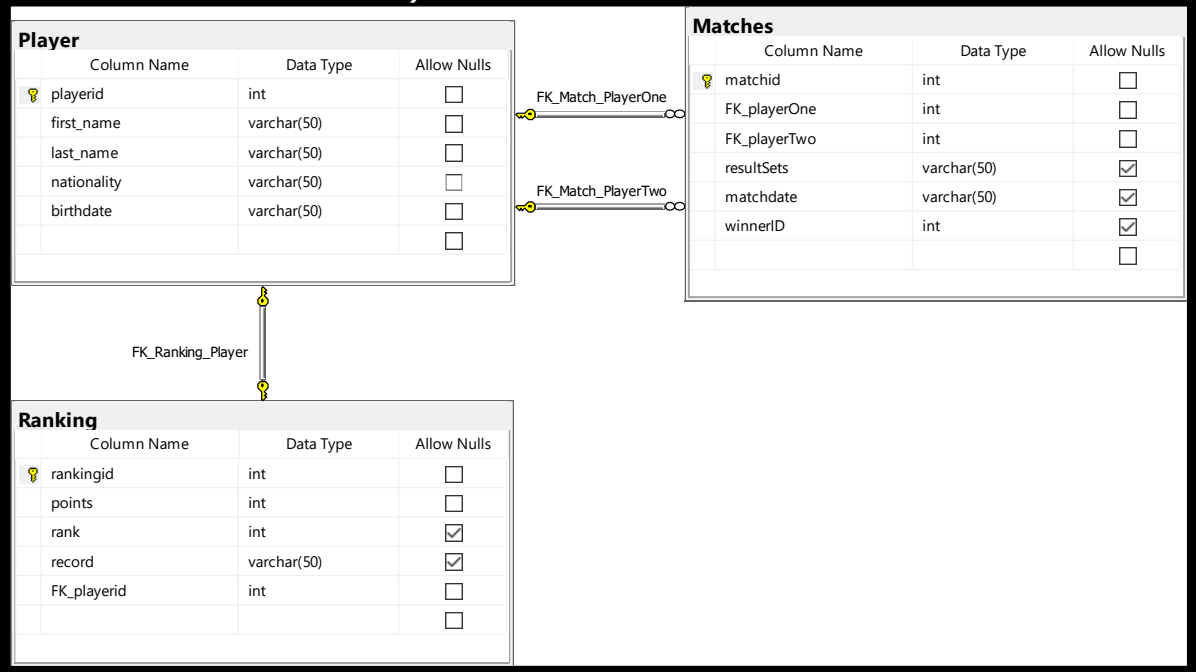
https://commons.wikimedia.org/wiki/File:Dbdesign.relational.nonull.jpg

RELATIONAL MODEL EXAMPLES

Relational schema



Relational database schema/diagram (still a valid relational model)



https://www.carnaghan.com/relational-student-database-report/

MAIN DIFFERENCES BETWEEN ER MODEL AND RELATIONAL MODEL

- Relational model has foreign keys, ER model just implies their existence
- ER model has N:M cardinalities, these cannot be implemented in a relational model
  - There has to be a foreign key reference
  - N:M is solved with interim/linking tables
- ER model is supposed to be a higher-level conceptual model
- Relational model should be possible to be implemented in a relational database as-is
  - Data types can be shown in the model

RELATIONAL ALGEBRA

- Similar to set theory / algebra of sets
- Not a necessity in database design
  - Useful when designing extremely large relational databases for theoretical testing
- Many relational algebra operations can be transformed to SQL queries
- There are eight basic operations

RELATIONAL ALGEBRA

Operation	Formula	Explanation	SQL comparison
Projections	$\pi a1..an(R)$	Project the desired attributes from the relation to the result	SELECT without WHERE
Selection / Restriction	$\sigma_{a\theta v}$ (R)	Select tuples that comply with the given conditions	SELECT with WHERE
Union	R∪S	Combine the tuples of two relations, duplicates are removed	UNION
Intersection	R∩S	Select tuples that are in both relations	Reference key
Difference	R−S	Select tuples of R that are not in relation S	NOT IN
Join	$R \bowtie_{a\theta b} S$	Join relations that fills the given condition	INNER JOIN
Division	$R \div S$	Make a result relation that includes attribute headers from R that are not in S	No direct method
Cross join / Cartesian product	$R \times S$	Cartesian product of two relations, combine all tuples and attributes.	SELECT FROM without where

CT60A4304 - BASICS OF DATABASE SYSTEMS

STRUCTURED QUERY LANGUAGE

Lecture

Jiri Musto, D.Sc.

