Modèle relationnel et théorème de Codd

Bases de données

Amélie Gheerbrant
IRIF, Université Paris Diderot
amelie@irif.fr

(dont quelques éléments empruntés à
Paolo Guagliardo, Björn Þór Jónsson,
Phokion Kolaitis, Pierre Senellart, Moshe Vardi…)

Systèmes de gestion de bases de données : rappels

- De nombreuses applications (logiciels, sites Web, etc.) ont besoin de gérer des données :
  - structurer les données utiles à l'application
  - permettre de les conserver de manière persistante (données conservées même quand l'application ne tourne pas)
  - rechercher efficacement des informations dans de grands volumes de données
  - mettre à jour des données sans violer certaines contraintes de structure
  - permettre l'accès et la mise à jour de données à des utilisateurs multiples, parfois de manière concurrente
- Souvent, souhaitable d'accéder aux mêmes données depuis plusieurs applications distinctes, depuis des ordinateurs distincts

Exemple : système d'information d'un hôtel

- Accès depuis un logiciel maison (réception), un site Web (clients), un logiciel de comptabilité. Besoins :
  - données structurées représentant les chambres, les clients, les réservations, les tarifs, etc.
  - pas de données perdues quand ces applications ne sont pas utilisées, ou quand une panne de courant générale survient
  - retrouver quasi instantanément quelles chambres sont réservées, et par qui, un jour donné, dans un historique de plusieurs années de réservation
  - facilement ajouter une réservation en s'assurant de ne pas réserver deux fois la même chambre le même jour
  - le client, le réceptionniste, le comptable, ne doivent pas avoir la même vue des données (confidentalité, simplicité, etc.) ; différents clients ne peuvent réserver la même chambre au même moment

Implémentative naïve (sans SGBD)

- Implémentation dans un langage de programmation classique (C++, Java, Python, etc.) de structures de données permettant de représenter l'ensemble des données utiles
- Définition de formats de fichiers ad hoc pour stocker les données sur disques, avec synchronisation régulière et mécanisme de récupération en cas de panne
- Stockage en mémoire des données avec l'application, avec structures de données (arbres binaires, tables de hachage) et algorithmes (recherche, tri, agrégation, parcours de graphe…) permettant de retrouver efficacement l'information
- Fonctions de mise à jour des données, avec du code vérifiant au passage qu'aucune contrainte n'est violée
- Définition au sein du logiciel de droits d'utilisateurs différents, d'un mécanisme d'authentification ; utilisation de threads pour répondre en même temps à différentes requêtes, verrous/sémaphores sur les fonctions critiques de manipulation de données
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de compatibilité, etc.

Implémentative naïve (sans BD)

- Implémentation dans un langage de programmation classique (C++, Java, Python, etc.) de structures de données permettant de représenter l'ensemble des données utiles
- Définition de formats de fichiers ad hoc pour stocker les données sur disques, avec synchronisa ation en cas de panne
- Stockage en mé ec structures de données (arbres binaires, erche, tri, agrégation, parcours de gra ment l'information
- Fonctions de mi ifiant au passage qu'aucune contr
- Définition au se rents, d'un mécanisme d'authentificatio , utilisation de threads pour répondre en même temps à différentes requêtes, verrous/sémaphores sur les fonctions critiques de manipulation de données
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de compatibilité, etc.

Beaucoup de travail !!

Demande un programmeur qui maîtrise la POO, la sérialisation, la reprise sur panne, les structures de données, l'algorithmique, la gestion de contraintes d'intégrité, la gestion de rôles, la programmation parallèle, la gestion de concurrence, la programmation réseau…

Implémentative naïve (sans BD)

- Implémentation dans un langage de programmation classique (C++, Java, Python, etc.) de structures de données permettant de représenter l'ensemble des données utiles
- Définition de formats de fichiers ad hoc pour stocker les données sur disques, avec synchronisa ration en cas de panne
- Stockage en mé ec structures de données (arbres binaires, erche, tri, agrégation, parcours de gra ment l'information
- Fonctions de mi ifiant au passage qu'aucune contr
- Définition au se erents, d'un mécanisme d'authentificatio en même temps à différentes requ ns critiques de manipulation de
- Définition et implémentation d'un protocole de communication réseau pour connecter ce composant logiciel à un serveur Web, un logiciel Windows, une suite logicielle de compatibilité, etc.

Beaucoup de travail !!

Demande un programmeur qui maîtrise la POO, la sérialisation, la reprise sur panne, les structures de données, l'algorithmique, la gestion de contraintes d'intégrité, la gestion de rôles, la programmation parallèle, la gestion de concurrence, la programmation réseau… A refaire pour chaque nouvelle application ayant à gérer des données !

