***re1. How triggers are helpful in database system***

A **database trigger** is [procedural code](https://en.wikipedia.org/wiki/Procedural_code) that is automatically executed in response to certain [events](https://en.wikipedia.org/wiki/Event_(computing)) on a particular [table](https://en.wikipedia.org/wiki/Table_(database)) or [view](https://en.wikipedia.org/wiki/View_(database)) in a [database](https://en.wikipedia.org/wiki/Database). The trigger is mostly used for maintaining the [integrity](https://en.wikipedia.org/wiki/Database_integrity) of the information on the database. For example, when a new record (representing a new worker) is added to the employees table, new records should also be created in the tables of the taxes, vacations and salaries.

triggers are commonly used to

* automatically generate derived column values
* prevent invalid transactions
* enforce complex security authorizations
* enforce referential integrity across nodes in a distributed database
* enforce complex business rules
* provide transparent event logging
* provide sophisticated auditing
* maintain synchronous table replicates
* gather statistics on table access

***2. Difference between where and having clause***

**HAVING** specifies a search condition for a group or an aggregate function used in SELECT statement. **HAVING** can be used only with the SELECT statement. **HAVING** is typically used in a GROUP BY **clause**. When GROUP BY is not used, **HAVING** behaves like a WHERE **clause**

HAVING specifies a search condition for a group or an aggregate function used in SELECT statement.

HAVING can be used only with the SELECT statement. HAVING is typically used in a GROUP BY clause. When GROUP BY is not used, HAVING behaves like a WHERE clause.

A HAVING clause is like a WHERE clause, but applies only to groups as a whole, whereas the WHERE clause applies to individual rows. A query can contain both a WHERE clause and a HAVING clause. The WHERE clause is applied first to the individual rows in the tables . Only the rows that meet the conditions in the WHERE clause are grouped. The HAVING clause is then applied to the rows in the result set. Only the groups that meet the HAVING conditions appear in the query output. You can apply a HAVING clause only to columns that also appear in the GROUP BY clause or in an aggregate function. (Reference :BOL)

Example of HAVING and WHERE in one query:

SELECT titles.pub\_id, AVG(titles.price)

FROM titles INNER JOIN publishers

ON titles.pub\_id = publishers.pub\_id

WHERE publishers.state = 'CA'

GROUP BY titles.pub\_id

HAVING AVG(titles.price) > 10

Sometimes you can specify the same set of rows using either a WHERE clause or a HAVING clause. In such cases, one method is not more or less efficient than the other. The optimizer always automatically analyzes each statement you enter and selects an efficient means of executing it. It is best to use the syntax that most clearly describes the desired result. In general, that means eliminating undesired rows in earlier clauses.

***3. functional dependency-define+example***

Functional dependency (FD) is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have same values for attributes A1, A2,..., An, then those two tuples must have to have same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign (→) that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

### Employee Department Model[[edit](https://en.wikipedia.org/w/index.php?title=Functional_dependency&action=edit&section=4)]

A classic example of functional dependency is the employee, department model. The following table

|  |  |  |  |
| --- | --- | --- | --- |
| **Employee ID** | **Employee Name** | **Department ID** | **Department Name** |
| 0001 | John Doe | 1 | Human Resources |
| 0002 | Jane Doe | 2 | Marketing |
| 0003 | John Smith | 1 | Human Resources |
| 0004 | Jane Goodall | 3 | Sales |

This case represents an example where multiple functional dependencies are embedded in a single representation of data. Note that because an employee can only be a member of one department, the unique ID of that employee determines the department.

* Employee ID → Employee Name
* Employee ID → Department ID

In addition to this relationship, the table also has a functional dependency through a non-key attribute

* Department ID → Department Name

## *4. Objectives of Normalization*

Develop a good description of the data, its relationships and constraints

Produce a stable set of relations that

* Is a faithful model of the enterprise
* Is highly flexible
* Reduces redundancy-saves space and reduces inconsistency in data
* Is free of update, insertion and deletion anomalies

***5. data consistency:-*** A transaction either creates a new and valid state of data, or, if any failure occurs, returns all data to its state before the transaction was started.

