Databases

Lecture 2
Structured SQL Query Language

What we know

- Data is stored according to predefined rules (data schema)
- Working with data according to predefined rules

Relational data model:

- A logical data model that does not depend on physical structures
- Based on mathematics and logic
- Relational Algebra

STRUCTURED QUERY LANGUAGE (SQL)

- Domain-specific language (Domain-specific language)
- Used to work with relational databases
- Managing a large amount of information with a single request
- No need to specify how we get the record

History

Donald Chamberlin and Ray Boyce, IBM:

- Square: (Specifying Queries As Relational Expressions)
- SEQUEL (Structured English QUEry Language), 1973-1974
- Pat Selinger – cost-based optimizer
- Raymond Laurie – query compiler

Later SEQUEL -> SQL

University of California Berkeley:

- QUEL – could not stand the competition with SQL

SQL language standard

Background:

- Different software from different manufacturers
- Native query language implementation

Wanted to get:

- Software portability

Received:

- Partial portability

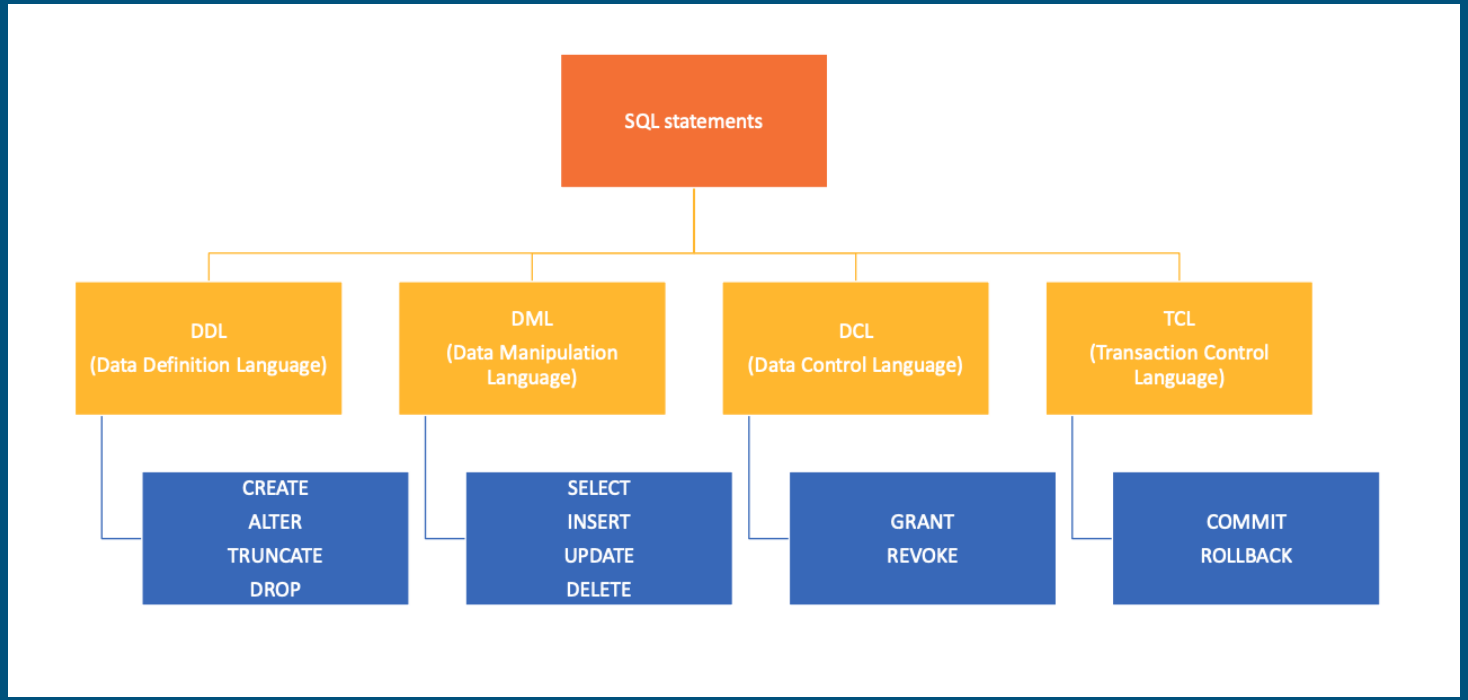
SQL language standard

1986	The first attempt at formalization
1989	SQL-86 + integrity constraint
1992	A lot of changes
1999	Regular expression matching, recursive queries, triggers, support for procedural and control operations, non-scalar types and object-oriented features.
2003	Support for SQL implementation in Java and vice versa
2006	XML-related functions, window functions, standardized sequences, and columns with automatically generated values
2008	The way SQL works with XML is defined: ways of importing and storing, publishing XML and regular data in XML format
2011	TRUNCATE, INSTEAD OF triggers. Improved window functions
2016	Adds string pattern matching, polymorphic table functions, JSON

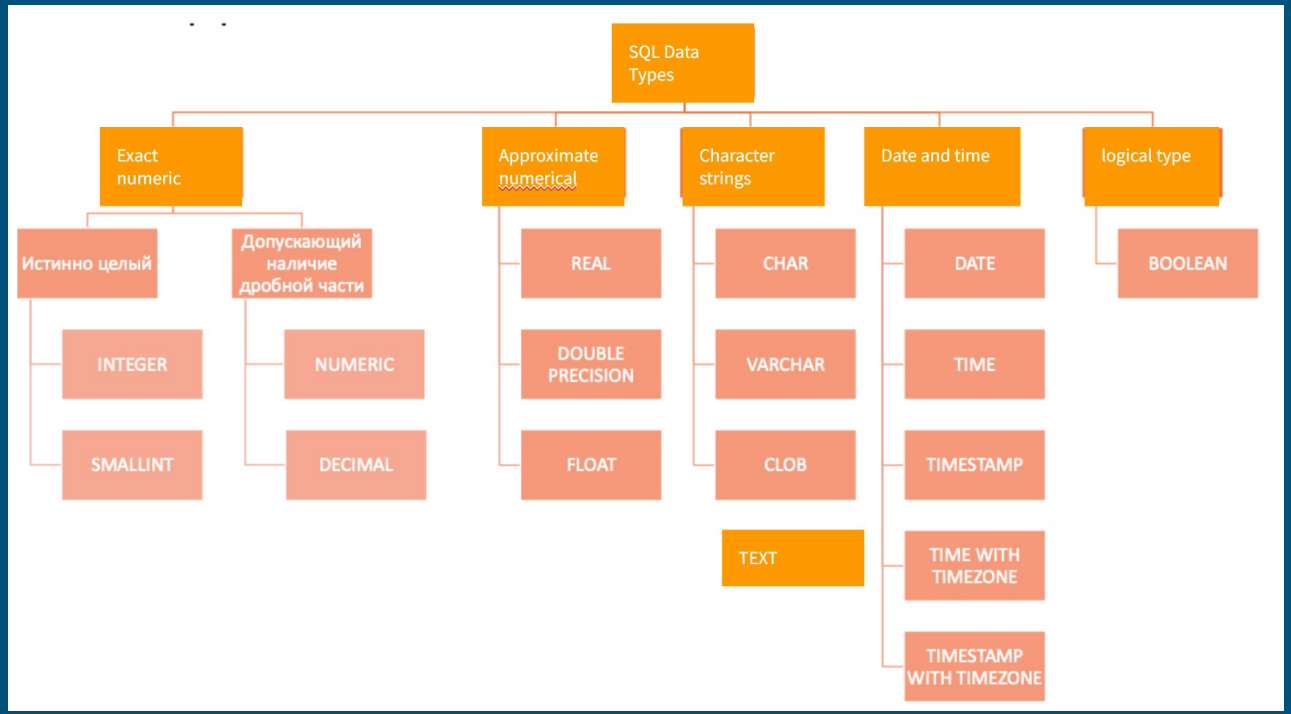
Standards issues

- Core-section of the standard (introduced in 1992)
- Manufacturers ensure compliance with Core only
- Implementation differences
- Syntax differences
- Differences in logic

SQL Statements



SQL Data Types



SQL program structure

- Program is a sequence of commands
- Command is a sequence of components. Command ends with ;
- Components are key words, identificators, operators, strings, special symbols.
- Components are separated by space symbols.
- Each command has its own syntax.

Comments

```
CREATE TABLE lecturers (
                id integer, --comment #1
                name text, --comment #2
    /* Multiline
comments
*/
birth_date date DEFAULT now() );
```