Fonctionnalités d'un SGBD

Indépendance physique : l'utilisateur n'a pas besoin de savoir comment les données sont stockées (dans un fichier, sur une partition brute, sur un système de fichiers distribués) ; le stockage peut être modifié sans impacter l'accès aux données

Indépendance logique : il est possible de fournir à l'utilisateur une vue partielle des données, en fonction de ses droits et besoins en termes d'accès

Accès aisé aux données : utilisation d'un langage déclaratif qui décrit l'intention de l'utilisateur plutôt que la manière dont ce sera implémenté

Optimisation des requêtes : les requêtes sont automatiquement optimisées pour être implémentées le plus efficacement possible sur la base de données

Intégrité logique : le SGBD impose des contraintes sur la structure des données ; toute modification violant ces contraintes est refusée

Intégrité physique : la base reste dans un état cohérent et les données sont préservées de manière durable, même en cas de panne

Partage des données : les données sont accessibles à des utilisateurs multiples, de manière concurrente, sans violation de leur intégrité physique et logique

Normalisation : l'utilisation d'un SGBD est standardisée, de sorte qu'il est possible de remplacer un SGBD par un autre sans (trop) changer le code de l'application

Variété des SGBD

- Dizaine de SGBD existant utilisés de manière non négligeable
- Tous les SGBD ne fournissent pas toutes ces fonctionnalités
- Mais les SGBD relationnels (comme PostgreSQL) les fournissent tous
- Les SGBD se distinguent les uns des autres par le modèle de données utilisé, les compromis réalisés en terme de performances et de fonctionnalités, la facilité d'utilisation, les volumes de données gérés, l'architecture interne, etc.

Grands types de SGBD

Relationnels (SGBDR) : tables, requêtes complexes (SQL), fonctionnalités riches

XML : arbres, requêtes complexes (XQuery), fonctionnalités similaires aux SGBDR

Graphes/Triplets : données graphes, requêtes complexes exprimant des parcours de graphe

Objets : modèle de données très complexe, inspiré de la POO

Documents : le SGBD impose des contraintes sur la structure des données ; toute modification violant ces contraintes est refusée

Clef-Valeur : modèle de données très basique, accent mis sur la performance

Orientés Colonnes : modèle de données entre celui des clef-valeur et des SGBDR ; accent mis sur le parcours ou l'agrégation efficace de colonnes

Grands types de SGBD

Relationnels (SGBDR) : tables, requêtes complexes (SQL), fonctionnalités riches

NoSQL = « not only SQL »

XML : arbres, requêtes complexes (XQuery), fonctionnalités similaires aux SGBDR

Graphes/Triplets : données graphes, requêtes complexes exprimant des parcours de graphe

Objets : modèle de données très complexe, inspiré de la POO

Documents : le SGBD impose des contraintes sur la structure des données ; toute modification violant ces contraintes est refusée

Clef-Valeur : modèle de données très basique, accent mis sur la performance

Orientés Colonnes : modèle de données entre celui des clef-valeur et des SGBDR ; accent mis sur le parcours ou l'agrégation efficace de colonnes

SGBD relationnels classiques

- Basés sur le modèle relationnel : décomposition des données en relations, ou tables
- Un langage de requêtes standard : SQL
- Données stockées sur disque
- Relations (tables) stockées ligne par ligne
- Système centralisé, avec possibilités limitées de distribution

Force des SGBD relationnels classiques

- Indépendance entre :
  - modèle de données et structures de stockage
  - requêtes déclaratives et exécution
- Requêtes complexes
- Optimisation très fine des requêtes, index permettant un accès rapide aux données
- Logiciels mûrs, stables, efficaces, riches en fonctionnalités et en interfaces
- Contraintes d'intégrité permettant d'assurer des invariants sur les données
- Gestion efficace de grands volumes de données (gigaoctet, voire téraoctet)
- Transactions (ensembles d'opérations élémentaires garantissant la gestion de la concurrence, l'isolation entre utilisateurs, la reprise sur panne

Propriétés ACID

Les transactions des SGBD relationnels classiques respectent les propriétés ACID :

Atomicité : l'ensemble des opérations d'une transaction est soit exécuté en bloc, soit annulé en bloc

Cohérence : les transactions respectent les contraintes d'intégrité de la base

Isolation : deux exécutions concurrentes de transactions résultent en un état équivalent à l'exécution sérielle des transactions

Durabilité : une fois une transaction confirmée, les données correspondantes restent durablement dans la base, même en cas de panne

(Transactions : au programme du cours de BD avancées de M1)

Faiblesse des SGBD relationnels classiques

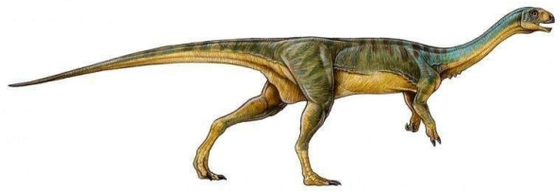
- Incapable de gérer de très grands volumes de données (de l'ordre du péta-octet)
- Impossible de gérer des débits extrêmes (plus que quelques milliers de requêtes par seconde)
- Le modèle relationnel est parfois peu adapté au stockage et à l'interrogation de certains types de données (données hiérarchiques, faiblement structurées, semi-structurées)
- Les propriétés ACID entraînent de sérieux surcoûts en latence, accès disques, temps CPU (verrous, journalisation, etc.)
- Performances limitées par les accès disques

