SQL language

Advanced topics

Topics

- Views
- Transactions
- Access control
- Index management
- Physical design

SQL language: advanced topics

Views

Management of views

- Introduction
- Creation and management of views in SQL
- Updating views
- Check option
- Privacy Management

The concept of view

- A view is a *"virtual"* table
  - the content (tuples) is defined by means of an SQL query on the database
    - the content of the view depends on the content of the other tables present in the database
  - the content is *not* memorized physically in the database
    - it is recalculated every time the view is used by executing the query that defines it
- A view is an object of the database
  - it can be used in queries as if it were a table

DB product suppliers

P

PId	PName	Color	Size	Store
P1	Jumper	Red	40	London
P2	Jeans	Green	48	Paris
P3	Blouse	Blue	48	Rome
P4	Blouse	Red	44	London
P5	Skirt	Blue	40	Paris
P6	Shorts	Red	42	London

S

SId	SName	#Employees	City
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

SP

SId	PId	Qty
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P3	200
S4	P4	300
S4	P5	400

Example n.1

- Definition of the view *small suppliers*
  - the suppliers that have fewer than 3 employees are considered "small suppliers"
- The view "small suppliers"
  - contains the code, name, number of employees and city of the suppliers that have fewer than 3 employees.

Example n.1: definition of the view

- Definition of the view "small suppliers"
  - contains the code, name, number of employees and city of suppliers with fewer than 3 employees

```
SELECT SId, SName, #Employees, City
FROM   S
WHERE #Employees <3
```

Query associated with the view

Example n.1: definition of the view

- Definition of the view "small suppliers"
  - it contains the code, name, number of employees and city of suppliers with fewer than 3 employees

Name of the views

```
CREATE VIEW SMALL_SUPPLIERS AS
SELECT SId, SName, #Employees, City
FROM   S
WHERE #Employees<3;
```

Example n.1: query

- View the code, name, employee number and city of "small suppliers" in London
- The query can be answered without using views

```
SELECT *
FROM   S
WHERE #Employees<3 AND
          City='London';
```

Example n.1: query

- View the code, name, employee number and city city of “small suppliers” in London
- The query can be answered using the view defined previously

```
SELECT *
FROM   SMALL_SUPPLIERS
WHERE City='London';
```

- The view SMALL_SUPPLIERS is used like a table

Rewriting the queries

- If the query refers to a view, it has to be reformulated by the DBMS before execution
- This operation is carried out automatically
  - the references to the view are substituted by its definition

Example n.1: reformulating the query

- View the code, name, employee number and city city of “small suppliers” in London

```
SELECT *
FROM   SMALL_SUPPLIERS
WHERE City='London';
```

Example n.1: reformulating the query

- View the code, name, employee number and city city of "small suppliers" in London

```
SELECT SId, SName, City, #Employees
FROM   SMALL_SUPPLIERS
WHERE City='London';
```

- Reformulate the SELECT clause
  - the attributes present in the definition of the view are made explicit

Example n.1: reformulating the query

- View the code, name, employee number and city city of "small suppliers" in London

```
SELECT SId, SName, City, #Employees
FROM   S
WHERE  #Employees<3  AND
       City='Torino';
```

- Introduction of the definition of the view
  - In the clause FROM
  - In the clause WHERE

Example n.2

- Definition of the view *number of suppliers per product*
  - The view contains the product code and the number of different suppliers providing it

Example n.2: definition of the view

- Definition of the view "number of suppliers per product"
  - The view contains the product code and the number of different suppliers providing it

```
SELECT PId, COUNT(*)
FROM   SP
GROUP BY PId
```


Query associated with the view

Example n.2: definition of the view

- Definition of the view "number of suppliers per product"
  - the view contains the product code and the number of different suppliers providing it

Name of the view

```
CREATE VIEW NUMSUPPLIERS_PER_PRODUCT
(PId, #Suppliers) AS
SELECT PId, COUNT(*)
FROM   SP
GROUP BY PId;
```

Example n.2: definition of the view

- Definition of the view "number of suppliers per product"
  - the view contains the product code and the number of different suppliers providing it

```
CREATE VIEW NUMSUPPLIERS_PER_PRODUCT
(PId, #Suppliers) AS
SELECT PId, COUNT(*)
FROM   SP
GROUP BY PId;
```

