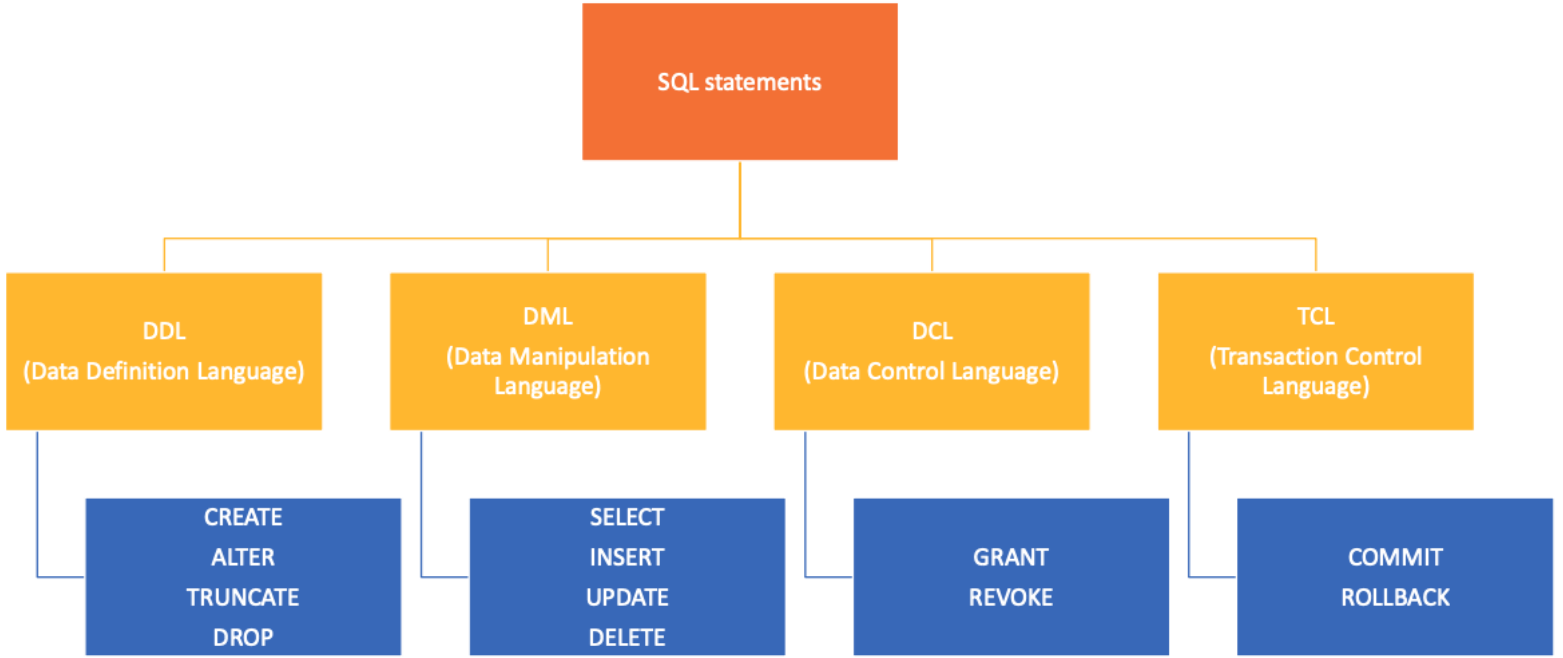