Data Definition Language

CREATE

- Database object creation operation

ALTER

- Database Object Modification Operator

DROP

- Database Object Deletion operator

TRUNCATE

- Operator for deleting the contents of a DB object

Data Definition Language

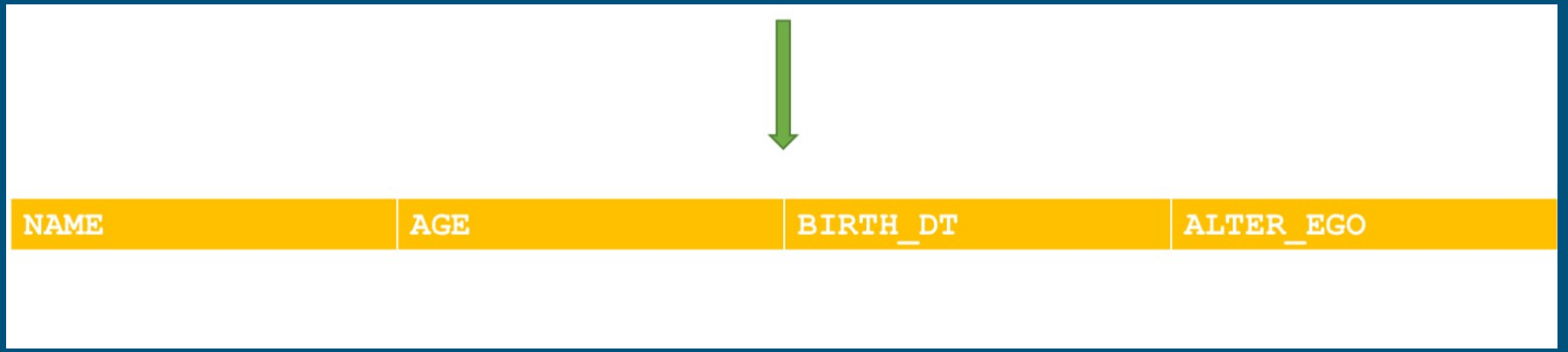
CREATE

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

Create

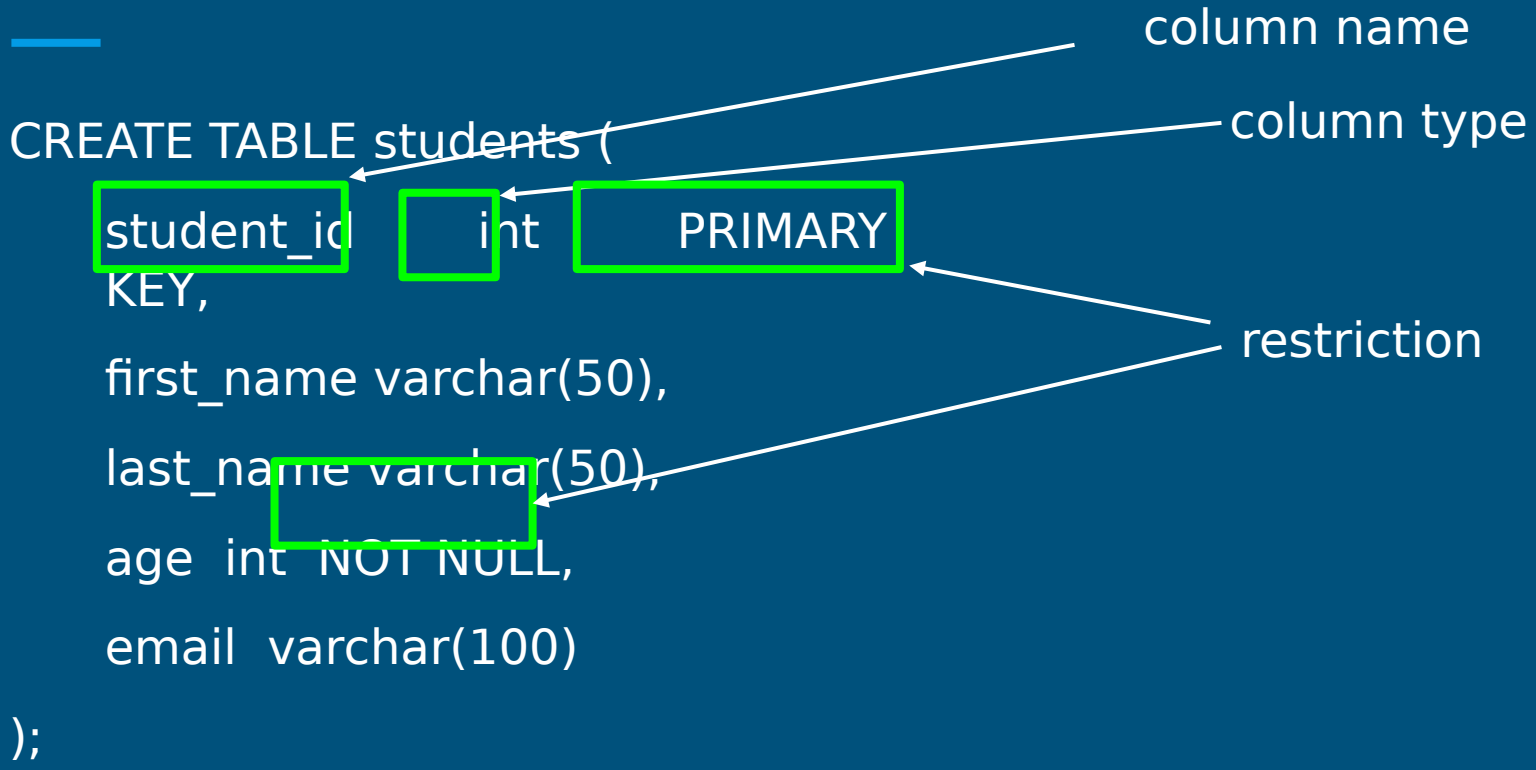
```
CREATE TABLE SUPERHERO(
  NAME          VARCHAR(100),
  AGE           INTEGER,
  BIRTH_DT      DATE,
  ALTER_EGO     VARCHAR(50)
);
```

Create



NAME	AGE	BIRTH_DT	ALTER_EGO

Create: restrictions



Restrictions

- Serve as a means of maintaining data integrity
- They guarantee that when entering data into the table, the relationships between the data will not be broken
- Ensure referential integrity between tables (relationships) - through foreign keys
- The constraint is specified after the data type or as a separate “statement” when defining the table

Restrictions: UNIQUE

UNIQUE - a group of one or more table columns can only contain unique values:

```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
);
```

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no)
);
```

```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    UNIQUE (a, c)
);
```

NULL value

A field with a NULL value is a field with no value.

If a field in a table is not required, then you can insert a new record or update without adding a value to the field; it will be stored as NULL.

A NULL value is different from a null value (0) or a field that contains spaces (" "). It is not possible to check for NULL values with comparison operators such as =, <, or <>.

You need to use IS NULL or IS NOT NULL

Restrictions: IS NULL / IS NOT NULL

IS NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NULL
```

IS NOT NULL Syntax

```
SELECT column_names
FROM table_name
WHERE column_name IS NOT NULL
```

Restrictions: PRIMARY KEY

PRIMARY KEY (primary key) is a field (or combination fields) that uniquely identifies the record.

PRIMARY KEY should:

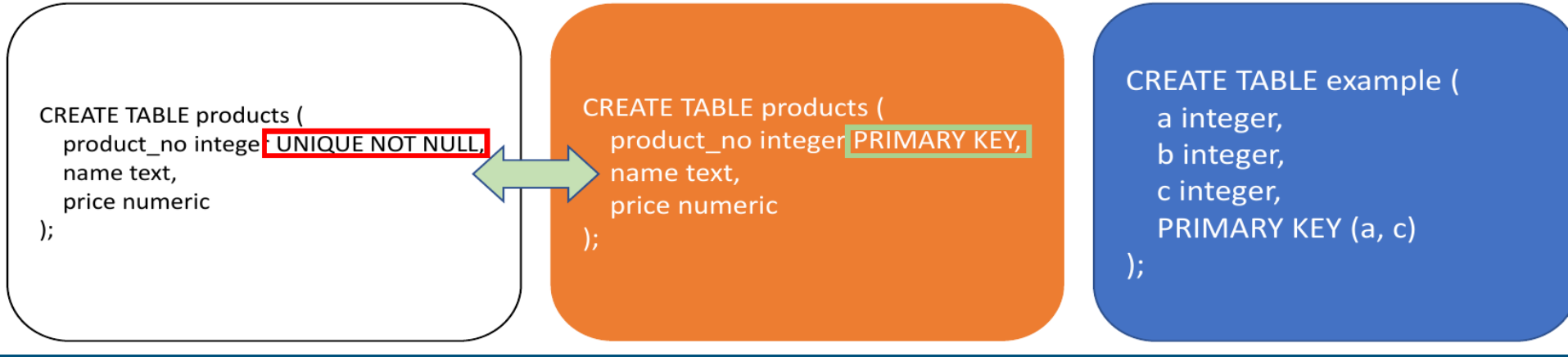
- contain unique values

- cannot contain NULL values.

Restrictions: PRIMARY KEY

A table cannot have two records with the same key value.

A table can only have one primary key.



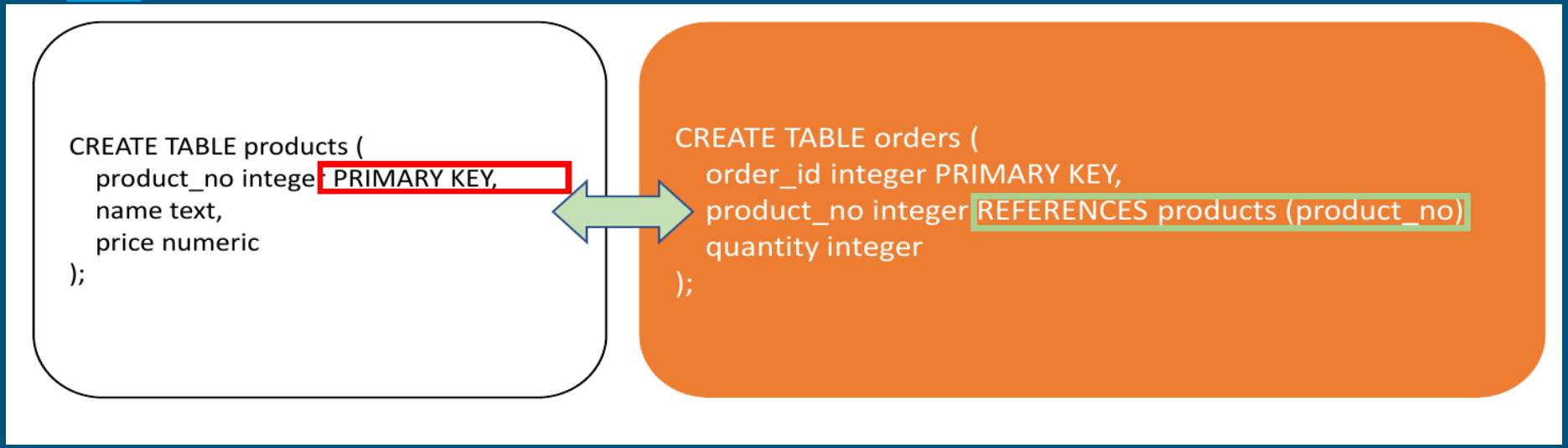
Restrictions: FOREIGN KEY

Foreign key - an attribute (or group of attributes) whose value can be repeated for several records.