Attributes of the view

Example n.2: query

- View the code of products supplied by the greatest number of suppliers
- Without using views

Example n.2: query

- View the code of products supplied by the greatest number of suppliers
- Without using views

```
SELECT PId
FROM   SP
GROUP BY PId
HAVING COUNT(*)=(SELECT MAX(#Suppliers)
                 FROM (SELECT COUNT(*) AS #Suppliers
                       FROM   SP
                       GROUP BY PId));
```

Example n.2: query

- View the code of products supplied by the greatest number of suppliers
- Using the view NUMSUPPLIERS_PER_PRODUCT

```
SELECT PId
FROM NUMSUPPLIERS_PER_PRODUCT
WHERE #Suppliers=(SELECT MAX(#Suppliers)
                  FROM
                       NUMSUPPLIERS_PER_PRODUCT);
```

Considerations on the examples

- The use of views simplifies query formulation
- The view SMALL_SUPPLIERS masks the definition of the concept of "small suppliers"
  - it is possible to redefine the concept of "small suppliers" just by changing the definition of the view
  - it is not necessary to modify the queries that use it
- The view NUMSUPPLIERS_PER_PRODUCT substitutes the use of CTEs and derived tables

Advantages of views

- Simplification of queries
  - by breaking down a complex query into subqueries associated with the views
  - useful in the presence of repeated (complex) subqueries
- Security management
  - it is possible to introduce different privacy protection mechanisms for each user or group
    - access authorization is associated with the view
    - each user, or group, accesses the database only via views that are appropriate for the operation they are authorized to carry out

Advantages of views (II)

- Database mainteinance and evolution
  - If a database is restructured, it is possible to change the views
  - it is not necessary to re-formulate the queries written before the restructuring and present in the applications that have already been developed

SQL language: advanced topics

Creation and management of views in SQL

Creating a view

CREATE VIEW *ViewName* [*(AttributeList)*]
AS *SQLquery*;

- If the names of the attributes of a view are not specified
  - use those present in the SQL query SELECT
- Attribute names have to be specified if
  - they represent the result of an internal function
  - they represent the result of an expression
  - they are constant
  - two columns (from different tables) have the same name

Effect of cancelling tables

- DROP VIEW *ViewName;*

- Cancelling a table that a view refers to can have various effects
  - automatic elimination of the associated views
  - automatic invalidation of the associated views
  - prohibition to execute the operation of cancelling the table
- the effect depends on the specific DBMS

Modifying the definition of a view

ALTER VIEW *ViewName* [*(AttributieList)*]
AS *SQLquery*;

Updating views

- It is possible to update the data in a view *only* for some types of views
- Standard SQL-92
  - views in which a single row of each table corresponds to a single row of the view can be updated
    - univocal correspondence between the tuple of the view and the tuple of the table on which it is defined
    - it is possibile to propagate without ambiguity the changes made to the view to each table on which it is defined

Updating views

- *It is not possible to update* a view for which the outer block of its defining query:
  - does not include the primary key of the table on which it is defined
  - contains joins that represent correspondences to one-to-many or many-to-many
  - contains aggregate functions
  - contains DISTINCT

Example n.1

- View SUPPLIER_CITY

```
CREATE VIEW SUPPLIER_CITY AS
SELECT SId, City
FROM   S;
```

Example n.1: insertion

- Insertion in SUPPLIER_CITY of
  ('S10', 'Rome')

  - corresponds to the insertion in S of
    ('S10',NULL,NULL,'Rome')
  - the attributes SName, #Employees have to admit the value NULL

Example n.1: cancellation

- Cancellation of SUPPLIER_CITY of
  ('S1', 'London')
  - cancellation from S of
    ('S1', 'Smith',20,'London')
  - identification of the tuple to cancel is permitted by the primary key

Example n.1: change

- change to SUPPLIER_CITY of
  ('S1', 'London') in ('S1', 'Milan')

  - change in S of
    ('S1', 'Smith',20,'London') in ('S1', 'Smith',20,'Milan')
  - identification of the tuple to change is permitted by the primary key