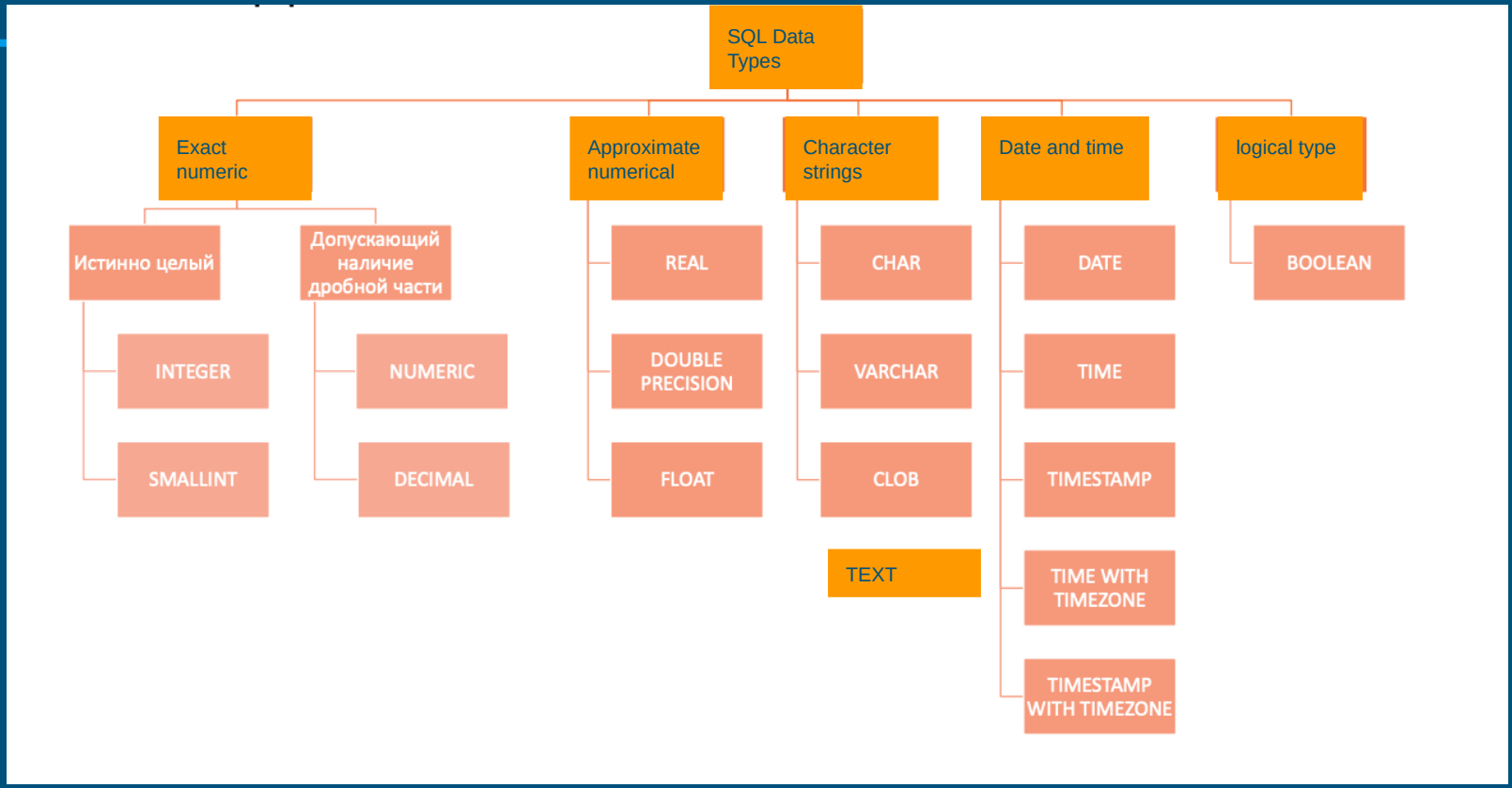
Databases