Contains a reference to a primary key field in another table.

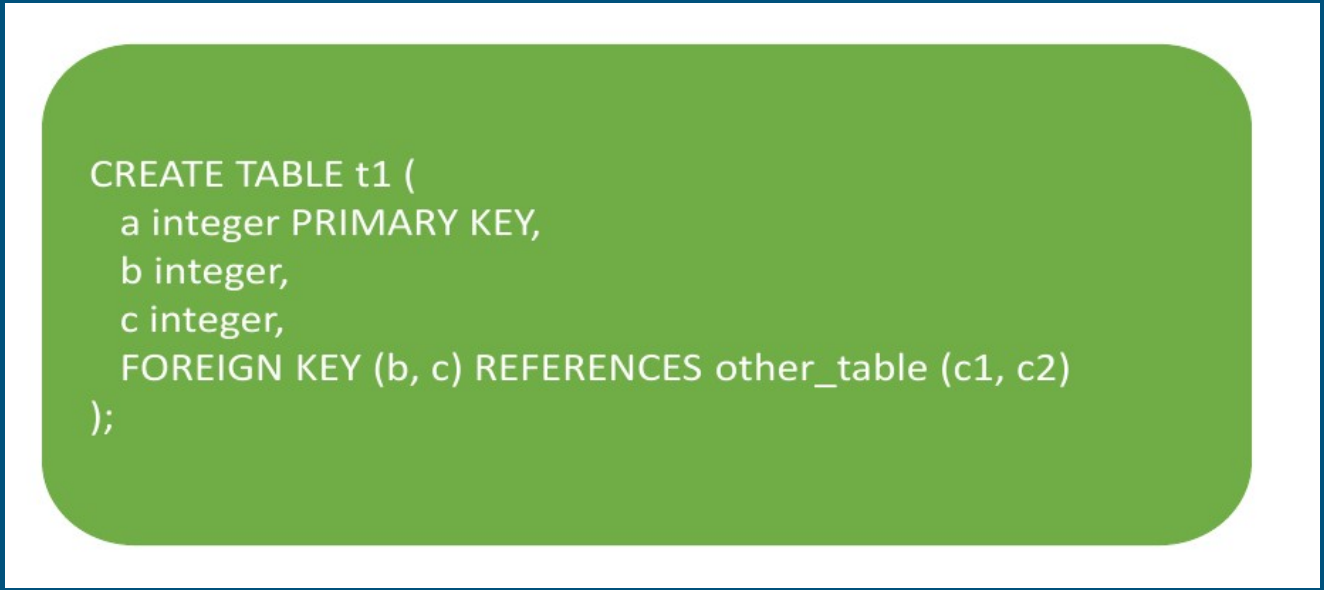
The table containing the foreign key is called a child table, containing the primary key - parent table.

Restrictions: FOREIGN KEY



Restrictions: FOREIGN KEY

FOREIGN KEY can refer to a group of columns of the target relation:



Restrictions: CHECK

CHECK - sets an arbitrary condition on the values of one or several columns in one table row.

• The CHECK constraint specifies an expression that returns a logical result that determines whether the add or change operation is successful for specific lines:

  • TRUE or UNKNOWN - operation completed successfully

  • FALSE - an error occurs, the operation does not change anything in the database

Restrictions: CHECK

```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);
```

Restrictions: DEFAULT

The value is specified by an expression without variables (in particular, cross-references to other columns of the current table are not allowed in it). Subqueries are also not allowed. The data type of the expression that specifies the default value must match the data type of the column.

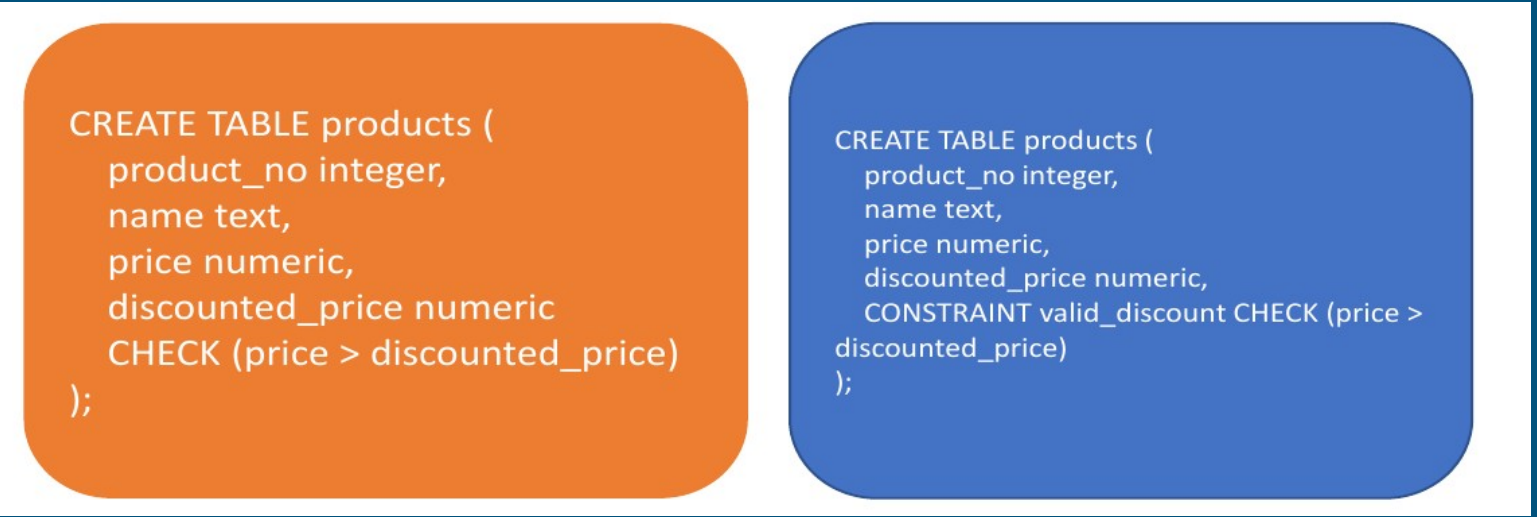
This expression will be used in all data append operations that do not specify a value for this column. If a default value is not defined, the value will be NULL.

Restrictions: DEFAULT

```
CREATE TABLE order (
        order_id INTEGER PRIMARY KEY,
        order_number INTEGER NOT NULL,
        order_date   DATE DEFAULT now()::date
);
```

Restrictions syntax

Constraints can be named using the command CONSTRAINT (this is useful for error output)



Keys: potential key

A potential key is a subset of relation attributes in a relational data model that satisfies the following requirements:

• uniqueness - there are not and cannot be two tuples of a given relation in which the values of this subset of attributes coincide (are equal)

• irreducibility (minimality) - the potential key does not contain a smaller subset of attributes that satisfies the uniqueness condition. In other words, if any attribute is removed from a potential key, it will lose its uniqueness property

A candidate key always exists, even if it includes all the attributes of the relation (follows from the properties of the relations). There can be several potential keys.

Keys: primary key

Primary key (PRIMARY KEY) is such a potential key of a relationship that is selected as the “primary”

- Any potential key is suitable as a primary key

Alternate key(s) – all other potential keys of the relationship

Keys: natural key

- Natural key – a key based on an existing relation attribute
- Surrogate key – based on a special, technical attribute that has no meaningful meaning (id). Despite the obvious advantages (immutability, guaranteed uniqueness, efficiency, etc.), surrogate keys also have their disadvantages (lack of information, use instead of normalization, vulnerability of generators)

Keys: foreign key

To reflect the functional dependencies between tuples of different relations, duplication of the primary key of one relation (parent) to another (child) is used. Attributes that are copies of parent relationship keys are called foreign keys.

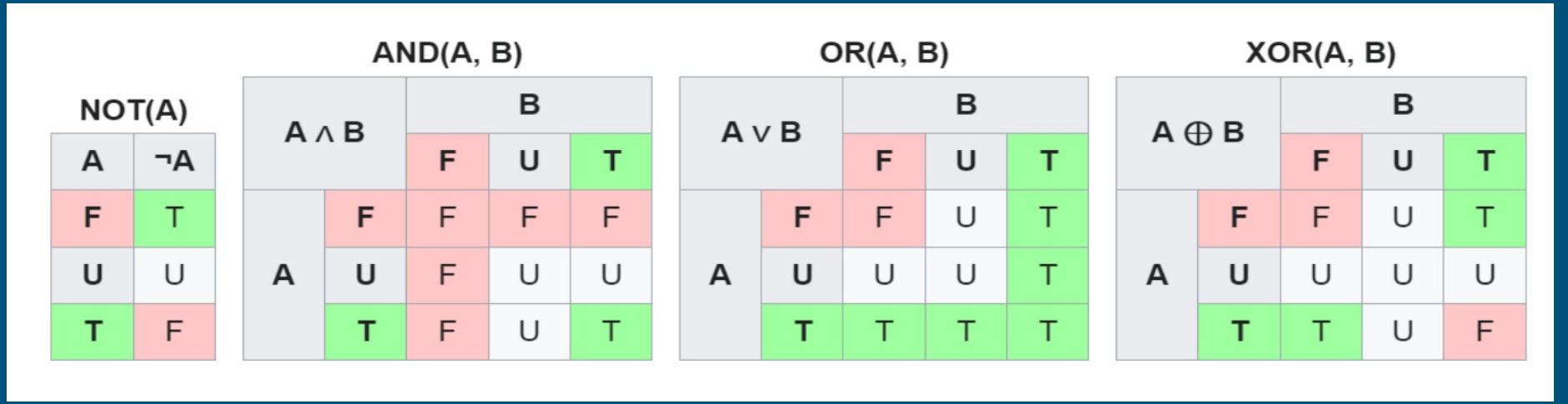
A foreign key in relation R2 is a non-empty subset FK of the set of attributes of this relation, such that:

1) There is a relation R1 with a potential key PK;

2) Each value of the foreign key FK in the current value of the relation R2 necessarily coincides with the value of the key PK of some tuple in the current value of the relation R1.