Example n.1: updating

- The view SUPPLIER_CITY *can be updated*
  - each tuple of the view corresponds to a single tuple of table S
  - the changes carried out on the view can be propagated to the table on which it is defined

Example n.2

- View NUMEMPLOYEE_CITY

```
CREATE VIEW NUMEMPLOYEE_CITY AS
SELECT DISTINCT #Employees, City
FROM   S;
```

Example n.2: insertion

- Insertion in NUMEMPLOYEE_CITY of
  (40, 'Rome')

  - it is impossible to insert in S
    (NULL,NULL,40,'Rome')
    - the value of the primary key is missing

Example n.2: cancellation

- Cancellation from NUMEMPLOYEE_CITY of (20, 'London')
  - several tuples are associated with the pair (20, 'London')
    - Which tuple has to be cancelled from S?

Example n.2: change

- Change in NUMEMPLOYEE_CITY of
  (20, 'London') in (30, 'Rome')

  - Several tuples are associated with the pair (20, 'London')
    - Which tuple has to be changed in S?

Example n.2: updating

- The view NUMEMPLOYEE_CITY *cannot be updated*
  - the primary key of table S is not present in the view
    - the insertion of new tuples in the view cannot be propagated to S
  - some tuples of the view correspond to several tuples in the table S
    - the association between the tuples in the view and the tuples in the table is ambiguous
    - it is not possible to propagate the changes carried out on the tuples of the view to the tuples of the table on which it is defined

Updating the views

- Some non-updatable views become updatable by changing the SQL expression associated with the view
  - it may be necessary to change the information content of the view

Example n.3: non-updatable view

```
CREATE VIEW SUPPLIER_LONDON AS
SELECT *
FROM S
WHERE City='London';
```

- The view is non-updatable
  - it does not explicitly select the primary key of table S
- It is sufficient to replace the symbol "*" with the name of the attributes

Example n.3: changed view

```
CREATE VIEW SUPPLIER_LONDON AS
SELECT SId, SName, #Employees, City
FROM S
WHERE City='London';
```

- The view is updatable

Example n.4: non-updatable view

```
CREATE VIEW BEST_SUPPLIER (SId, SName) AS
SELECT DISTINCT SId, SName
FROM   S, SP
WHERE S.SId=SP.SId AND
          Qty>100;
```

- The view is non-updatable
  - a join is present
  - the keyword DISTINCT is present

Example n.4: changed view

```
CREATE VIEW BEST_SUPPLIER (SId, SName) AS
SELECT SId, SName
FROM   S
WHERE SId IN (SELECT SId
                    FROM   SP
                    WHERE Qty>100);
```

- The view is updatable
  - the join was realised using IN
  - the keyword DISTINCT is no longer necessary

Example n.5: non-updatable view

```
CREATE VIEW TOP_SUPPLIER (SId, SName, TotQty) AS
SELECT SId, SName, SUM(Qty)
FROM   S, SP
WHERE S.SId=SP.SId
GROUP BY SId, SName
HAVING SUM(Qty)>500;
```

- The view is non-updatable
  - an aggregate function is present
  - a join is present

Example n.5: changed view

```
CREATE VIEW TOP_SUPPLIER (SId, SName) AS
SELECT SId, SName
FROM   S
WHERE SId IN (SELECT SId FROM SP
                    GROUP BY SId
                    HAVING SUM(Qty)>500);
```

- The view is updatable
  - The "group by" has been moved into the nested query
- The *information content has changed*

SQL language: advanced topics

Transactions

Example of application

- Banking operations
  - cash withdrawal from a current account using a cash card
  - depositing cash on a current account

Cash withdrawal

- Operations
  - specify the amount required
  - check availability
  - memorize transaction
  - update balance
  - enable withdrawal of the amount required
- All the operations have to be carried out correctly, otherwise the cash cannot be withdrawn

Transactions

- It is necessary when several users can simultaneously access the data
- It provides efficient mechanisms for
  - managing competing access to data
  - recovery after a malfunction

Transactions

- A transaction is a sequence of operations that
  - represents an elementary unit of work
  - can end in success or failure
    - in the case of success, the result of the operations has to be permanent in the database
    - in the case of failure, the database has to return to the original state before the transaction was initiated