Seminar 2

SQL Statements



SQL Data Types



Data Definition Language

1. CREATE – creation of objects in the database

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name(
    col_name_1   datatype_1,
    col_name_2   datatype_2,
    ...
    col_name_N   datatype_N
);
```

Data Definition Language

2. ALTER - modification of objects

ALTER TABLE table_name ADD column_name datatype;

ALTER TABLE table_name DROP column_name;

ALTER TABLE table_name RENAME column_name TO new_column_name;

ALTER TABLE table_name ALTER column_name TYPE datatype;

...

Data Definition Language

3. DROP – deleting objects

DROP TABLE [IF EXISTS] table_name;

4. TRUNCATE – deleting the contents of the database object (data is deleted as a whole piece, cannot be deleted by condition)

TRUNCATE TABLE table_name;

Data Manipulation Language

1. SELECT – selects data that meets the specified conditions
2. INSERT – adds new data
3. UPDATE – changes (updates) existing data

```
UPDATE table_name
SET update_assignment_comma_list
WHERE conditional_experssion;
```

4. DELETE – deletes existing data (data is deleted line by line – you can set a condition, "roll back" deletion)

```
DELETE

    FROM table_name

[WHERE conditional_expression];
```

Table Join Operations

Connection operations are divided into 3 groups:

CROSS JOIN - Cartesian product of 2 tables

INNER JOIN - joining 2 tables by condition. The resulting selection will include only those records that satisfy the connection condition

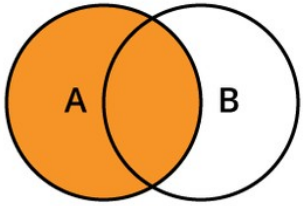
OUTER JOIN - joining 2 tables by condition. The resulting selection may include records that do not satisfy the connection condition:

LEFT (OUTER) JOIN - all rows of the "left" table are included in the final selection

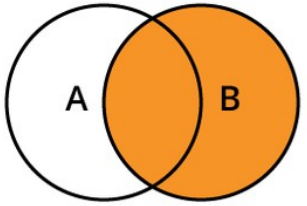
RIGHT (OUTER) JOIN - all rows of the "right" table are included in the final selection

FULL (OUTER) JOIN - all rows of both tables are included in the final selection

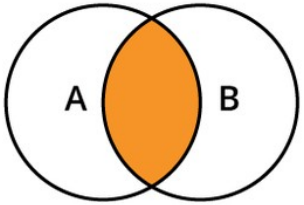
SQL JOIN QUERIES



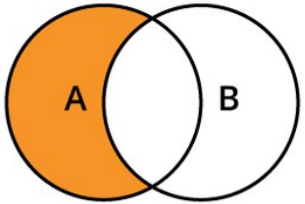

```
SELECT <select_list>
FROM TableA
LEFT JOIN TableB
ON A.Key = B.Key
```



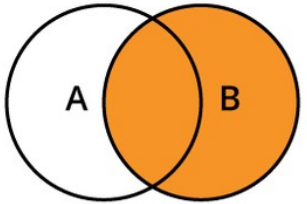

```
SELECT <select_list>
FROM TableA
RIGHT JOIN TableB
ON A.Key = B.Key
```



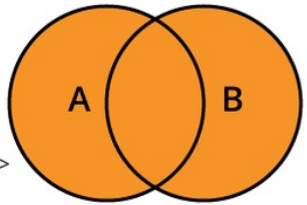

```
SELECT <select_list>
FROM TableA
INNER JOIN TableB
ON A.Key = B.Key
```



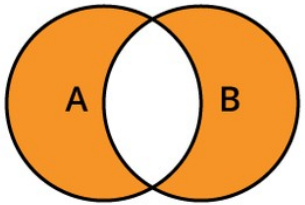

```
SELECT <select_list>
FROM TableA
LEFT JOIN TableB
ON A.Key = B.Key
WHERE B.Key IS NULL
```




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

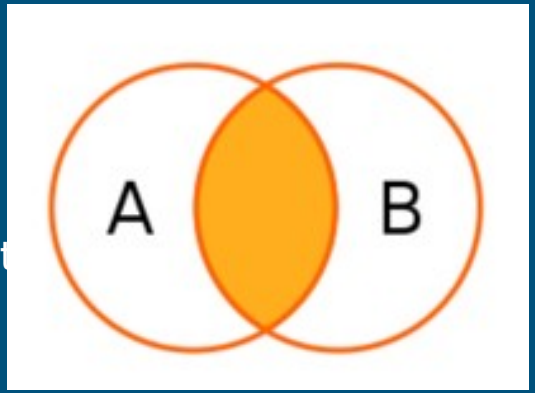