The relationships R1 and R2 are not necessarily different.

Ternary logic: True, False, Unknown



NOT(A)

A	¬A
F	T
U	U
T	F

AND(A, B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A, B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

XOR(A, B)

A ⊕ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	U
	T	T	U	F

Ternary logic

- IS [NOT] TRUE
- IS [NOT] FALSE
- IS [NOT] UNKNOWN

p	IS TRUE	IS FALSE	IS UNKNOWN		IS NOT TRUE	IS NOT FALSE	IS NOT UNKNOWN
TRUE	TRUE	FALSE	FALSE		FALSE	TRUE	TRUE
FALSE	FALSE	TRUE	FALSE		TRUE	FALSE	TRUE
UNKNOWN	FALSE	FALSE	TRUE		TRUE	TRUE	FALSE

Data Definition Language

ALTER - modification of objects

ALTER TABLE table_name ADD column_name datatype;

ALTER TABLE table_name DROP column_name;

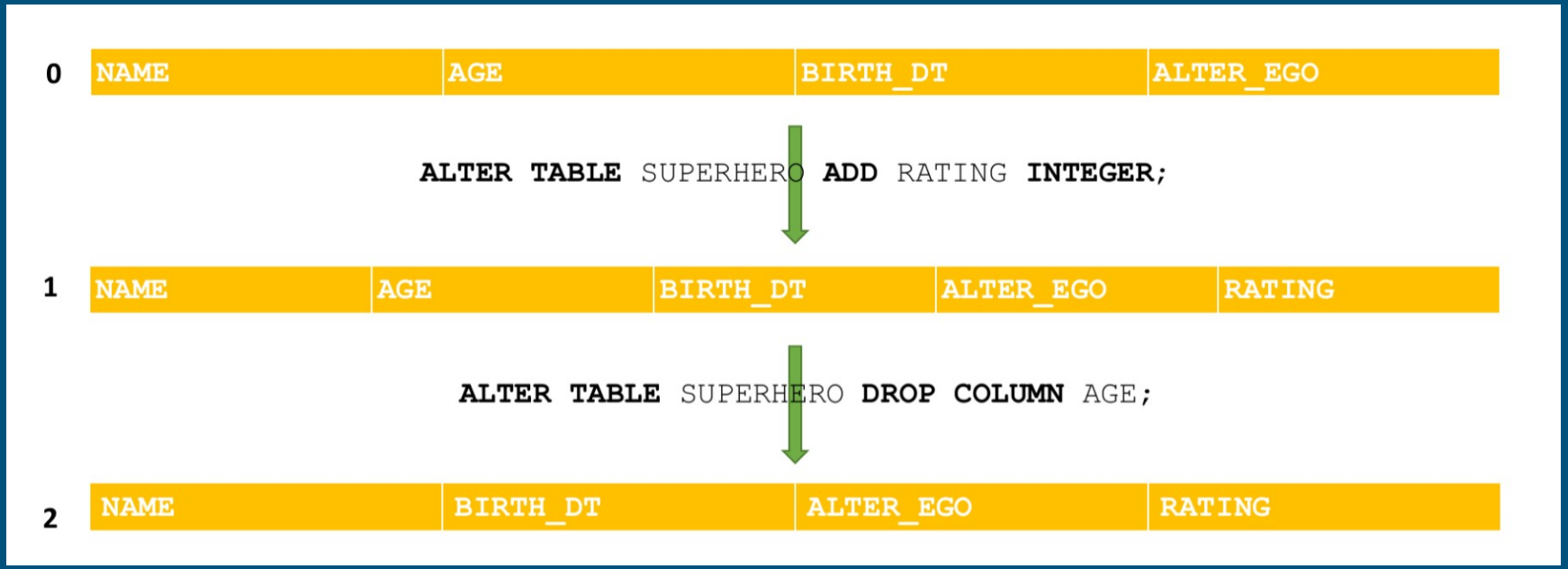
ALTER TABLE table_name RENAME column_name TO new_column_name;

ALTER TABLE table_name ALTER column_name TYPE datatype;

Alter

```
1. ALTER TABLE SUPERHERO ADD RATING INTEGER;

2. ALTER TABLE SUPERHERO DROP COLUMN AGE;
```



Data Definition Language

```
TRUNCATE TABLE SUPERHERO;
```

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

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90

NAME	BIRTH_DATE	ALTER_EGO	RATING

Drop

Deleting an object from the database

```
DROP TABLE SUPERHERO;
```

NAME	AGE	BIRTH_DT	ALTER_EGO

Data Manipulation Language

SELECT

- Selection of data that meets the specified conditions

INSERT

- Adding new data

UPDATE

- Change existing data

DELETE

- Delete existing data

DML: INSERT

```
INSERT
   INTO table_name [(comma_separated_column_names)]
VALUES (comma_separated_values);
```

Insert

```
INSERT
   INTO SUPERHERO (NAME, BIRTH_DT, ALTER_EGO, RATING)
VALUES ( 'Natasha Romanoff', '01-AUG-1999', 'Black Widow', 59 );
```

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1999	Black Widow	59

DML: UPDATE

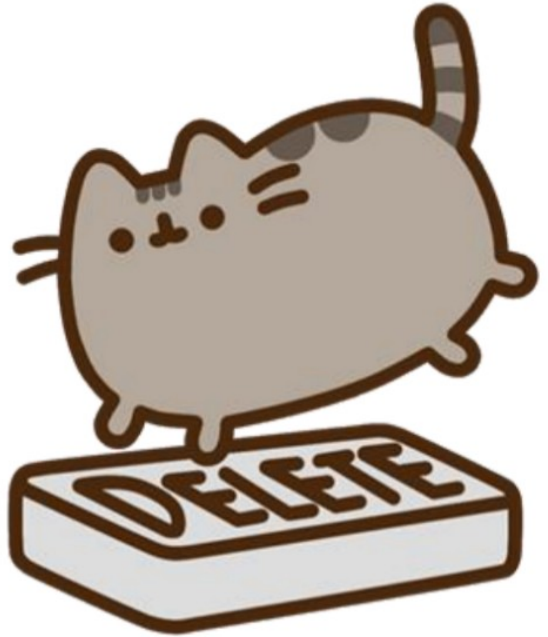
```
UPDATE table_name
   SET update_assignment_comma_list
[WHERE conditional_expression];
```

Update

```
UPDATE SUPERHERO
   SET BIRTH_DT = '01-AUG-1940'
 WHERE NAME = 'Natasha Romanoff';
```

NAME	BIRTH_DATE	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59

DML: DELETE




```
DELETE
   FROM table_name
[WHERE conditional_expression];
```

Delete

```
DELETE
  FROM SUPERHERO
 WHERE NAME = 'Bruce Banner';
```

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	100
~~Bruce Banner~~	~~28-FEB-1969~~	~~Hulk~~	~~80~~
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59

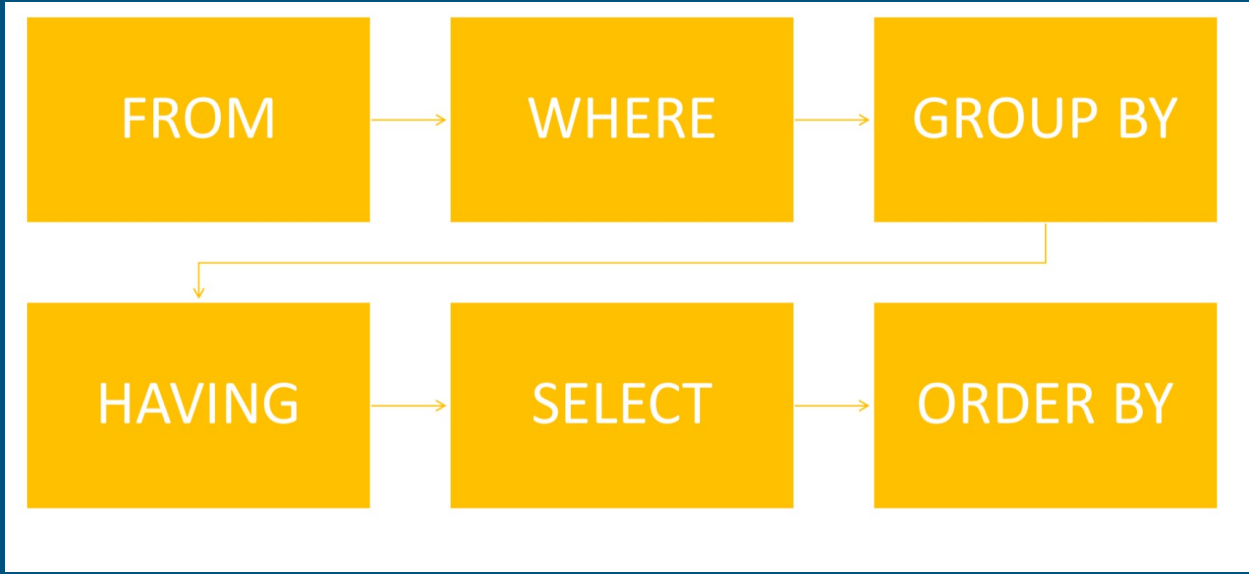
DML: DELETE vs DDL:TRUNCATE

Delete	**Truncate**
"Delete" line by line	Deleting the entire block at once
You can set conditions for deletion	Conditions for deletion cannot be set
Ability to roll back changes	There is no possibility of rolling back changes
Physically, the lines are not deleted, only marked "invisible" from a certain moment VACUUM is needed for removal	Deleting data and freeing up disk space occurs immediately