SYNTAX AND CONVENTIONS

Syntax	Why, what it does
Uppercase keywords	Not mandatory, a convention left from old times (easier to separate keywords from tables/columns)
.	Dots are used in the following: [table name].[column name]
;	Semicolon marks the end of the query
‘’	Single quotes used for strings
“”	Double quotes are generally not used (may work in some DBMS). If used, double quotes are used for database identifiers (“table name” “column name”
*	Used to mark ALL, has other usages in some DBMS
%	Wildcard for any number of characters (%sea)
_	Wildcard for a single character (st_dent)
--	Write comments in SQL files

SELECT AND FROM

- SELECT – Choose the columns to show in the results
- FROM – Specify the table(s) you are targeting with the statement

```
--Basic SELECT query
SELECT [column] FROM [table];
--Show all columns in the table [Entertainment]
SELECT * FROM Entertainment;
--Show names from the table [Employee]
SELECT Name FROM Employee;
--Show distinct values
SELECT DISTINCT startYear FROM Director;
--Select multiple columns from multiple tables
SELECT name, year_won FROM Employee, PrizesWon;
```

entertainmentID	entertainmentType	name	year_released
1	1	Iron Man	2008
2	1	The Avengers	2012
3	1	Captain America, Civil War	2016
4	1	Doctor Strange	2008
5	1	Thor: Rahnarök	2008
6	1	Avengers: Endgame	2008
7	1	Spider-Man: No Way Home	2008
8	2	The Witcher	2019
9	2	Sherlock	2010
10	2	The Office	2005
11	2	Doctor Who	2005
12	2	The Walking Dead	2010

	Name
4	Gary Halvorson
5	Joss Whedon
6	Henry Cavill
7	Robert Downe...
8	Tom Holland
9	Eiichiro Oda
10	Eric Kripke
11	David Crane
12	Seth MacFarlane
13	Benedict Cumb...

	startYear
1	1976
2	1989
3	1995
4	1997

INSERT INTO

- Add data into database
- Can do mass inserts
- Can be used to copy data from one table to another using SELECT

```
--Single insert, all columns
--[startYear, filmsWritten, episodesWritten]
INSERT INTO Writer VALUES (2006, 20, 45);
--Single insert, specific columns
--[table name]([column names])
INSERT INTO Writer (startYear) VALUES (2022);
--Mass insert
INSERT INTO Writer VALUES
    (2010, 10, 11),
    (2015, 7, 15),
    (2017, 2, 5);
--Copy data using INSERT and SELECT
INSERT INTO Animation (seasons, episodes)
SELECT seasons, episodes FROM Series;
```

UPDATE

- Update existing data in the database
- Can update multiple rows

```sql
--Update data
UPDATE [table] SET [column = value];

--Single row update based on primary key
UPDATE Employee SET name = 'Arnold Schwarzenegger'
WHERE employeeID = 14;

--Multirow update based on column value
UPDATE Series SET on_goign = 'false' WHERE on_going = 'true';
```

DELETE

- Remove data from database
- Empties the whole table if no filter is specified

```
--Remove data
DELETE FROM [table];
--Delete everything from [Employee]
DELETE FROM Employee;
```