```
SELECT <select_list>
FROM TableA
FULL OUTER JOIN TableB
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA
FULL OUTER JOIN TableB
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS Null
```

Inner Join



- joins 2 tables
- symmetrical (the order of the tables is not important
- Algorithm:

Each row of the first table is compared with the rows in the second table.

For each row the join condition is checked.

If the Join condition is met, a row is added to the result.

Inner Join Example

People

Id	LastName	FirstName	...
1	Goldstein	Larry	...
2	Burton	Tom	...
3	Hamilton	Lisa	...
4	Jackson	Kim	...

Contacts

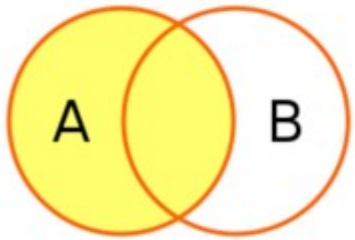
Id	PersonId	ContactType	ContactValue
1	1	Landline	555-0100
2	1	Email	gl@acme.com
3	1	Email	l.goldstein@gmail.com
4	1	Skype	GoldenLarry123
5	4	Landline	555-0101
6	4	Mobile	+10123444444

```
SELECT *
FROM People p
JOIN Contacts c ON c.PersonId = p.Id
```

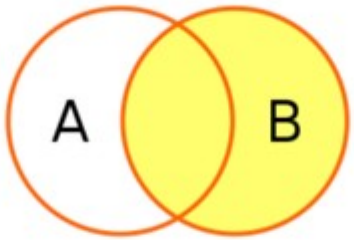
p.Id	p.LastName	p.FirstName	...	c.Id	c.PersonId	c.ContactType	c.ContactValue
1	Goldstein	Larry	...	1	1	Landline	555-0100
1	Goldstein	Larry	...	2	1	Email	gl@acme.com
1	Goldstein	Larry	...	3	1	Email	l.goldstein@gmail.com
1	Goldstein	Larry	...	4	1	Email	GoldenLarry123
4	Jackson	Kim	...	5	4	Landline	555-0101
4	Jackson	Kim	...	6	4	Mobile	+10123444444

Result:

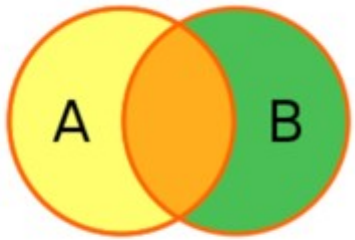
Outer Join



LEFT OUTER JOIN



RIGHT OUTER JOIN



FULL OUTER JOIN

- Joins 2 tables
- The result includes all rows from one of the tables

Left Outer Join Example

People

Id	LastName	FirstName	…
1	Goldstein	Larry	…
2	Burton	Tom	…
3	Hamilton	Lisa	…
4	Jackson	Kim	…

Contacts

Id	PersonId	ContactType	ContactValue
1	1	Landline	555-0100
2	1	Email	gl@acme.com
3	1	Email	l.goldstein@gmail.com
4	1	Skype	GoldenLarry123
5	4	Landline	555-0101
6	4	Mobile	+10123444444

```
SELECT *
FROM People p
LEFT OUTER JOIN Contacts c ON c.PersonId = p.Id
```

p.Id	p.LastName	p.FirstName	…	c.Id	c.PersonId	c.ContactType	c.ContactValue
1	Goldstein	Larry	…	1	1	Landline	555-0100
1	Goldstein	Larry	…	2	1	Email	gl@acme.com
1	Goldstein	Larry	…	3	1	Email	l.goldstein@gmail.com
1	Goldstein	Larry	…	4	1	Email	GoldenLarry123
2	Burton	Tom	…	NULL	NULL	NULL	NULL
3	Hamilton	Lisa	…	NULL	NULL	NULL	NULL
4	Jackson	Kim	…	5	4	Landline	555-0101
4	Jackson	Kim	…	6	4	Mobile	+10123444444

Result:

Right Outer Join Example

People

Id	LastName	FirstName	…
1	Goldstein	Larry	…
2	Burton	Tom	…
3	Hamilton	Lisa	…
4	Jackson	Kim	…

Contacts

Id	PersonId	ContactType	ContactValue
1	1	Landline	555-0100
2	1	Email	gl@acme.com
3	1	Email	l.goldstein@gmail.com
4	1	Skype	GoldenLarry123
5	4	Landline	555-0101
6	4	Mobile	+10123444444
7	NULL	Skype	JoeJack1977