DML: SELECT

```
 SELECT [DISTINCT] select_item_comma_list
   FROM table_reference_comma_list
 [WHERE conditional_expression]
 [GROUP BY column_name_comma_list]
[HAVING conditional_expression]
 [ORDER BY order_item_comma_list];
```

The order of execution of the request



SELECT: FROM

```
SELECT ALTER_EGO
  FROM SUPERHERO;
```

```
SELECT NAME, ALTER_EGO, COMICS_N
  FROM SUPERHERO, COMICS;
```

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	98
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59
Thor	13-FEB-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Maximoff	22-OCT-1974	Quicksilver	82
Charles Xavier	30-JUN-1933	Professor X	100
Jean Grey	12-SEP-1961	Phoenix	93
Wade Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99

SELECT: WHERE

```
SELECT NAME, ALTER_EGO
  FROM SUPERHERO
 WHERE RATING > 90;
```

NAME	ALTER_EGO
Tony Stark	Iron Man
Charles Xavier	Professor X
Jean Grey	Phoenix
James Howlett	Wolverine

```
SELECT NAME, ALTER_EGO
  FROM SUPERHERO
 WHERE RATING < 50;
```

NAME	ALTER_EGO

Where

- `WHERE X = value_1 AND X <> value_2;`
- `WHERE X = value_1 OR X <> value_2;`
- `WHERE X = value_1 AND NOT X < value3;`

- `WHERE X < value_1 AND X > value_2 OR X = value_3`

priority: not, and, or

Useful functions

Sometimes it is useful to use special functions in a request:

IN - belonging to a certain set of values:

X IN (a1, a2, ..., an) ≣ X = a1 or X = a2 or ... or X = an

BETWEEN - belonging to a certain range of values:

X BETWEEN A AND B ≣ (X >= A and X <= B) or (X <= A and X >= B)

LIKE - satisfying the text pattern:

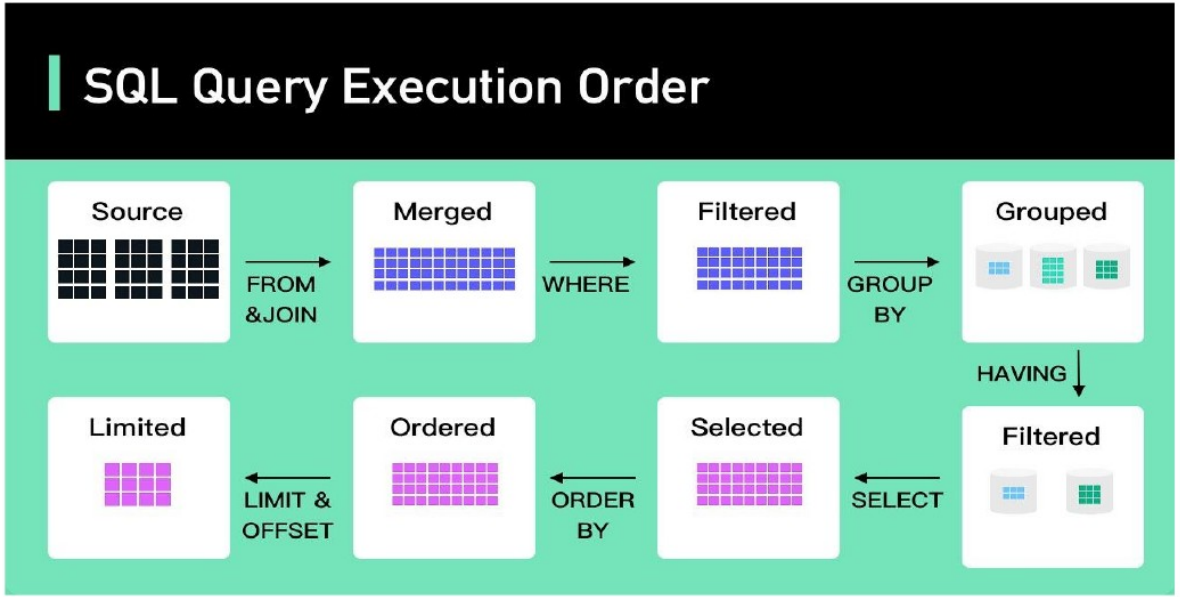
X LIKE '0%abc_0', where _ is exactly 1 character, and % is any sequence of characters (including zero length).

```
SELECT DATE_BIRTH
     , RATING
     , NAME
  FROM SUPERHERO
 WHERE RATING < 90
   AND RATING > 70
 GROUP BY DATE_BIRTH;
```

Why it would not work?

Steps

1. FROM / JOIN
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY
7. LIMIT / OFFSET



Aggregating functions

- count() – the number of records with a known value. If you need to count the number of unique values, you can use count(DISTINCT field_nm)
- max() - the largest of all selected field values
- min() - the smallest of all selected field values
- sum() - sum of all selected field values
- avg() - average of all selected field values

Count

```
SELECT count(ALTER_EGO)
  FROM SUPERHERO;
```

COUNT(ALTER_EGO)
12

NAME	BIRTH_DT	ALTER_EGO	RATING
Tony Stark	06-JAN-1966	Iron man	98
Bruce Banner	28-FEB-1969	Hulk	80
Steve Rogers	07-MAR-1921	Captain America	90
Natasha Romanoff	01-AUG-1940	Black Widow	59
Thor	13-FEB-1976	Thor	74
Clint Barton	17-DEC-1969	Hawkeye	55
Wanda Maximoff	22-OCT-1974	Scarlet Witch	81
Pietro Maximoff	22-OCT-1974	Quicksilver	82
Charles Xavier	30-JUN-1933	Professor X	100
Jean Grey	12-SEP-1961	Phoenix	93
Wade Wilson	13-APR-1980	Deadpool	89
James Howlett	01-JAN-1887	Wolverine	99

Group by

```
SELECT BIRTH_DT,
       count(ALTER_EGO)
  FROM SUPERHERO
 WHERE BIRTH_DT = '22-OCT-1974'
 GROUP BY BIRTH_DT;
```

BIRTH_DT	COUNT(ALTER_EGO)
22-OCT-1974	2

Having

- It is used in conjunction with GROUP BY to impose restrictions on the selection after grouping
- The restriction with the use of WHERE can be imposed only before grouping

```
GROUP BY column_name(s)
HAVING expression_clause
```

```
SELECT ITEM,
       avg(PRICE)
  FROM CATALOG
 GROUP BY ITEM;
```

ITEM	AVG(PRICE)
Computer	1035.67
Laptop	1000
Mobile phone	200
Printer	300
Scanner	200
Camera	525
Headphones	200

```
SELECT ITEM, avg(PRICE)
  FROM CATALOG
 GROUP BY ITEM
HAVING avg(PRICE) <= 500;
```

ITEM	AVG(PRICE)
Mobile phone	200
Printer	300
Scanner	200
Headphones	200

Order by

```
SELECT ID, ITEM, MAGAZINE, PRICE, DELIVERY
  FROM CATALOG
ORDER BY PRICE ASC;
```

ASC – ascending

DESC – descending

Distinct

```
SELECT DISTINCT
       ITEM
FROM CATALOG;
```

ID	ITEM	MAGAZINE	PRICE	DELIVERY
5	Mobile phone	1	150	15
9	Scanner	2	200	7
12	Headphones	4	200	0
7	Mobile phone	3	250	10
8	Printer	1	300	15
10	Camera	2	500	5
11	Camera	3	550	10
4	Laptop	1	999	15
1	Computer	1	1000	15
6	Laptop	2	1001	5
2	Computer	2	1007	5
3	Computer	3	1100	10

ITEM
Computer
Laptop
Mobile phone
Printer
Scanner
Camera
Headphones

Select *

```
SELECT *
  FROM CATALOG
 WHERE ITEM = 'Scanner';
```

ID	ITEM	MAGAZINE	PRICE	DELIVERY
9	Scanner	2	200	7

Select: alias

```
SELECT column_name AS alias_column_name
  FROM table_name alias_table_name;
```

```
SELECT MAGAZINE,
       count(ITEM) AS cnt_items
  FROM CATALOG
 WHERE DELIVERY < 15
 GROUP BY MAGAZINE
HAVING count(ITEM) > 1
 ORDER BY cnt_items;
```