WHERE

- Used to filter other queries (SELECT, DELETE, UPDATE, and INSERT copying)

```
--Filter data, combine with SELECT, DELETE, INSERT and UPDATE
WHERE [column = value];
SELECT * FROM Entertainment WHERE name = 'Iron Man';
DELETE * FROM Writer WHERE filmsWrittenr = 0;
UPDATE Movie SET box_office = 1500000 WHERE entertainmentID = 10;
```

	entertainmentID	entertainmentType	name	year_released
1	1	1	Iron Man	2008

AND OR

- Additional conditions for filtering
- Remember to use brackets if necessary

```sql
--WHERE [condition] AND [condition] OR [condition]
SELECT * FROM Series
WHERE seasons = 10 AND episodes = 30 OR on_going = 'false';
--Remember brackets
SELECT * FROM Series
WHERE seasons = 10 AND (episodes = 30 OR on_going = 'false');
```

	entertainmentID	on_going	seasons	episodes	year_ended	type	entertainmentType
1	9	false	4	15	2017	Crime, Mystery	2
2	10	false	9	188	2013	Comedy	2
3	12	false	11	177	2022	Drama, Horror	2
4	13	false	10	235	2004	Comedy, Romance	2
5	14	false	15	327	2020	Drama, Fantasy	2

	entertainmentID	on_going	seasons	episodes	year_ended	type	entertainmentType
1	13	false	10	235	2004	Comedy, Romance	2

AS

- Rename columns or tables
- Specify names used within query (like variables)
- Rename headers for the result

```
--Rename columns and tables. Can be omitted
SELECT year_released AS 'Year released' FROM Entertainment;

SELECT * FROM Writer AS W1, Actor
WHERE W1.employeeID = Actor.employeeID;
```

	Year released
1	2008
2	2012
3	2016
4	2008
5	2008
6	2008
7	2008
8	2019
9	2010
10	2005
11	2005
12	2010
13	1994
14	2005
15	1999
16	1989
17	2013
18	1999
19	2013

	employeeID	startYear	filmsWritten	episodesWritten	employeeID	numberOfLeadRoles	numberOfSideRoles	startYear
1	12	1995	7	823	12	25	32	1987

JOIN

- Combine another table, most used with FOREIGN KEYS (but not required)
- Different types of joins: INNER and OUTER (LEFT, RIGHT, FULL)
  - INNER JOIN is used most often and works in all relational databases
  - All OUTER JOIN clauses may not work in all relational databases

```sql
--Join
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
INNER JOIN Actor ON Employee.employeeID = Actor.employeeID
INNER JOIN Writer ON Employee.employeeID = Writer.employeeID;
```

name	filmsWritten	numberOfLeadRoles
Seth MacFarlane	7	25

JOIN

```
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
LEFT JOIN Actor ON Employee.employeeID = Actor.employeeID
LEFT JOIN Writer ON Employee.employeeID = Writer.employeeID;
```

name	filmsWritten	numberOfLeadRoles
Scott Derrickson	NULL	NULL
Paul Feig	NULL	NULL
Robert Singer	NULL	NULL
Gary Halvorson	NULL	NULL
Joss Whedon	13	NULL
Henry Cavill	NULL	7
Robert Downey Jr.	NULL	37
Tom Holland	NULL	9
Eiichiro Oda	10	NULL
Eric Kripke	6	NULL

```
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
RIGHT JOIN Actor ON Employee.employeeID = Actor.employeeID
RIGHT JOIN Writer ON Employee.employeeID = Writer.employeeID;
```

name	filmsWritten	numberOfLeadRoles
Henry Cavill	NULL	7
Robert Downey Jr.	NULL	37
Tom Holland	NULL	9
Seth MacFarlane	7	25
Benedict Cumberbatch	NULL	44
NULL	13	NULL
NULL	10	NULL
NULL	6	NULL
NULL	2	NULL
Seth MacFarlane	7	25

```
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
RIGHT JOIN Actor ON Employee.employeeID = Actor.employeeID
LEFT JOIN Writer ON Employee.employeeID = Writer.employeeID;

SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
RIGHT JOIN Actor ON Employee.employeeID = Actor.employeeID
FULL JOIN Writer ON Employee.employeeID = Writer.employeeID;
```