```
SELECT *
FROM People p
RIGHT OUTER JOIN Contacts c ON c.PersonId = p.Id
```

p.Id	p.LastName	p.FirstName	…	c.Id	c.PersonId	c.ContactType	c.ContactValue
1	Goldstein	Larry	…	1	1	Landline	555-0100
1	Goldstein	Larry	…	2	1	Email	gl@acme.com
1	Goldstein	Larry	…	3	1	Email	l.goldstein@gmail.com
1	Goldstein	Larry	…	4	1	Email	GoldenLarry123
4	Jackson	Kim	…	5	4	Landline	555-0101
4	Jackson	Kim	…	6	4	Mobile	+10123444444
NULL	NULL	NULL	NULL	7	NULL	Skype	JoeJack1977

Result:

Full Outer Join Example

People

Id	LastName	FirstName	…
1	Goldstein	Larry	…
2	Burton	Tom	…
3	Hamilton	Lisa	…
4	Jackson	Kim	…

Contacts

Id	PersonId	ContactType	ContactValue
1	1	Landline	555-0100
2	1	Email	gl@acme.com
3	1	Email	l.goldstein@gmail.com
4	1	Skype	GoldenLarry123
5	4	Landline	555-0101
6	4	Mobile	+10123444444
7	NULL	Skype	JoeJack1977

```
SELECT *
FROM People p
FULL OUTER JOIN Contacts c ON c.PersonId = p.Id
```

p.Id	p.LastName	p.FirstName	…	c.Id	c.PersonId	c.ContactType	c.ContactValue
1	Goldstein	Larry	…	1	1	Landline	555-0100
1	Goldstein	Larry	…	2	1	Email	gl@acme.com
1	Goldstein	Larry	…	3	1	Email	l.goldstein@gmail.com
1	Goldstein	Larry	…	4	1	Email	GoldenLarry123
2	Burton	Tom	…	NULL	NULL	NULL	NULL
3	Hamilton	Lisa	…	NULL	NULL	NULL	NULL
4	Jackson	Kim	…	5	4	Landline	555-0101
4	Jackson	Kim	…	6	4	Mobile	+10123444444
NULL	NULL	NULL	NULL	7	NULL	Skype	JoeJack1977

Result:

Table Join Operations

If join condition includes columns with the same names then it can be abbreviated with USING:

```
ON left_table.a = right_table.a AND left_table.b = right_table.b
USING (a, b)
```

A shorter way of writing the same expression uses NATURAL:

```
SELECT select_list
FROM T1 NATURAL JOIN T2
```

The principle of expressions with NATURAL:

- Similar to USING with the indication of all columns of the same name
- If there are no columns with the same name, then ON TRUE is the same

Keys

A **potential key** is a subset of the attributes of a relationship that meets the requirements of uniqueness and minimality:

- Uniqueness: there are no two tuples of a given relationship in which the values of this subset of attributes match;
- Minimality: a smaller tuple of attributes satisfying the uniqueness condition is missing from the potential key;

Types:

- simple (consists of exactly one attribute)
- composite (consists of two or more attributes)

Example of a composite key:

```
CREATE TABLE flight_schedule ( departure timestampt, gate, pilot
 UNIQUE(departure, gate),
UNIQUE(departure, pilot) );
```

Primary and Alternative Keys

Primary keys (PK) are any of the potential keys selected as the primary key. Generally the primary key is chosen so that it

- takes up less memory
- will not lose its uniqueness over time. (a potential key always exists, even if it includes all the attributes of the relationship).

Alternative keys are potential keys that were not selected as primary keys.

Types of keys:

- natural (based on an already existing field)
- intelligent (based on a natural key with an added additional field)

Surrogate Keys

A **surrogate key** is an additional service field that is added to the already existing information fields of the table, the only purpose of which is to serve as a primary key. (the value is generated artificially).

Let R1 and R2 be two variable relations, not necessarily different. The foreign key FK (Foreign key) in R2 is a subset of the attributes of the R2 variable such that the following requirements are met:

- In the relation variable R1 there is a potential PK key such that PK and FK match exactly up to the renaming of attributes (FK from R2 is PK from R1)
- At any given time, each value of FK in the current value of R2 is identical to the value of PK in some tuple in the current value of R1. In other words, at any given time, the set of all values of FK in R2 is a subset of the values of PK in R1.
- The *parent (main/target)* relation is the R1 relation containing the potential key.
- The *child (subordinate)* relationship is the R2 relationship, which contains a reference to the entity in which the attributes we need are located. (containing a foreign key)

Creating Keys: Primary Key