Consistency is one of the four guarantees that define [ACID](https://en.wikipedia.org/wiki/ACID) [transactions](https://en.wikipedia.org/wiki/Database_transaction); however, significant ambiguity exists about the nature of this guarantee. It is defined variously as:

* The guarantee that any transactions started in the future necessarily see the effects of other transactions committed in the past[[1]](https://en.wikipedia.org/wiki/Consistency_(database_systems)#cite_note-CAP_Theorem_Paper-1)[[2]](https://en.wikipedia.org/wiki/Consistency_(database_systems)#cite_note-Ports_et_al-2)
* The guarantee that [database constraints](https://en.wikipedia.org/wiki/Relational_database#Constraints) are not violated, particularly once a transaction commits[[3]](https://en.wikipedia.org/wiki/Consistency_(database_systems)#cite_note-Haerder_.26_Reuter-3)[[4]](https://en.wikipedia.org/wiki/Consistency_(database_systems)#cite_note-4)[[5]](https://en.wikipedia.org/wiki/Consistency_(database_systems)#cite_note-5)[[6]](https://en.wikipedia.org/wiki/Consistency_(database_systems)#cite_note-6)
* The guarantee that operations in transactions are performed accurately, correctly, and with validity, with respect to application semantics

The [consistency](https://en.wikipedia.org/wiki/Consistency_(database_systems)) property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including [constraints](https://en.wikipedia.org/wiki/Integrity_constraints), [cascades](https://en.wikipedia.org/wiki/Cascading_rollback), [triggers](https://en.wikipedia.org/wiki/Database_trigger), and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined rules.

Examples[[edit](https://en.wikipedia.org/w/index.php?title=ACID&action=edit&section=6)]

The following examples further illustrate the ACID properties. In these examples, the database table has two columns, A and B. An [integrity constraint](https://en.wikipedia.org/wiki/Integrity_constraints) requires that the value in A and the value in B must sum to 100. The following [SQL](https://en.wikipedia.org/wiki/SQL) code creates a table as described above:

**CREATE** **TABLE** acidtest (A INTEGER, B INTEGER, **CHECK** (A + B = 100));

### Consistency failure

Consistency is a very general term, which demands that the data must meet all validation rules. In the previous example, the validation is a requirement that A + B = 100. Also, it may be inferred that both A and B must be integers. A valid range for A and B may also be inferred. All validation rules must be checked to ensure consistency. Assume that a transaction attempts to subtract 10 from A without altering B. Because consistency is checked after each transaction, it is known that A + B = 100 before the transaction begins. If the transaction removes 10 from A successfully, atomicity will be achieved. However, a validation check will show that A + B = 90, which is inconsistent with the rules of the database. The entire transaction must be cancelled and the affected rows rolled back to their pre-transaction state. If there had been other constraints, triggers, or cascades, every single change operation would have been checked in the same way as above before the transaction was committed.

***6. differences between simple and complex views.***  
  
1. A Simple view selects from one table. A Complex view selects from one or more tables.  
2. A Simple view does not contain functions but Complex views contain functions.  
3. You can perform DML through Simple views but you cannot always perform DML through Complex views.

### Creating a simple View -

**sql> CREATE OR REPLACE VIEW TEST\_VIEW**  
**AS SELECT \* FROM EMP\_MASTER;**  
  
**sql> CREATE VIEW TEST\_VIEW**  
**AS SELECT EMPLOYEE\_ID, EMP\_NAME, EMAIL, PHONE\_NUMBER**  
**FROM EMP\_MASTER;**

### Creating a Complex View -

**sql> CREATE VIEW TEST\_VIEW AS**  
**SELECT B.DEPT\_NAME, B.LOCATION, SUM (A.SALARY) "TOTAL SAL"**  
**FROM EMP\_MASTER A, DEPT\_DETAIL B**  
**WHERE A.DEPT\_ID=B.DEPT\_NO**  
**GROUP BY B.DEPT\_NAME, B.LOCATION;**

# 7. Two-phase locking

In [databases](https://en.wikipedia.org/wiki/Database) and [transaction processing](https://en.wikipedia.org/wiki/Transaction_processing), **two-phase locking** (**2PL**) is a [concurrency control](https://en.wikipedia.org/wiki/Concurrency_control) method that guarantees [serializability](https://en.wikipedia.org/wiki/Serializability).