name	filmsWritten	numberOfLeadRoles
Henry Cavill	NULL	7
Robert Downey Jr.	NULL	37
Tom Holland	NULL	9
Seth MacFarlane	7	25
Benedict Cumberbatch	NULL	44
NULL	13	NULL
NULL	10	NULL
NULL	6	NULL
NULL	2	NULL
Seth MacFarlane	7	25

```
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
LEFT JOIN Actor ON Employee.employeeID = Actor.employeeID
FULL JOIN Writer ON Employee.employeeID = Writer.employeeID;
```

name	filmsWritten	numberOfLeadRoles
Scott Derrickson	NULL	NULL
Paul Feig	NULL	NULL
Robert Singer	NULL	NULL
Gary Halvorson	NULL	NULL
Joss Whedon	13	NULL
Henry Cavill	NULL	7
Robert Downe...	NULL	37
Tom Holland	NULL	9

UNION

- Combine the result of SELECT statements
- Need the same amount of columns and the same data types

```sql
--Union
SELECT 'Writer' as job, startYear FROM Writer
UNION
SELECT 'Actor', startYear FROM Actor
UNION
SELECT 'Director', startYear FROM Director;
```

	job	startYear
1	Actor	1970
2	Actor	1987
3	Actor	2001
4	Actor	2002
5	Actor	2010
6	Director	1976
7	Director	1989
8	Director	1995
9	Director	1997
10	Writer	1987
11	Writer	1989
12	Writer	1995
13	Writer	1997
14	Writer	1998

COMPARISON

- Comparison operations for attribute values,
  - Equal =
  - Not equal <> or !=
  - Less than <
  - More than >

```
--Comparison
SELECT * FROM Series
WHERE seasons < 12 AND type != 'Comedy';
```

entertainmentID	on_going	seasons	episodes	year_ended	type	entertainmentType
8	true	2	17	NULL	Action, Adventure	2
9	false	4	15	2017	Crime, Mystery	2
12	false	11	177	2022	Drama, Horror	2
13	false	10	235	2004	Comedy, Romance	2

WHERE IN

- A list of values that are acceptable
  - Can be achieved with multiple [equal OR] statements

```
--WHERE IN
SELECT * FROM Series
WHERE seasons IN (2,14,9,12);
--Where in using SELECT
SELECT name FROM Employee, Writer
WHERE Writer.employeeID = Employee.employeeID
AND (Writer.employeeID IN (SELECT employeeID FROM Actor)
OR Writer.employeeID IN (SELECT employeeID FROM Director));
```

entertainmentID	on_going	seasons	episodes	year_ended	type	entertainmentType
8	true	2	17	NULL	Action, Adventure	2
10	false	9	188	2013	Comedy	2
11	true	14	200	NULL	Adventure, Sci-Fi	2

name
Joss Whedon
Seth MacFarlane

GROUP BY

- Group results based on the result columns
- HAVE TO use all result columns either in grouping or other aggregation functions

```sql
--GROUP BY
SELECT Employee.name, Entertainment.name FROM Employee
INNER JOIN Contract ON Employee.employeeID = Contract.employeeID
INNER JOIN Entertainment ON
    Contract.entertainmentID= Entertainment.entertainmentID
ORDER BY Employee.name;
--vs
SELECT Employee.name, COUNT(Entertainment.name)  FROM Employee
INNER JOIN Contract ON Employee.employeeID = Contract.employeeID
INNER JOIN Entertainment ON
    Contract.entertainmentID= Entertainment.entertainmentID
GROUP BY Employee.name
ORDER BY Employee.name;
```

name	name
Benedict Cumberbatch	Avengers: Endgame
Benedict Cumberbatch	Sherlock
Benedict Cumberbatch	Doctor Strange
David Crane	Friends
Eiichiro Oda	One Piece
Eric Kripke	Supernatural
Gary Halvorson	Friends
Henry Cavill	The Witcher
Joss Whedon	The Avengers
Paul Feig	The Office
Robert Downey Jr.	Avengers: Endgame
Robert Downey Jr.	Iron Man
Robert Downey Jr.	Captain America, Civil War
Robert Singer	Supernatural
Seth MacFarlane	Family Guy
Seth MacFarlane	Family Guy
Tom Holland	Avengers: Endgame
Tom Holland	Captain America, Civil War