```
CREATE TABLE PERSON (
            ID INTEGER PRIMARY KEY,
            LAST_NAME VARCHAR(255) NOT NULL,
            FIRST_NAME VARCHAR(255) NOT NULL,
            AGE INTEGER );
ALTER TABLE PERSON ADD PRIMARY KEY (ID);
----------------------------------------
CREATE TABLE PERSON (
            ID INTEGER,
            LAST_NAME VARCHAR(255),
            FIRST_NAME VARCHAR(255) NOT NULL,
            AGE INTEGER,
            CONSTRAINT PK_Person PRIMARY KEY (ID, LAST_NAME) );

ALTER TABLE PERSON
ADD CONSTRAINT PK_Person PRIMARY KEY (ID, LAST_NAME);

ALTER TABLE PERSON DROP CONSTRAINT PK_Person;
```

Creating Keys: Foreign Key

```
CREATE TABLE ORDER (
        ORDER_ID INTEGER,
        ORDER_NUMBER INTEGER NOT NULL,
        ERSON_ID INTEGER,
        PRIMARY KEY (ORDER_ID),
        CONSTRAINT FK_PersonOrder FOREIGN KEY (PERSON_ID) REFERENCES
        PERSON(PERSON_ID) );

ALTER TABLE ORDER ADD CONSTRAINT FK_PersonOrder FOREIGN KEY (PERSON_ID)
REFERENCES PERSON(PERSON_ID); ALTER TABLE ORDER DROP CONSTRAINT
FK_PersonOrder;
----------------------------------------
CREATE TABLE ORDER (
        ORDER_ID INTEGER PRIMARY KEY,
        ORDER_NUMBER INTEGER NOT NULL,
        PERSON_ID INTEGER REFERENCES PERSON(PERSON_ID) );