Transactional system

- A system that makes a mechanism available for the definition and execution of transactions is called a *transactional system*
- The DBMS contain architecture blocks that offer transaction management services

Beginning a transaction

- To define the beginning of a transaction, the SQL language uses the instruction
  - START TRANSACTION
- Usually the instruction to begin a transaction is omitted
  - the beginning is implicit for
    - the first SQL instruction of the programme that accesses the database
    - the first SQL instruction following the instruction ending the previous transaction

Ending a transaction

- The SQL language has instructions for defining the end of a transaction
  - Transaction successful
    - COMMIT [WORK]
    - the action associated with the instruction is called *commit*
  - Transaction failed
    - ROLLBACK [WORK]
    - the action associated with the instruction is called *abort*

Commit

- Action executed when a transaction ends with success
- The database is in a new (final) correct state
- The changes to the data executed by the transaction become
  - permanent
  - visibile to other users

Rollback

- Action executed when a transaction ends because of an error
  - for example, an error in application
- All the operations modifying the data executed during the transaction are "cancelled"
- The database returns to the state prior to the beginning of the transaction
  - the data is visible again to the other users

Example

- Transfer the sum of 100
  - from current account number IT92X0108201004300000322229
  - to current account number IT32L0201601002410000278976

```
START TRANSACTION;
UPDATE Account
    SET Balance= Balance - 100
    WHERE IBAN='IT92X0108201004300000322229';
UPDATE Account
    SET Balance = Balance + 100
    WHERE IBAN= 'IT32L0201601002410000278976';
COMMIT;
```

Transaction properties

- The principal properties of transactions are
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- They are summarized by the English acronym *ACID*

Atomicity

- A transaction is an indivisible unit (atom) of work
  - all the operations contained in the transaction have to be executed
  - or none of the operations contained in the transaction have to be executed
    - the transaction has no effect on the database
- The database cannot remain in an intermediate state arrived at during the processing of a transaction

Consistency

- The execution of a transaction has to take the database
  - from an initial state of consistency (correct)
  - to a final state of consistency
- Correctness is verified by integrity constraints defined on the database
- When there is a violation of the integrity constraint the system intervenes
  - to abort the transaction
  - or to modify the state of the database by eliminating the violation of the constraint

Isolation

- The execution of a transaction is independent from the simultaneous execution of other transactions
- The effects of a transaction are not visible by other transactions until the transaction is terminated
  - the visibility of unstable intermediate states is avoided
    - an intermediate state can be aborted by a subsequent rollback
    - in the case of rollback, it is necessary to rollback the other transactions that have observed the intermediate state (domino effect)

Durability

- The effect of a transaction that has executed a commit is memorized permanently
  - the changes to the data carried out by a transaction ending successfully are permanent after a commit
- It guarantees the reliability of the operations of data modification
  - the DBMS provides mechanisms for recovery to the correct state of the database after a malfunction has occurred

SQL language: advanced topics

Access control

Data security

- Protection of data from
  - unauthorized readers
  - alteration or destruction
- The DBMS provides protection tools which are defined by the database administrator (DBA)

Data security

- Security control verifies that users are authorized to execute the operations they request
- Security is guaranteed through a set of constraints
  - specificied by the DBA in an appropriate language
  - memorized in the data dictionary system

Access control

Resources and privileges

Resources

- Any component of the database scheme is a resource
  - table
  - view
  - attribute in a table or view
  - domain
  - procedure
  - ...
- Resources are protected by the definition of *access privileges*

Access privileges

- Describe access rights to system resources
- SQL provides very flexible access control mechanisms for specifying
  - the resources users can access
  - the resources that have to remain private

Privileges: characteristics

- Each privilege is characterized by the following information
  - the resource it refers to
  - the type of privilege
    - describes the action allowed on the resource
  - the user granting the privilege
  - the user receiving the privilege
  - the faculty to transmit the privilege to other users

Types of privilege (1/2)

- INSERT
  - enables the insertion of a new object in the resource
  - valid for tables and views
- UPDATE
  - enables updating the value of an object
  - valid for tables, views and attributes
- DELETE
  - enables removal of objects from the resource
  - valid for tables and views

Types of privilege (2/2)