name	(No column name)
Benedict Cumberbatch	3
David Crane	1
Eiichiro Oda	1
Eric Kripke	1
Gary Halvorson	1
Henry Cavill	1
Joss Whedon	1
Paul Feig	1
Robert Downey Jr.	3
Robert Singer	1
Seth MacFarlane	2
Tom Holland	2

ORDER BY

- Order results by a column
  - Can be any column from any table used in the query
  - Even if you do not use the column in the query
- Can select ascending or descending order using ASC/DESC

name	filmsWritten	numberOfLeadRoles
Tom Holland	NULL	9
Benedict Cumberbatch	NULL	44
Henry Cavill	NULL	7
Seth MacFarlane	7	25
Robert Downey Jr.	NULL	37
Eiichiro Oda	10	NULL
Eric Kripke	6	NULL
David Crane	2	NULL
Scott Derrickson	NULL	NULL
Paul Feig	NULL	NULL
Robert Singer	NULL	NULL
Gary Halvorson	NULL	NULL
Joss Whedon	13	NULL

```
--ORDER BY  --ASC / DESC
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
LEFT JOIN Actor ON Employee.employeeID = Actor.employeeID
LEFT JOIN Writer ON Employee.employeeID = Writer.employeeID
ORDER BY Actor.startYear DESC;
```

AGGREGATION FUNCTIONS

- COUNT() – count the number of values (skips NULL)
- MAX() – Retrieve the maximum value
- MIN() – Retrieve the minimum value
- SUM() – Calculate the sum of all values
- AVG() – Calculate the average of all values

```sql
--AVG MIN MAX SUM COUNT
SELECT AVG(episodes) as AVG,
    SUM(episodes) as SUM,
    MIN(episodes) as MIN,
    MAX(episodes) as MAX,
    COUNT(episodes) as COUNT
FROM Series;
```

- Cannot be used with WHERE, have to use HAVING

	AVG	SUM	MIN	MAX	COUNT
1	165	1159	15	327	7

NULL, EXISTS

- Sometimes it is necessary to find out if something has a value
- NULL values cannot be compared with = or != operators
  - IS NULL, NOT NULL
- EXISTS / NOT EXISTS test if the record is found in a subquery
  - Returns true or false

```sql
--EXISTS
SELECT name FROM Employee E1
WHERE NOT EXISTS
    (SELECT name FROM Employee E2
    INNER JOIN Contract ON E2.employeeID = Contract.employeeID
    WHERE E1.employeeID = E2.employeeID
    AND employeeTypeID = 3);

-- Not null
SELECT name, filmsWritten, numberOfLeadRoles FROM Employee
    LEFT JOIN Actor ON Employee.employeeID = Actor.employeeID
    LEFT JOIN Writer ON Employee.employeeID = Writer.employeeID
    WHERE filmsWritten IS NOT NULL
ORDER BY Actor.startYear DESC;
```

	name
1	Scott Derrickson
2	Paul Feig
3	Robert Singer
4	Gary Halvorson
5	Joss Whedon
6	Henry Cavill
7	Robert Downey Jr.
8	Tom Holland
9	Benedict Cumberbatch

name	filmsWritten	numberOfLeadRoles
Seth MacFarlane	7	25
Joss Whedon	13	NULL
Eiichiro Oda	10	NULL
Eric Kripke	6	NULL
David Crane	2	NULL