More aliases

Alias Column Syntax

SELECT *column_name* AS *alias_name*
FROM *table_name;*

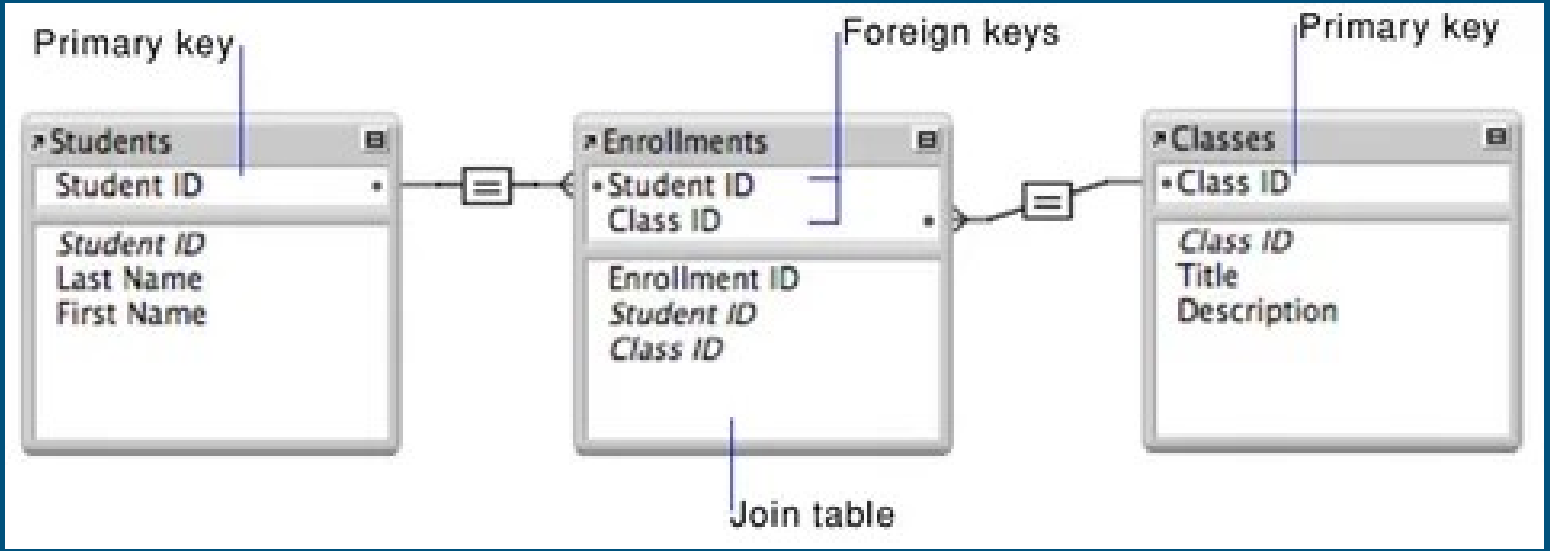
Alias Table Syntax

SELECT *column_name(s)*
FROM *table_name* AS *alias_name;*

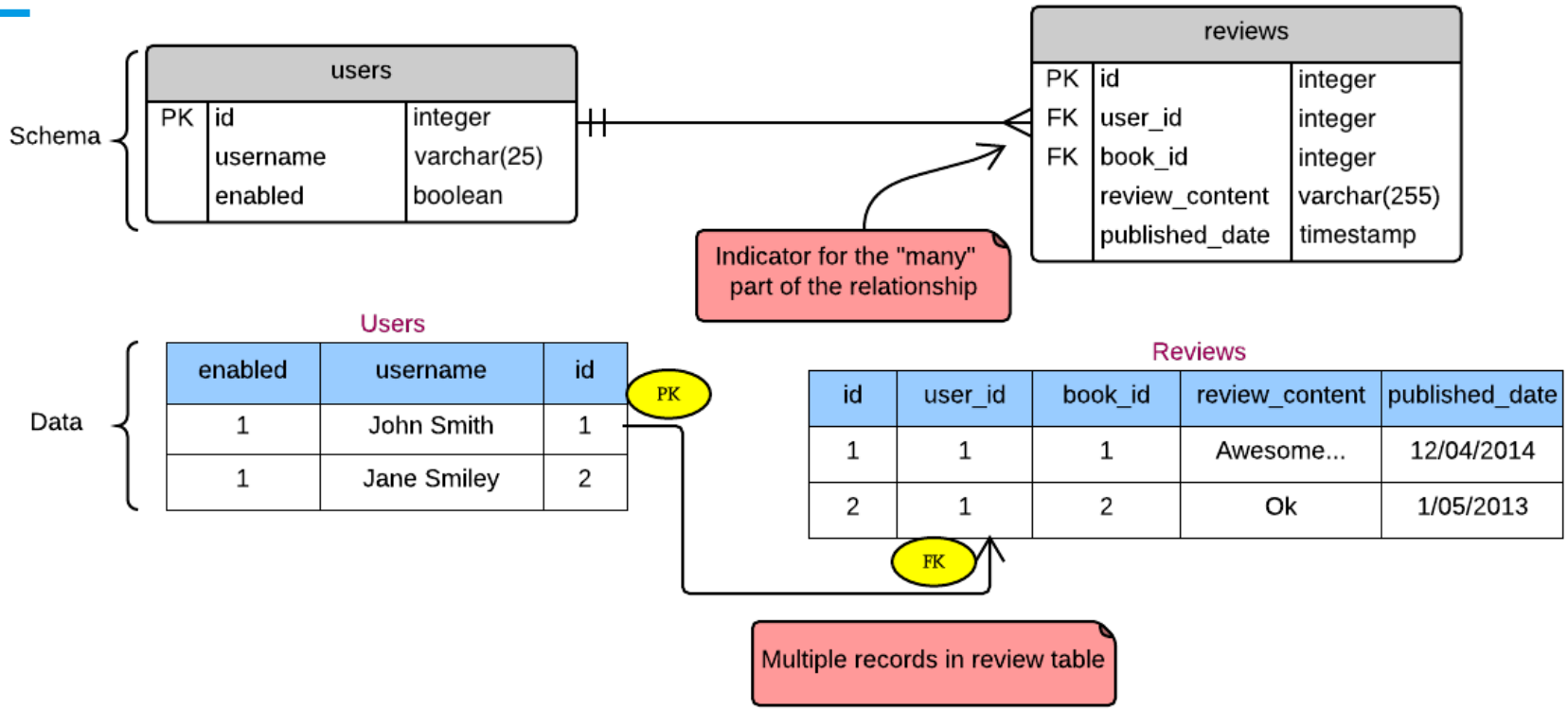
Relations between tables

- one to one
- one to many
- many to many

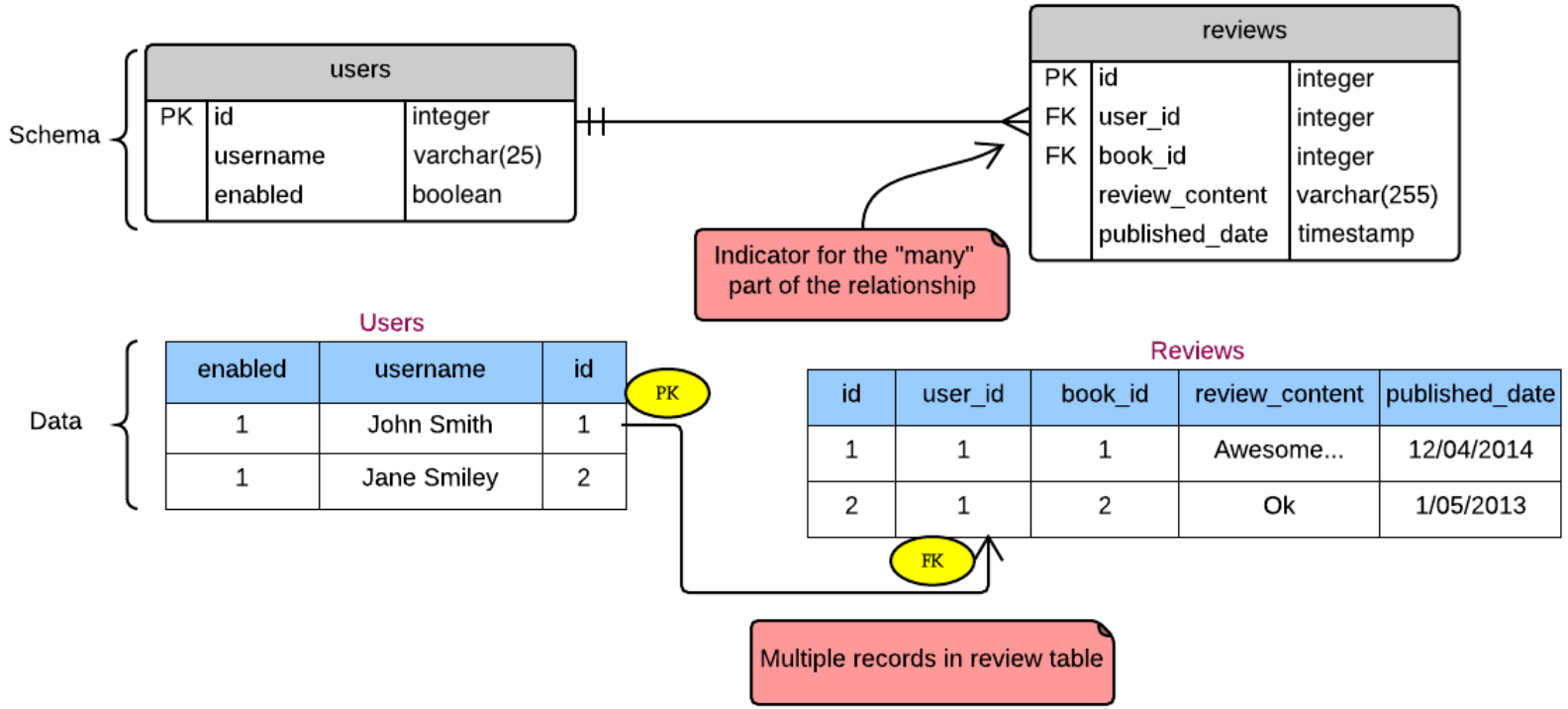
FOREIGN KEY


Primary key
Foreign keys
Primary key
Students
Student ID
Student ID
Last Name
First Name
Enrollments
Student ID
Class ID
Enrollment ID
Student ID
Class ID
Classes
Class ID
Class ID
Title
Description
Join table