- SELECT
  - enables using the resource in a query
  - valid for tables and views
- REFERENCES
  - enables referring to a resource in the definition of a table scheme
  - can be associated with tables and attributes
- USAGE
  - enables use of the resource (e.g. a new type of data) in the definition of new schemes

Resource creator privileges

- When a resource is created, the system grants all privileges over that resource to the user that created it
- Only the resource creator has the privilege to eliminate a resource (DROP) and modify a scheme (ALTER)
  - the privilege to eliminate and modify a resource cannot be granted to any other user

System administrator privileges

- The system administrator (user system) possesses all privileges over all the resources

Management of privileges in SQL

- Privileges are granted or revoked using SQL instructions
  - GRANT
    - grants privileges over a resource to one or more users
  - REVOKE
    - revokes privileges granted to one or more users

GRANT

GRANT *PrivilegeList* ON *ResourceName* TO *UserList*
[WITH GRANT OPTION]

- *PrivilegeList*
  - specifies the list of privileges
  - ALL PRIVILEGES
    - Keyword for identifying all privileges
- *ResourceName*
  - specifies the resource for which the privilege is granted
- *UserList*
  - Specifies the users who are granted the privilege

GRANT

GRANT *PrivilegeList* ON *ResourceName* TO *UserList*
[WITH GRANT OPTION]

- WITH GRANT OPTION
  - faculty to transfer the privilege to other users

Example n. 1

```
GRANT ALL PRIVILEGES ON P
TO Smith, Singh
```

- Users Smith and Singh are granted all privileges for table P

Example n. 2

```
GRANT SELECT ON S TO Red
                WITH GRANT OPTION
```

- User Red is granted the privilege to SELECT in table S
- User Red has the faculty to grant the privilege to other users

REVOKE

REVOKE *PrivilegeList* ON *ResourceName* FROM *UserList* [RESTRICT|CASCADE]

- The command REVOKE can remove
  - all the privileges that have been granted
  - a subset of privileges granted

Example n. 1

```
REVOKE UPDATE ON P FROM White
```

- User White's privilege to UPDATE table P is revoked

REVOKE

REVOKE *PrivilegeList* ON *ResourceName* FROM *UserList* [RESTRICT|CASCADE]

- RESTRICT
  - the command must not be executed if revoking the user's privileges entails revoking other privileges
    - Example: the user has received the privileges with the GRANT OPTION and has propagated the privileges to other users
  - default value

Example n. 1

```
REVOKE UPDATE ON P FROM White
```

- User White's privilege to UPDATE table P is revoked
  - the command is not executed if it entails revoking the privilege of other users

REVOKE

REVOKE *PrivilegeList* ON *ResourceName* FROM *UserList* [RESTRICT|CASCADE]

- CASCADE
  - revokes also all the privileges which have been propagated
    - generates a chain reaction
  - for each privilege revoked
    - all granted privileges are revoked in a cascade
    - all database elements which have been created exploiting these privileges are removed

Example n. 2

REVOKE SELECT ON S FROM Red CASCADE

- User Red's privilege to SELECT table S is revoked
- User Red had received the privilege through GRANT OPTION
  - if Red has propagated the privilege to other users, the privilege is revoked in cascade
  - if Red has created a view using the SELECT privilege, the view is removed

Concept of role (1/2)

- The role is an access profile
  - Defined by its set of privileges
- Each user has a defined role
  - it enjoys the privileges associated with that role

Concept of role (2/2)

- Advantages
  - access control is more flexible
    - a user can have different roles at different times
  - it simplifies administration
    - an access profile need not be defined at the moment of its activation
    - it is easy to define new user profiles

Roles in SQL-3

- Definition of a role

  CREATE ROLE *RoleName*

- Definition of role privileges and user roles
  - instruction GRANT
- A user can have different roles at different times
  - dynamic association of a role with a user

SET ROLE *RoleName*

Physical design

Physical data organization

- In a relational DBMS the data are represented as collections of records memorized in one or more files
  - the physical organization of the data in a file influences the time required to access the information
  - each physical data organization makes some operations efficient and others cumbersome
- There is no physical data organization that is efficient for any type of data reading and writing