CASE, ANY, ALL

- CASE is similar to ‘if’ or ‘switch-case’ statements in programming
  - Give different conditions and results for values
- ANY is a broad comparison that returns TRUE if any matching data is found
  - Useful for broad filtering
- ALL requires each compared data to match
  - Not useful with equal operations

```sql
--Case
SELECT name,
CASE
    WHEN year_released < 2000 THEN
        'from the 90s'
    WHEN year_released < 2010 THEN
        '21st century'
    WHEN year_released < 2020 THEN
        'not quite new'
    END 'Colum name'
FROM Entertainment;
```

name	Colum name
Iron Man	21st century
The Avengers	not quite new
Captain America, Civil War	not quite new
Doctor Strange	21st century
Thor: Rahnarök	21st century
Avengers: Endgame	21st century
Spider-Man: No Way Home	21st century
The Witcher	not quite new
Sherlock	not quite new
The Office	21st century
Doctor Who	21st century
The Walking Dead	not quite new
Friends	from the 90s
Supernatural	21st century
Family Guy	from the 90s

```sql
--Any, a contarct exists between employee and entertainment where the entertainment type not a series
SELECT name FROM Employee
WHERE employeeID = ANY
    (SELECT employeeID
    FROM Contract
    INNER JOIN Entertainment ON Entertainment.entertainmentID = Contract.entertainmentID
    WHERE entertainmentType != 2
    );
```

```sql
--SELECT all employees that do not have a contract with entertainment type of not a series
SELECT name FROM Employee
WHERE employeeID != ALL
    (SELECT employeeID
    FROM Contract
    INNER JOIN Entertainment ON Entertainment.entertainmentID = Contract.entertainmentID
    WHERE entertainmentType != 2
    );
```

BASIC STEPS OF BUILDING A QUERY

1. What do you want to know?
    1. Formulate a question or a sentence you are trying to answer.
2. Based on the question, what data you need?
    1. What columns you need to answer your question? (SELECT)
    2. What tables those attributes reside in? (FROM)
    3. If you need multiple relations,
        1. Need new columns from multiple tables? (JOIN, subquery)
        2. Need same columns from multiple tables? (UNION)
    4. Need to filter results?
        1. Specific case? (WHERE)
        2. Multiple conditions? (AND, OR, NOT)
        3. Results reside in a set? (WHERE IN)

INTRO EXAMPLES REVISITED, PART 1

1. What to answer: *Return all countries together with the number of related calls and their average duration in seconds. In the result display only countries where average call duration is greater than the average call duration of all calls.*
2. Resulting columns:
    1. Country name, number of calls,
       AVG call time in seconds
3. From where?
    1. Country name from Country, call duration from call
    2. Need to join
       country → city → customer → call
4. Filter
    1. Only those countries that have a higher average call duration compared to average duration of all countries

```sql
SELECT
    country.country_name_eng,
    SUM(CASE WHEN call.id IS NOT NULL THEN 1 ELSE 0 END) AS calls,
    AVG(ISNULL(DATEDIFF(SECOND, call.start_time, call.end_time),0)) AS avg_difference
FROM country
LEFT JOIN city ON city.country_id = country.id
LEFT JOIN customer ON city.id = customer.city_id
LEFT JOIN call ON call.customer_id = customer.id
GROUP BY
    country.id,
    country.country_name_eng
HAVING AVG(ISNULL(DATEDIFF(SECOND, call.start_time, call.end_time),0)) > (SELECT AVG(DATEDIFF(SECO
ORDER BY calls DESC, country.id ASC;
```