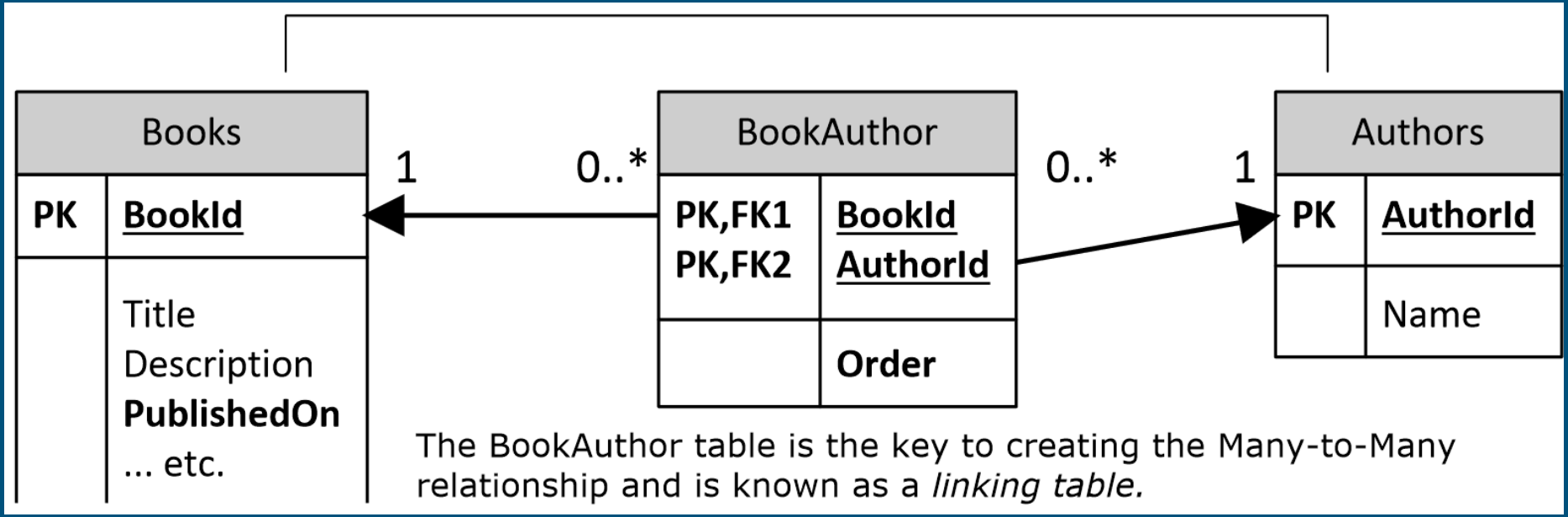
One-to-one


Schema
users
PK id integer
username varchar(25)
enabled boolean
reviews
PK id integer
FK user_id integer
FK book_id integer
review_content varchar(255)
published_date timestamp
Indicator for the "many" part of the relationship
Data
Users
enabled username id
1 John Smith 1
1 Jane Smiley 2
PK
Reviews
id user_id book_id review_content published_date
1 1 1 Awesome... 12/04/2014
2 1 2 Ok 1/05/2013
FK
Multiple records in review table

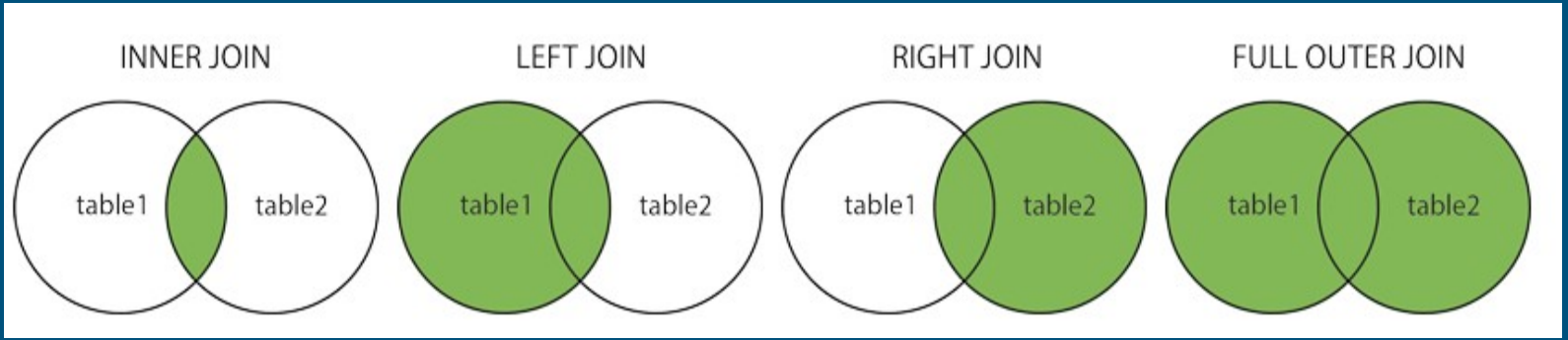
One-to-many


Schema
users
PK id integer
username varchar(25)
enabled boolean
reviews
PK id integer
FK user_id integer
FK book_id integer
review_content varchar(255)
published_date timestamp
Indicator for the "many" part of the relationship
Data
Users
enabled username id
1 John Smith 1
1 Jane Smiley 2
PK
Reviews
id user_id book_id review_content published_date
1 1 1 Awesome... 12/04/2014
2 1 2 Ok 1/05/2013
FK
Multiple records in review table

Many-to-many


Books
PK
BookId
Title
Description
PublishedOn
... etc.
1
0..*
BookAuthor
PK,FK1
PK,FK2
BookId
AuthorId
Order
0..*
1
Authors
PK
AuthorId
Name
The BookAuthor table is the key to creating the Many-to-Many relationship and is known as a *linking table.*

JOIN


INNER JOIN
table1
table2
LEFT JOIN
table1
table2
RIGHT JOIN
table1
table2
FULL OUTER JOIN
table1
table2

INNER JOIN

INNER JOIN: Returns records that have matching values in both tables

INNER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2* ON *table1.column_name = table2.column_name*

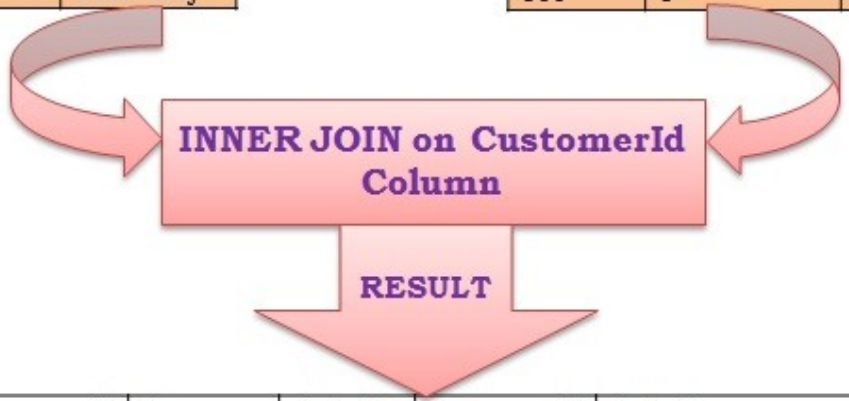
INNER JOIN

Customers

CustomerId	Name
1	Shree
2	Kalpana
3	Basavaraj

Orders

OrderId	CustomerId	OrderDate
100	1	2014-01-29 23:56:57.700
200	4	2014-01-30 23:56:57.700
300	3	2014-01-31 23:56:57.700



CustomerId	Name	OrderId	CustomerId	OrderDate
1	Shree	100	1	2014-01-30 23:48:32.850
3	Basavaraj	300	3	2014-02-01 23:48:32.853

LEFT JOIN

LEFT (OUTER) JOIN: Returns all records from the left table and matching records from the table on the right

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

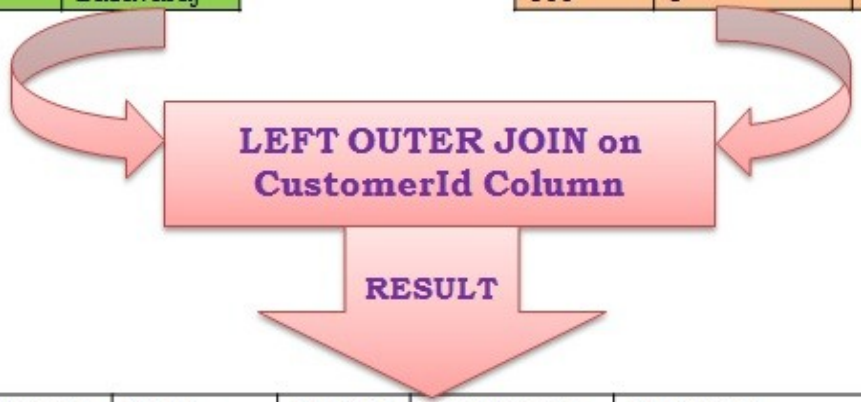
LEFT OUTER JOIN

Customers

CustomerId	Name
1	Shree
2	Kalpana
3	Basavaraj

Orders

OrderId	CustomerId	OrderDate
100	1	2014-01-29 23:56:57.700
200	4	2014-01-30 23:56:57.700
300	3	2014-01-31 23:56:57.700