Indexes

- *Indexes* are the accessory physical structures provided by the relational DBMS to improve the efficiency of data access operations
  - indexes are realized using different types of physical structures
    - trees
    - hash tables
- The instructions for managing the indexes are not part of the standard SQL

Tree structure

- Efficient associative data access, based on the value of a key
  - the key can be composed of one or more attributes
- A *tree structure* enables access to the set of physical locations of the records corresponding to the selected value of the key field
  - The physical location of a record indicates the physical position of the record in the file in the secondary memory
- Examples: B-tree, B^+-tree

Calculated access structure

- Efficient associative data access, based on the value of a key field
  - the key can be composed of one or more attributes
- It requires a *calculation algorithm* to locate the physical block of the file containing the records corresponding to the value of the key field
- It does not require a specific record system in the secondary memory
- Example: hash structure

Index definition in SQL

- SQL language provides the following instructions for defining indexes
  - to create an index
    - CREATE INDEX
  - to cancel an index
    - DROP INDEX
- The instructions for the management of indexes are not part of the standard SQL

To create an index

CREATE INDEX *IndexName*
ON T*ableName (AttributeList)*

- Create an index
  - with the name *IndexName*
  - on the table *TableName*
  - defining its attributes in *AttributeList*

To create an index

CREATE INDEX *IndexName*
ON T*ableName (AttributeList)*

- The order in which the attributes appear in *AttributeList is important*
  - the order of the index keys is
    - first on the basis of the first attribute in *AttributeList*
    - equal in value to the first attribute on the values of the second attribute
    - and so on, in order, until the last attribute

Example database

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MonthlySalary
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00

Example n.1

- Creation of an index on the attribute Residence of the table EMPLOYEE

```
CREATE INDEX ResidenceIndex
ON EMPLOYEE (Residence)
```

Example n.2

- Creation of an index on a combination of attributes Surname and Name of the table EMPLOYEE

```
CREATE INDEX SurnameNameIndex
ON EMPLOYEE (Surname,Name)
```

- The index is jointly defined on the two attributes
- The index keys are ordered
  - first on the basis of the value of the attribute Surname
  - of equal value to the attribute Surname, on the value of the attribute Name

To delete an index

DROP INDEX *IndexName*

- Eliminate the index with the name *IndexName*
- This command is used when
  - the index is no longer utilized
  - the improvement in performance is insufficient
    - reduced reduction in response time for the queries
    - slowing down of updates due to index maintenance

Physical design

- This is the final phase of database design
  - it requires choosing the DBMS to be utilized
  - it is linked to the characteristics of the product chosen

Physical design: entry data

- Logical scheme of the database
- Characteristics of the chosen DBMS
  - physically available options
    - physical memory structures
    - indexes
- Data volumes
  - cardinality of tables
  - cardinality and distribution of the attribute values domain

Physical design: entry data

- Estimate of application load
  - most important queries and their frequency
  - most important updating operations and their frequency
  - response time requirements for important queries/updates

Physical design: result

- Physical scheme of the database
  - physical organization of tables
  - indexes
- Memorization and functioning parameters
  - Initial file sizes, expansion possibilities, free space at outset, ...

Procedure

- Physical design is carried out empirically, using a trial and error approach
  - there are no reference methodologies

Procedure

- Characterization of application load
  - for each important query it is necessary to define
    - access relationships
    - attributes to be viewed
    - attributes involved in selections/joins
    - degree of selectivity of selection conditions
  - for each important update it is necessary to define
    - type of update
      - Insertion, cancellation, modification
    - relation to any attributes involved
    - degree of selectivity of selection conditions

Procedure

- Choices to be made
  - physical structuring of the files containing the tables
    - ordered, unordered
  - choice of attributes to index
    - piloted by estimating applicative load and data volume
  - definition of type for each index
    - e.g. hash or B-tree
  - any variations of the scheme
    - horizontal partitioning in the secondary memory
    - denormalization of tables
      - utilized in data warehouses

Tuning

- If the result is not satisfactory
  - *Tuning*, adding and removing indexes
- This is a procedure guided by the availability of tools that enable
  - verification of the execution plan adopted by the chosen DBMS
    - the execution plan defines the sequence of activities carried out by the DBMS to execute a query
      - data access methods
      - join methods
  - assess the execution cost of various alternatives