Source:https://www.sqlshack.com/learn-sql-how-to-write-a-complex-select-query/

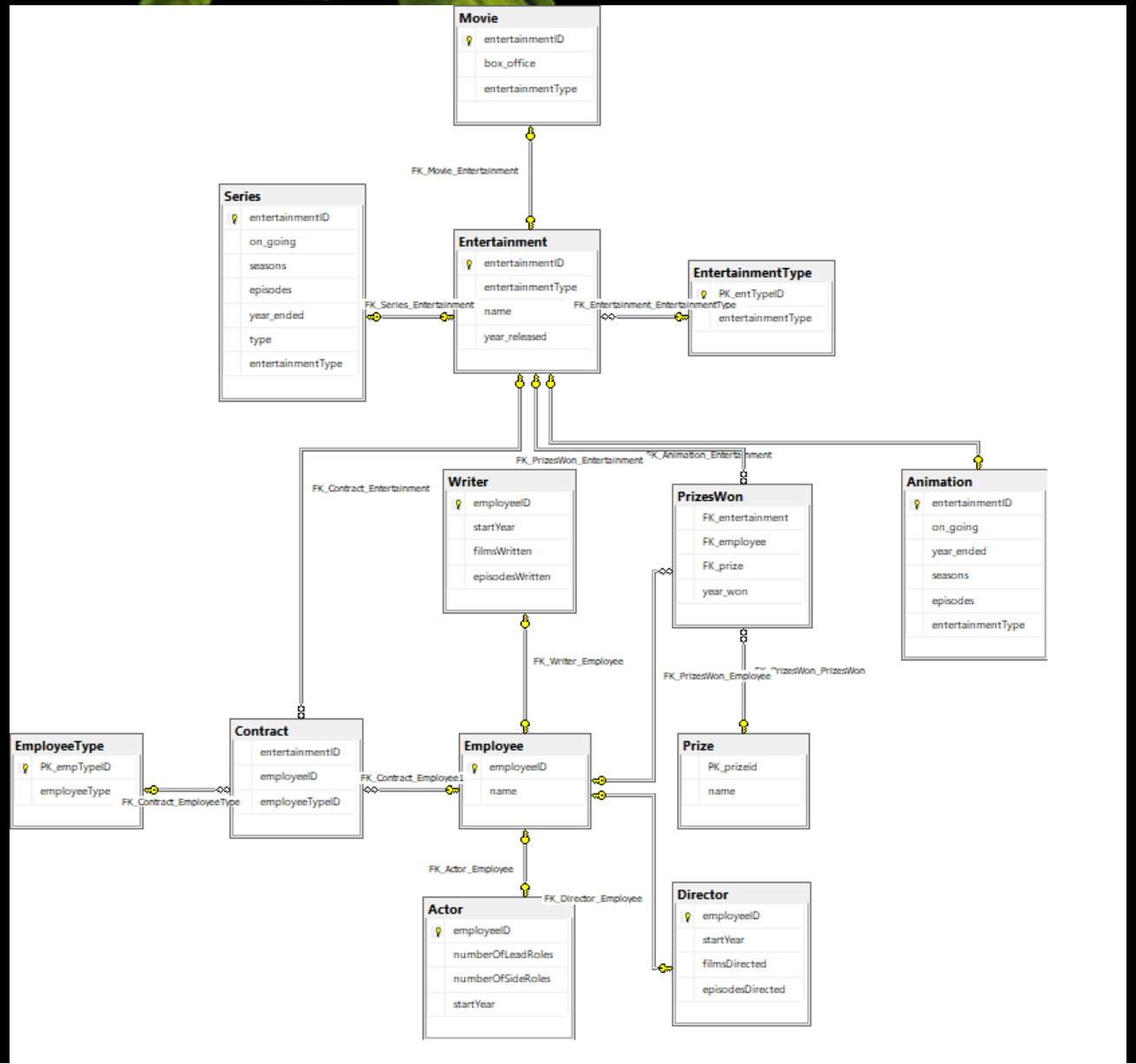
INTRO EXAMPLES REVISITED, PART 2

1. What to answer: *Print the names of a pair of students who have received a grade from a course they enrolled to and have no more lectures left.*
2. Resulting columns:
    1. Student names
3. From where?
    1. Name from Person
    2. Need to join person→ student
    3. Find enrolment information student→enroll→lecture
4. Filter
    1. Only those students who have a grade AND have NO lectures left (NOT EXISTS clause)
       There should be NO lectures where the S1 and S2 should attend.

```
SELECT P1.Name, P2.Name
    FROM Person P1, Person P2, Student S1, Student S2, Enroll E1, Enroll E2
    WHERE P1.Name = S1.Name AND P1.DateOfBirth = S1.DateOfBirth
    AND S1.StudNo = E1.StudNo AND E1.Grade IS NOT NULL
    AND P2.Name = S2.Name AND P2.DateOfBirth = S2.DateOfBirth
    AND S2.StudNo = E2.StudNo AND E2.Grade IS NOT NULL
    AND S1.StudNo < S2.StudNo AND
    NOT EXISTS (
            SELECT * FROM Lecture AS L
                WHERE L.CourseNo IN
                (SELECT B.CourseNo FROM Enroll AS B
                    WHERE S1.StudNo = B.StudNo
                    OR S2.StudNo = B.StudNo)
                AND
                NOT EXISTS (
                    ( SELECT * FROM Enroll AS B1
                        WHERE S1.StudNo = B1.StudNo
                            AND B1.CourseNo = L.CourseNo)
                        UNION
                        (SELECT * FROM Enroll AS B2
                            WHERE S2.StudNo = B2.StudNo
                            AND B2.CourseNo = L.CourseNo)
                        )
                );
```

QUESTION

1. *Who have participated in what movies, series or animation that have been released during 21st century and what are their respective roles?*
2. Resulting columns:
    1. Employee names and roles, entertainment names and types.
3. From where?
    1. Name from Employee, type from Contract, name and type from Entertainment
    2. Need to join Employee→ Contract, Contract→Entertainment
4. Filter