ALTER TABLE ORDER ADD FOREIGN KEY (PERSON_ID) REFERENCES PERSON(PERSON_ID);
```

Practice

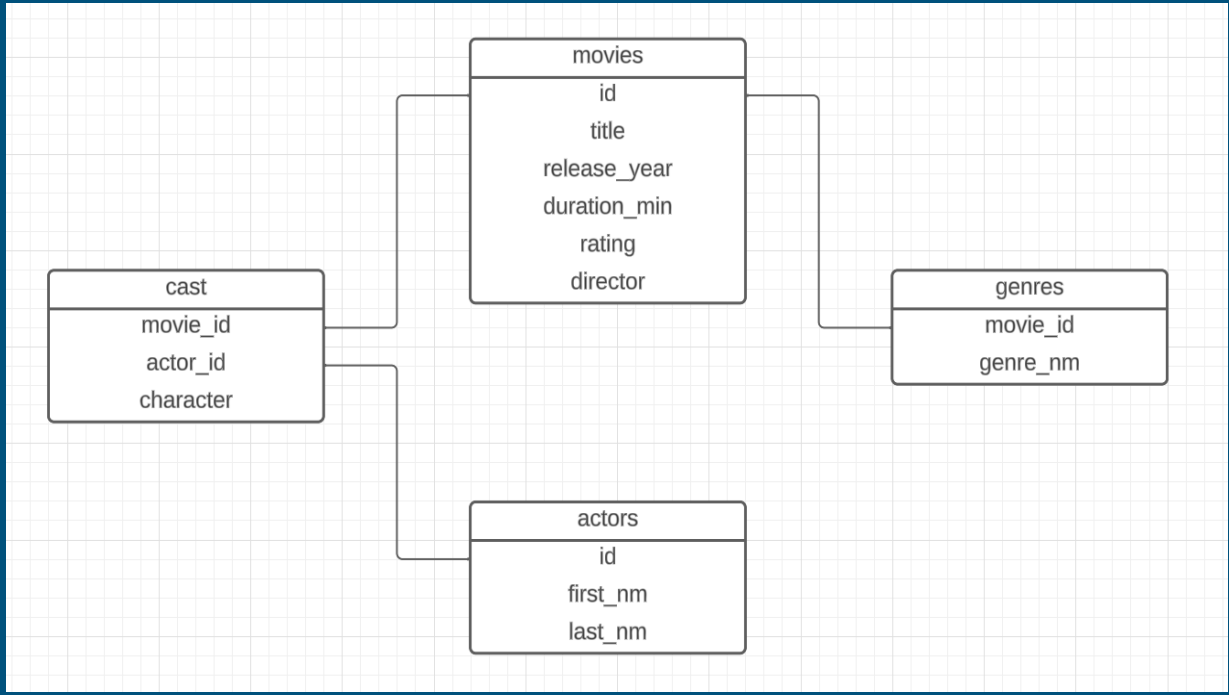
Create the following tables. What primary and foreign keys are needed here? Create them.
Create an id column of the serial type. What is the name of this type of key?

Tables:

Movies:
id
title
release_year
duration_min
rating
director

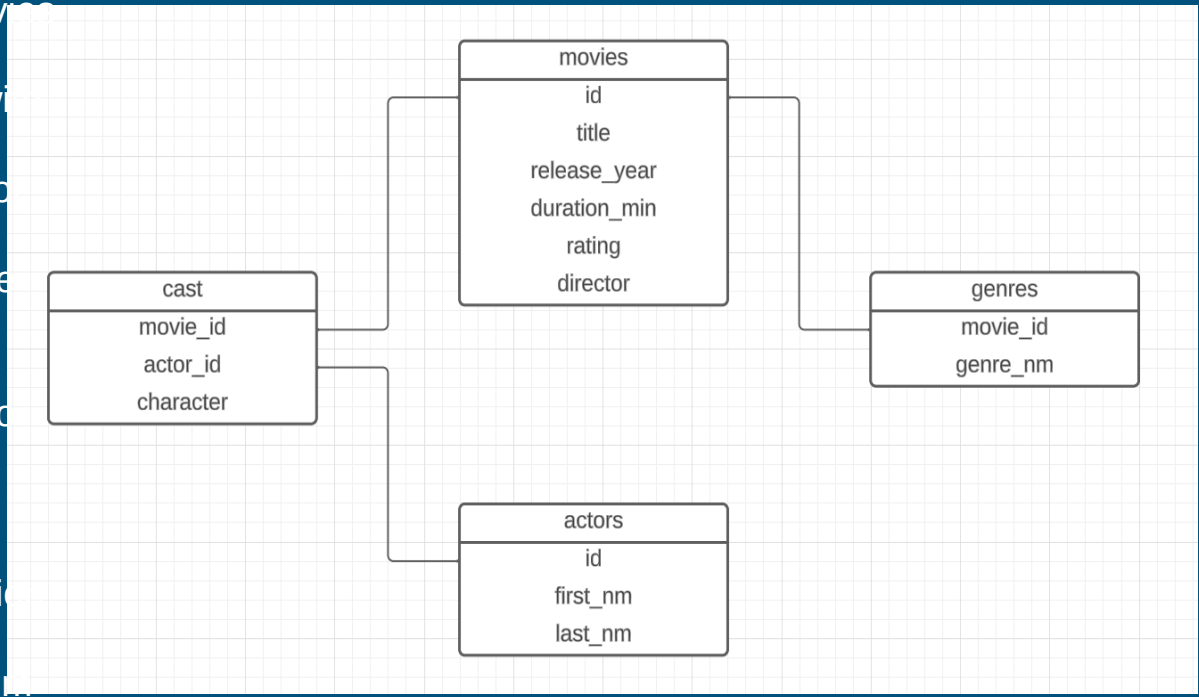
Cast:
movie_id
actor_id
character_nm
genres
movie_id
genre_nm

Actors:
id
first_nm
last_nm



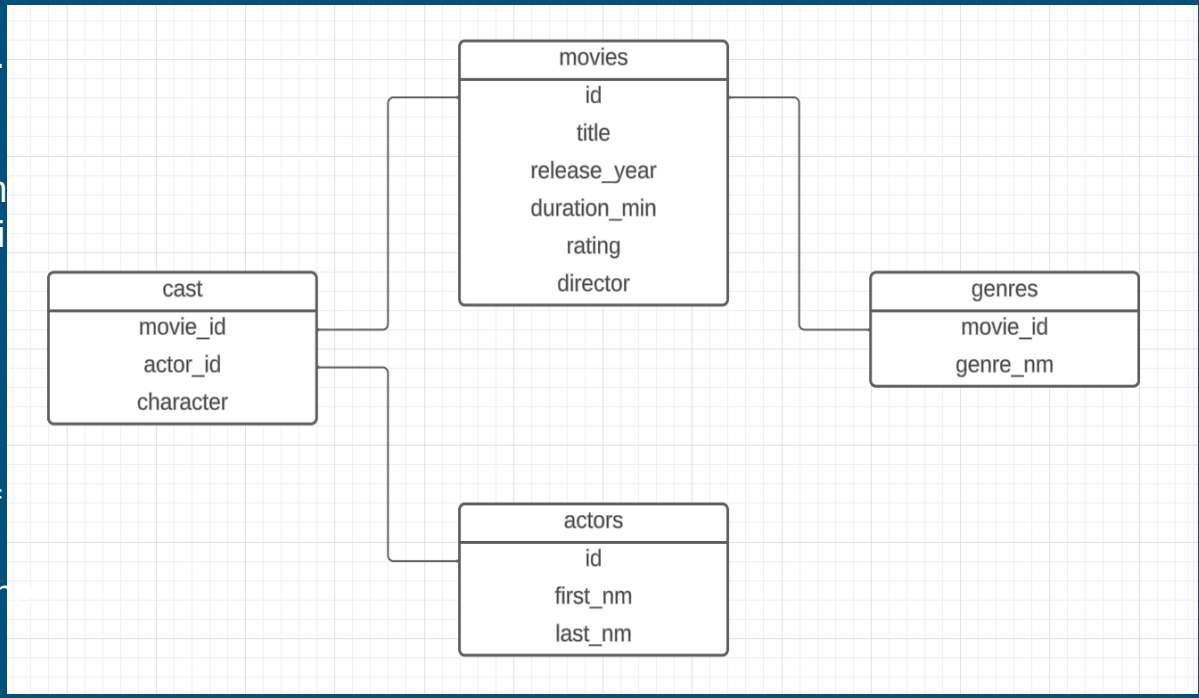
Practice

1. Fill in the movies table with 3 text lines.
2. Add a new comment field to the movies table.
3. Write a request to update the field with a comment.
You must specify your own comment for each line.
Think about how to do this with a single UPDATE operation, rather than five different requests.
4. Delete one of the rows of the table to choose from.
5. Clear the table using the DDL group operator.
6. Fill in the table again and pay attention to the IDs. Clean it completely again.
7. Delete a column with a comment from the table.



Practice

8. Start insertion operations from a separate file
9. Find all the films of the Crime genre. Print the title of the film, the year of release and the rating
10. Find the IDs of the actors for whom there is no information about the films i which they starred
11. What is the name of the actor who played 'Harry Potter'?
12. Output all films of the 90s genres Drama and Romance
13. For each genre, find the number of films and the average rating
Sort by descending average rating, if th number of films is equal in descending order



Practice

14. For each actor, print the number of films in which he played (maybe 0).
Sort the number of movies in descending order
15. Find all the movies that Jake Gyllenhaal has played in. Print the name of the movie, the year of release and the duration. Sort by increasing the length of the movie
16. Bring out all the movies with the actor who played 'Captain Jack Sparrow'
17. For each movie, output its genres separated by commas as a string
(for example, using STRING_AGG)
If the genre is not specified for the movie, output -.
18. Find all the actors who played with Leonardo DiCaprio.
Optional: display the films in which they played together.