By the 2PL protocol locks are applied and removed in two phases:

1. Expanding phase: locks are acquired and no locks are released.
2. Shrinking phase: locks are released and no locks are acquired.

Two types of locks are utilized by the basic protocol: *Shared* and *Exclusive* locks. Refinements of the basic protocol may utilize more lock types. Using locks that block processes, 2PL may be subject to [deadlocks](https://en.wikipedia.org/wiki/Deadlock) that result from the mutual blocking of two or more transactions.

***8. relationship between 2 tables***

Kinds of table relationships

* One-to-many relationships
* Many-to-many relationships
* One-to-one relationships

designing a relational database is dividing the data elements into related tables.

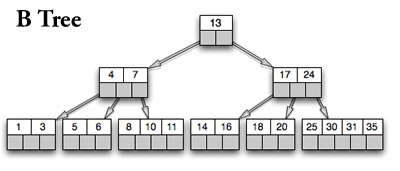
 **One-to-one:** Both tables can have only one record on either side of the relationship. Each primary key value relates to only one (or no) record in the related table. They're like spouses—you may or may not be married, but if you are, both you and your spouse have only one spouse. Most one-to-one relationships are forced by business rules and don't flow naturally from the data. In the absence of such a rule, you can usually combine both tables into one table without breaking any normalization rules.

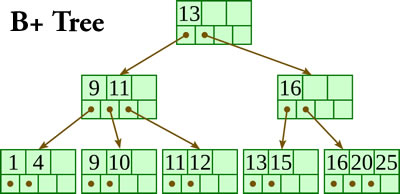
 **One-to-many:** The primary key table contains only one record that relates to none, one, or many records in the related table. This relationship is similar to the one between you and a parent. You have only one mother, but your mother may have several children.

 **Many-to-many:** Each record in both tables can relate to any number of records (or no records) in the other table. For instance, if you have several siblings, so do your siblings (have many siblings). Many-to-many relationships require a third table, known as an associate or linking table, because relational systems can't directly accommodate the relationship.

***9. Comparison between B Tree and B+ Tree:***

|  |  |  |
| --- | --- | --- |
|  | **B Tree** | **B+ Tree** |
| Short web descriptions | A B tree is an organizational structure for information storage and retrieval in the form of a tree in which all terminal nodes are at the same distance from the base, and all non-terminal nodes have between n and 2 n sub-trees or pointers (where n is an integer). | B+ tree is an n-array tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children. |
| Also known as | Binary tree. | B plus tree. |
| Space | O(n) | O(n) |
| Search | O(log n) | O(logb n) |
| Insert | O(log n) | O(logb n) |
| Delete | O(log n) | O(logb n) |
| Stores | In a B tree, search keys and data stored in internal or leaf nodes. | In a B+ tree, data stores only leaf nodes. |
| Space consuming | These trees consume space. | There trees do not consume space. |
| Function of leaf nodes | In B tree, the leaf node cannot store using linked list. | In B+ tree, leaf node data are ordered in a sequential linked list. |
| Searching | Here, searching becomes difficult in B- tree as data cannot be found in the leaf node. | Here, searching of any data in a B+ tree is very easy because all data is found in leaf nodes. |
| Search accessibility | Here in B tree the search is not that easy as compared to a B+ tree. | Here in B+ tree the searching becomes easy. |
| Redundant key | They do not store redundant search key. | They store redundant search key. |
| Applications | They are an older version and are not that advantageous as compared to the B+ trees. | Many database system implementers prefer the structural simplicity of a B+ tree. |





***10 three-schema approach purpose***

The **three-schema approach**, or the *Three Schema Concept*, in [software engineering](https://en.wikipedia.org/wiki/Software_engineering) is an approach to building [information systems](https://en.wikipedia.org/wiki/Information_systems) and systems [information management](https://en.wikipedia.org/wiki/Information_management) from the 1970s. It proposes to use three different [views](https://en.wikipedia.org/wiki/View_model) in systems development, in which [conceptual modelling](https://en.wikipedia.org/wiki/Conceptual_model) is considered to be the key to achieving [data integration](https://en.wikipedia.org/wiki/Data_integration).