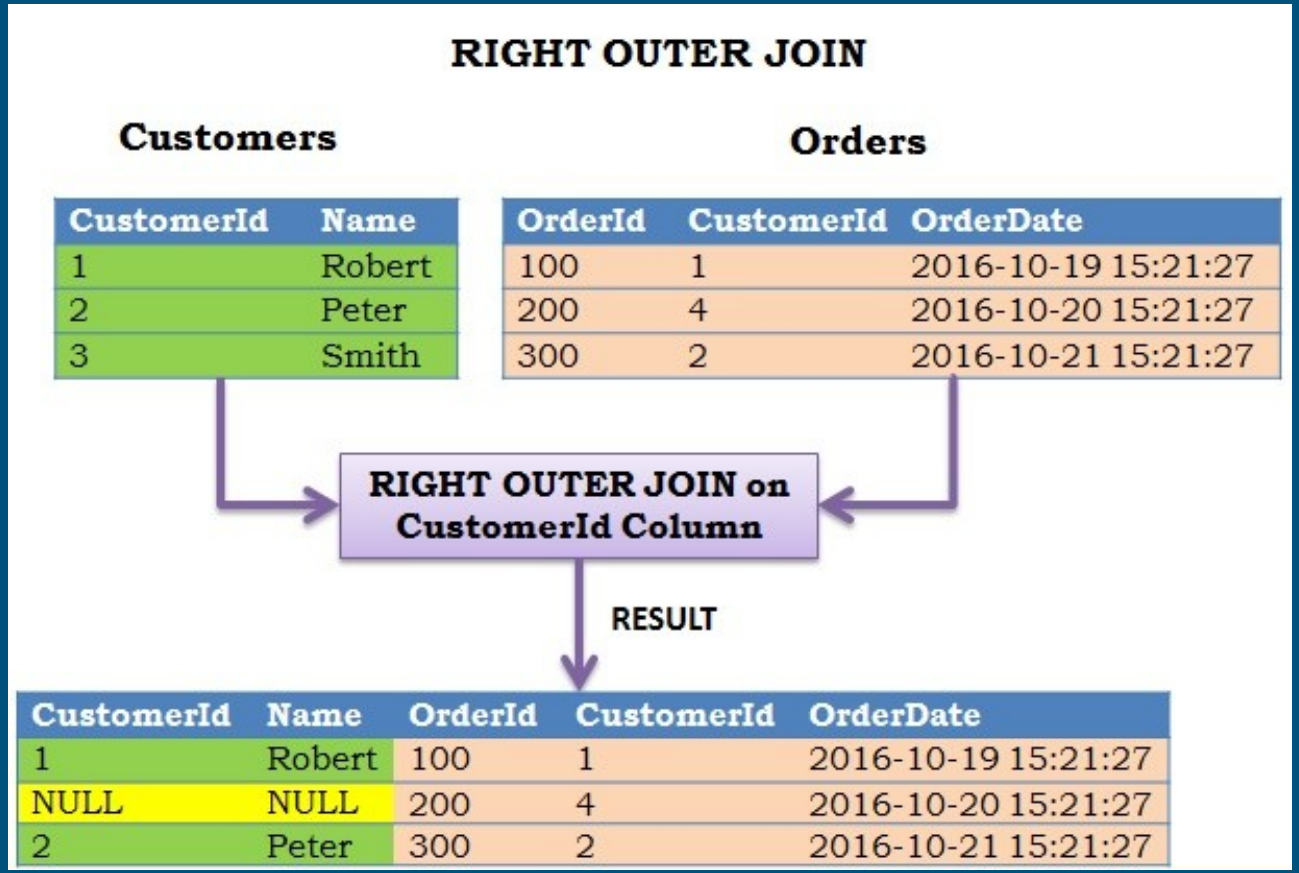
CustomerId	Name	OrderId	CustomerId	OrderDate
1	Shree	100	1	2014-01-30 23:48:32.850
2	Kalpana	NULL	NULL	NULL
3	Basavaraj	300	3	2014-02-01 23:48:32.853

RIGHT JOIN

RIGHT (OUTER) JOIN: Returns all records from the table on the right, and matching records from the left table

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;
```


RIGHT OUTER JOIN
Customers
Orders
CustomerId Name
1 Robert
2 Peter
3 Smith
OrderId CustomerId OrderDate
100 1 2016-10-19 15:21:27
200 4 2016-10-20 15:21:27
300 2 2016-10-21 15:21:27
RIGHT OUTER JOIN on CustomerId Column
RESULT
CustomerId Name OrderId CustomerId OrderDate
1 Robert 100 1 2016-10-19 15:21:27
NULL NULL 200 4 2016-10-20 15:21:27
2 Peter 300 2 2016-10-21 15:21:27

FULL JOIN

FULL (OUTER) JOIN: Returns all records when there is a match in the left or right table

FULL OUTER JOIN Syntax

SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2* ON *table1.column_name* = *table2.column_name*;

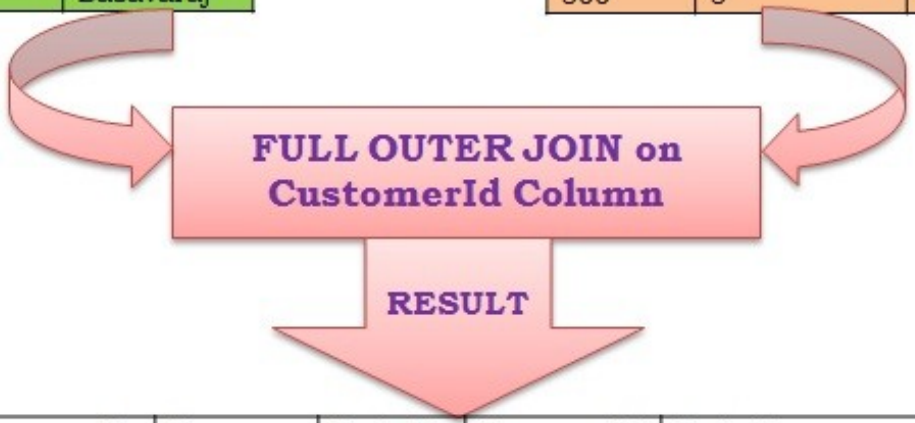
FULL OUTER JOIN

Customers

CustomerId	Name
1	Shree
2	Kalpana
3	Basavaraj

Orders

OrderId	CustomerId	OrderDate
100	1	2014-01-29 23:56:57.700
200	4	2014-01-30 23:56:57.700
300	3	2014-01-31 23:56:57.700



CustomerId	Name	OrderId	CustomerId	OrderDate
1	Shree	100	1	2014-01-30 23:48:32.850
2	Kalpana	NULL	NULL	NULL
3	Basavaraj	300	3	2014-02-01 23:48:32.853
NULL	NULL	200	4	2014-01-31 23:48:32.853

CROSS JOIN

CROSS JOIN = Cartesian product of two tables

```
SELECT column_name(s)
    FROM table1
    CROSS JOIN  table2
```

CROSS JOIN

Customers

CustomerId	Name
1	Shree
2	Kalpana
3	Basavaraj

Orders

OrderId	CustomerId	OrderDate
100	1	2014-01-29 23:56:57.700
200	4	2014-01-30 23:56:57.700
300	3	2014-01-31 23:56:57.700

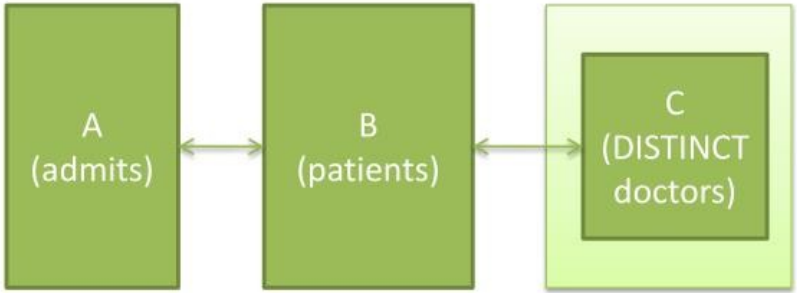
RESULT

CustomerId	Name	OrderId	CustomerId	OrderDate
1	Shree	100	1	2014-01-30 23:48:32.850
2	Kalpana	100	1	2014-01-30 23:48:32.850
3	Basavaraj	100	1	2014-01-30 23:48:32.850
1	Shree	200	4	2014-01-31 23:48:32.853
2	Kalpana	200	4	2014-01-31 23:48:32.853
3	Basavaraj	200	4	2014-01-31 23:48:32.853
1	Shree	300	3	2014-02-01 23:48:32.853
2	Kalpana	300	3	2014-02-01 23:48:32.853
3	Basavaraj	300	3	2014-02-01 23:48:32.853

Subquery example

```
PROC SQL ;
CREATE TABLE prim2 AS
 SELECT pt_id, admdate, disdate, hosp, md_id,
        b.lastname AS ptname,
        c.lastname AS mdname
   FROM ex.admits a, ex.patients b,
     (SELECT DISTINCT md_id, lastname
        FROM ex.doctors) c
   WHERE (a.pt_id EQ b.id)  AND
         (a.md EQ b.primmd)  AND
          (a.md EQ c.md_id)
   ORDER BY a.pt_id, admdate ;
QUIT;
```

Create inline table C to join with A and B using IDs



Subquery

Subqueries (also known as inner queries or nested queries) are a tool for performing operations in multiple steps. For example, if you wanted to take the sums of several columns, then average all of those values, you'd need to do each aggregation in a distinct step.

Subquery example 1

```
SELECT sub.*
  FROM (
        SELECT *
          FROM tutorial.sf_crime_incidents_2014_01
         WHERE day_of_week = 'Friday'
       ) sub
 WHERE sub.resolution = 'NONE'
```

Subquery example 2

```
SELECT *
  FROM tutorial.sf_crime_incidents_2014_01
 WHERE Date = (SELECT MIN(date)
                 FROM tutorial.sf_crime_incidents_2014_01
              )
```

Subquery example 3

```
SELECT *
  FROM tutorial.sf_crime_incidents_2014_01
 WHERE Date IN (SELECT date
                  FROM tutorial.sf_crime_incidents_2014_01
                 ORDER BY date
                 LIMIT 5
               )
```

Subquery example 4

```
SELECT incidents.*,  sub.incidents AS incidents_that_day
  FROM tutorial.sf_crime_incidents_2014_01 incidents
  JOIN ( SELECT date,   COUNT(incidnt_num) AS incidents
           FROM tutorial.sf_crime_incidents_2014_01
         GROUP BY 1   ) sub
     ON incidents.date = sub.date
 ORDER BY sub.incidents DESC, time
```

UNIONS

```
SELECT *
  FROM tutorial.crunchbase_investments_part1
 UNION ALL
 SELECT *
   FROM tutorial.crunchbase_investments_part2;
```

SQL has strict rules for appending data:

1. Both tables must have the same number of columns
2. The columns must have the same data types in the same order as the first table