    1. Filter Entertainment based on year_released > 1999



QUERY

```sql
--Example solution 1
SELECT Employee.name,
CASE
    WHEN Contract.employeeTypeID = 1 THEN
        'Director'
    WHEN Contract.employeeTypeID = 2 THEN
        'Actor'
    WHEN Contract.employeeTypeID = 3 THEN
        'Writer'
    END Role,
Entertainment.name,
CASE
    WHEN Entertainment.entertainmentType = 1 THEN
        'Movie'
    WHEN Entertainment.entertainmentType = 2 THEN
        'Series'
    WHEN Entertainment.entertainmentType = 3 THEN
        'Animation'
    END Type
FROM Contract
INNER JOIN Employee ON
    Contract.employeeID = Employee.employeeID
INNER JOIN Entertainment ON
    Contract.entertainmentID = Entertainment.entertainmentID
WHERE Entertainment.year_released > 1999;
```

```sql
--Example solution 2
SELECT Employee.name, EmployeeType.employeeType AS Role,
Entertainment.name, EntertainmentType.entertainmentType AS Type
FROM Contract
INNER JOIN Employee ON
    Contract.employeeID = Employee.employeeID
INNER JOIN EmployeeType ON
    Contract.employeeTypeID = EmployeeType.PK_empTypeID
INNER JOIN Entertainment ON
    Contract.entertainmentID = Entertainment.entertainmentID
INNER JOIN EntertainmentType ON
    Entertainment.entertainmentType = EntertainmentType.PK_entTypeID
WHERE Entertainment.year_released > 1999;
```

	name	Role	name	Type
1	Robert Downey Jr.	Actor	Iron Man	Movie
2	Joss Whedon	Director	The Avengers	Movie
3	Robert Downey Jr.	Actor	Captain America, Civil War	Movie
4	Tom Holland	Actor	Captain America, Civil War	Movie
5	Benedict Cumberbatch	Actor	Doctor Strange	Movie
6	Robert Downey Jr.	Actor	Avengers: Endgame	Movie
7	Tom Holland	Actor	Avengers: Endgame	Movie
8	Benedict Cumberbatch	Actor	Avengers: Endgame	Movie
9	Henry Cavill	Actor	The Witcher	Series

	name	Role	name	Type
6	Robert Downey Jr.	Actor	Avengers: Endgame	Movie
7	Tom Holland	Actor	Avengers: Endgame	Movie
8	Benedict Cumberbatch	Actor	Avengers: Endgame	Movie
9	Henry Cavill	Actor	The Witcher	Series
10	Benedict Cumberbatch	Actor	Sherlock	Series
11	Paul Feig	Director	The Office	Series
12	Robert Singer	Director	Supernatural	Series
13	Eric Kripke	Writer	Supernatural	Series


LUT
University