The three-schema approach offers three types of schemas with schema techniques based on formal language descriptions:[[3]](https://en.wikipedia.org/wiki/Three_schema_approach#cite_note-3)

* External schema for user views
* [Conceptual schema](https://en.wikipedia.org/wiki/Conceptual_schema) integrates external schemata
* Internal schema that defines physical storage structures

The objective of the three-level architecture is to separate the users’ view,

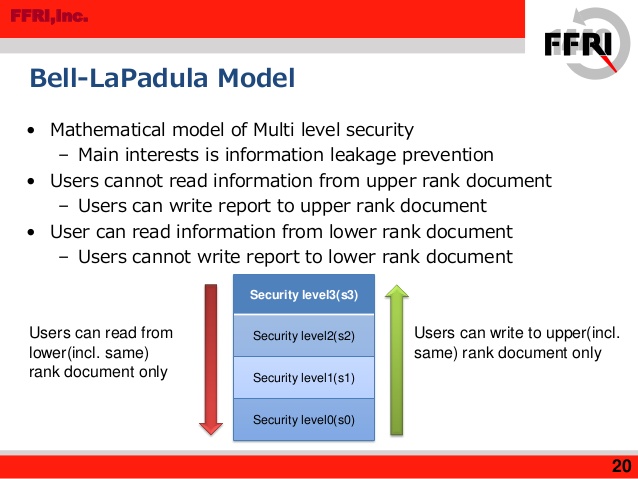
* It allows independent customized user views: Each user should be able to access the same data, but have a different customized view of the data. These should be independent: changes to one view should not affect others.
* It hides the physical storage details from users: Users should not have to deal with physical database storage details.
* The database administrator should be able to change the database storage structures without affecting the users’ views.
* The internal structure of the database should be unaffected by changes to the physical aspects of the storage: For example, a changeover to a new disk.

Some important facts about this level are:

1. DBA works at this level.
2. Describes the structure of all users.
3. Only DBA can define this level.
4. Global view of database.
5. Independent of hardware and software.

* Internal Level: The internal level involves how the database is physically represented on the computer system. It describes how the data is actually stored in the database and on the computer hardware.

***11. bell lapadula model***



# 12. Database Designer Job Description

**PRIMARY RESPONSIBILITIES**

**1.Determine purpose of database.**

**2.Gather information that will be recorded in database.**

**3. Divide information into tables.**

**4. Turn information into columns.**

**5.Specify primary keys.**

**6. Set up table relationships.**

**7. Refine and enhance design.**

**8. Apply normalization rules to ensure tables are structured correctly.**

**9. Make adjustments as necessary.**

**10. Follow W3C programming standards to address accessibility guidelines.**

**11.Develop structural design of various systems, applications, and databases for custom database-driven website.**

**12. Coordinate information systems with program objectives.**

**13. Code, document, and test programs.**

**14. Create data migration/conversion techniques for system conversions or upgrades.**

**15. Assist organizational members and clients with database-driven websites, network, database, and application training and technical support.**

**16. Assist staff with internal systems, applications, and databases, including developing procedures, forms, and other organizational tools.**

***13. difference between primary key and super key***

**Employee (**  
**Employee ID,**  
**FullName,**  
**SSN,**  
**DeptID**  
**)**  
  
**1. Candidate Key:** are individual columns in a table that qualifies for uniqueness of all the rows. Here in Employee table **EmployeeID** & **SSN** are Candidate keys.  
  
**2. Primary Key:** is the columns you choose to maintain uniqueness in a table. Here in Employee table you can choose either **EmployeeID** or **SSN** columns, **EmployeeID** is preferable choice, as SSN is a secure value.  
  
**3. Alternate Key:** Candidate column other the Primary column, like if EmployeeID is PK then **SSN** would be the Alternate key.  
  
**4. Super Key:** If you add any other column/attribute to a Primary Key then it become a super key, like **EmployeeID + FullName** is a Super Key.  
  
**5. Composite Key:** If a table do have a single columns that qualifies for a Candidate key, then you have to select 2 or more columns to make a row unique. Like if there is no**EmployeeID** or **SSN** columns, then you can make **FullName + DateOfBirth** as Composite primary Key. But still there can be a narrow chance of duplicate row.

***14. difference between relationship type and relationship set***

A relationship type R among n entity types E1, E2, …, En is a set of associations among entities from these types. Actually, R is a set of relationship instances ri where each ri is an n-tuple of entities (e1, e2, …, en), and each entity ej in ri is a member of entity type Ej, 1≤j≤n. Hence, a relationship type is a mathematical relation on E1, E2, …, En, or alternatively it can be defined as a subset of the Cartesian product E1x E2x … xEn . Here, entity types E1, E2, …, En defines a set of relationship, called relationship sets.

***15. partial key***

Weak Entity: An entity that is dependent on another entity.

Partial Key: Specifies a key that that is only partially unique. Used for weak entities.

vs

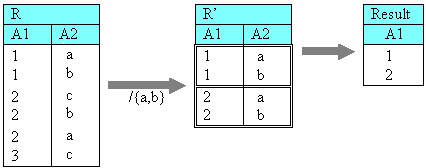
Foreign Key: A key that is used to establish and enforce a relation between data in different tables.

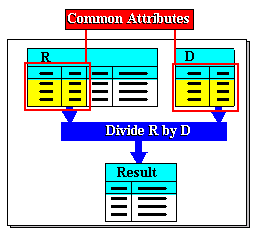
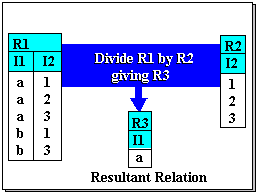
***partial key:-***A [key](http://dictionary.reference.com/browse/key) which identifies a subset of a set of information items (e.g. database "[records](http://dictionary.reference.com/browse/records)"), and which could narrow the subset to one item if other partial key(s) were combined with it.

**Partial Key:** It is a set of attributes that can uniquely identify weak entities and that are related to same owner entity. It is sometime called as Discriminator.

***16. division operation-define+purpose.***

As the name of this operation implies, it involves dividing one relation by another. Division is in principle a partitioning operation. Thus, 6 ÷ 2 can be paraphrased as partitioning a single group of 6 into a number of groups of 2 - in this  case, 3 groups of 2.

The illustration above shows how we may divide a relation R, which is a simple binary relation in this case with two attributes A1 and A2. For clarity, the values of attribute A1 have been sorted so that a given value appears in contiguous rows (where there's more than one). The question we're interested in is which of these values have in common an arbitrary subset of values of attribute A2.

We can now specify the form of the operation:

**divide <dividend-relation-name> by <divisor-relation-name>**

**giving <result-relation-name>**

<dividend-relation-name> and <divisor-relation-name> must be names of defined relations or results of previous operations. <result-relation-name> must be a unique name used to denote the result relation. As mentioned above, the divisor must share attributes with the dividend. In fact, we shall insist (on a stronger condition) that the intension of the divisor must be a subset of the dividend's. This is not really a restriction as any relation that shares attributes with the dividend can be turned into the required form simply by projecting over them.

The Divide operation takes the form:

**divide  by  giving **

# / (Divide) (Transact-SQL)

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635846752425826566

Applies To: Azure SQL Data Warehouse, Azure SQL Database, Parallel Data Warehouse, SQL Server (starting with 2008)

Divides one number by another (an arithmetic division operator).

Topic link icon [Transact-SQL Syntax Conventions](https://msdn.microsoft.com/en-IN/library/ms177563.aspx)

## [Syntax](javascript:void(0))

dividend / divisor

## [Arguments](javascript:void(0))

dividend

Is the numeric expression to divide. dividend can be any valid [expression](https://msdn.microsoft.com/en-IN/library/ms190286.aspx) of any one of the data types of the numeric data type category, except the **datetime** and **smalldatetime** data types.

divisor

Is the numeric expression by which to divide the dividend. divisor can be any valid expression of any one of the data types of the numeric data type category, except the **datetime** and **smalldatetime**data types.

## [Result Types](javascript:void(0))

Returns the data type of the argument with the higher precedence. For more information, see [Data Type Precedence (Transact-SQL)](https://msdn.microsoft.com/en-IN/library/ms190309.aspx).

If an integer dividend is divided by an integer divisor, the result is an integer that has any fractional part of the result truncated.

## [Remarks](javascript:void(0))

The actual value returned by the / operator is the quotient of the first expression divided by the second expression.

**Conclusion**

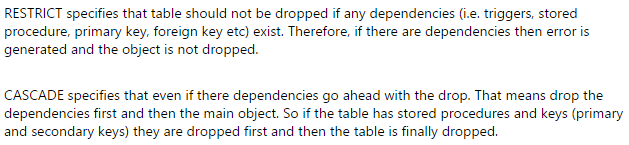
Relational Algebra Operators are critical mathematical tools used to retrieve queries by describing a sequence operations on tables or even databases(schema) involved mathematically. With relational algebra operators, a query is always composed of a number of operators, which each in turn are composed of relations as variables and return an individual abstraction as the end product. Relational Algebra operations can easily be translated into SQL commands to retrieve query results, making it a powerful tool in the hands of any Database designer, user, and administrator.

Division identifies the attribute values from a relation that are found to be paired with all of the values from another relation. Viewed another way: As multiplication is to division in arithmetic, Cartesian Product (×) is to Division in relational algebra.

Division is considered the most challenging of the eight operators Defined using three operators (π, −, and ×) and six operations Based on finding values that are not answers Not easily expressed in SQL A challenge to explain to students Often an afterthought in database texts

17.





For example, if I have two tables - Parent and Child - with a foreign key on Child that references Parent and has ON DELETE CASCADE, which records trigger a cascade and which records get deleted by the cascade? My first guess would be the Child records get deleted when Parentrecords are deleted, since Child records depend on Parent records, but the ON DELETE is ambiguous; it could mean delete the Parent record when the Child record is deleted, or it could mean delete the Child record when the Parent is deleted. So which is it?

I wish the syntax was ON PARENT DELETE, CASCADE, ON FOREIGN DELETE, CASCADE or something similar to remove the ambiguity.

# 18. Database state

**Database state** may refer to:

* Database state, in database technology the set of stored data. Entering, modifying, or deleting information changes the database state. \*Actual data stored in a particular moment in time.
  + See also [State transition system](https://en.wikipedia.org/wiki/State_transition_system) and [Finite-state machine](https://en.wikipedia.org/wiki/Finite-state_machine)models.
* A state that practices [Mass surveillance](https://en.wikipedia.org/wiki/Mass_surveillance).

|  |
| --- |
| **19. Database Security Threats and Countermeasures** |

Databases need to have level of security in order to protect the database against both malicious and accidental threats. A threat is any type of situation that will adversely affect the database system. Some factors that drive the need for security are as follows:  
  
- Theft and fraud   
- Confidentiality  
- Integrity   
- Privacy   
- Database availability  
  
Threats to database security can come from many sources. People are a substantial source of database threats. Different types of people can pose different threats. Users can gain unauthorised access through the use of another person's account. Some users may act as hackers and/or create viruses to adversely affect the performance of the system. Programmers can also pose similar threats. The Database Administrator can also cause problems by not imposing an adequate security policy.  
  
Some threats related to the hardware of the system are as follows:  
  
- Equipment failure  
- Deliberate equipment damage (e.g. arson, bombs)  
- Accidental / unforeseen equipment damage (e.g. fire, flood)  
- Power failure   
- Equipment theft  
  
Threats can exist over the communication networks that an organisation uses. Techniques such as wire tapping, cable disruption (cutting / disconnecting), and electronic interference can all be used to disrupt services or reveal private information.

**Countermeasures**  
  
Some countermeasures that can be employed are outlined below:  
  
- Access Controls (can be Discretionary or Mandatory)  
- Authorisation (granting legitimate access rights)  
- Authentication (determining whether a user is who they claim to be)   
- Backup  
- Journaling (maintaining a log file - enables easy recovery of changes)  
- Encryption (encoding data using an encryption algorithm)  
- RAID (Redundant Array of Independent Disks - protects against data loss due to disk failure)   
- Polyinstantiation (data objects that appear to have different values to users with different access rights / clearance)   
- Views (virtual relations which can limit the data viewable by certain users)