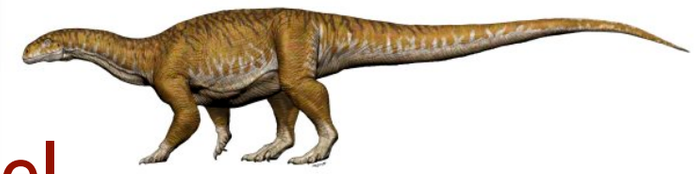
NoSQL

- No SQL ou Not Only SQL
- SGBD avec d'autres compromis que ceux faits par les systèmes classiques
- Écosystème très varié
- Fonctionnalités recherchées : modèle de données différent, passage à l'échelle, performances extrêmes
- Fonctionnalités abandonnées : ACID, (parfois) requêtes complexes

New SQL

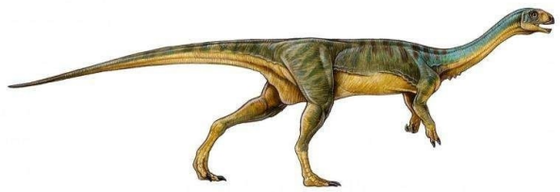
- Certaines applications nécessitent :
- un langage de requêtes riches (SQL)
- une conformité aux propriétés ACID
- mais des performances supérieures à celles des SGBD classiques
- Solutions possibles :
- Se débarrasser des goulots d'étranglement classiques desSGBD : verrous, journalisation, gestion des caches
- Bases de données en mémoire vive , avec copie sur disque asynchrone
- Une gestion de concurrence sans verrou (MVCC)
- Une architecture distribuée sans partage d'information (shared nothing) et avec équilibrage de charge transparent



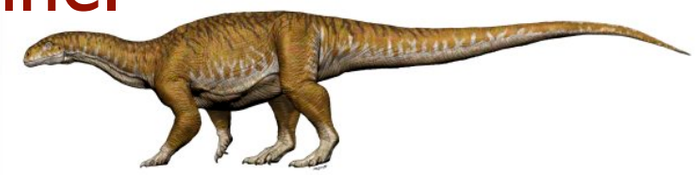


La préhistoire du modèle relationnel

- Le modèle relationnel a donné lieu à une industrie de plusieurs millards de $$$
- Mais comment en est-on arrivés là ?



La préhistoire du modèle relationnel



- D'abord, il y a eu les cartes perforées (1910) :

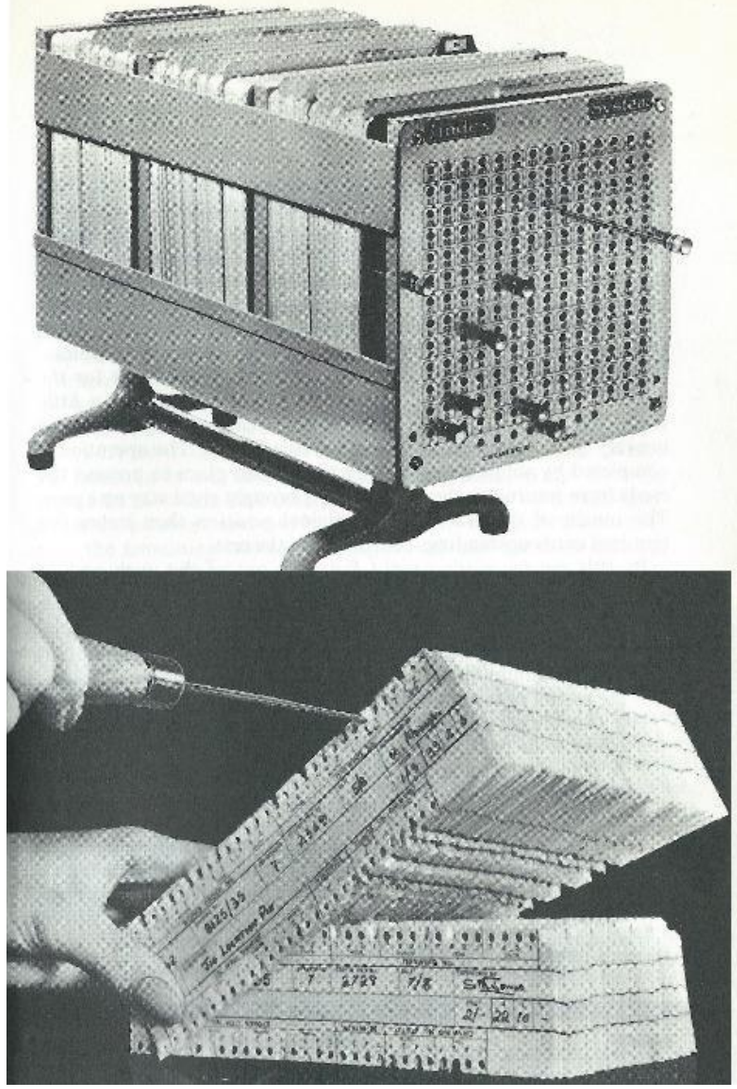
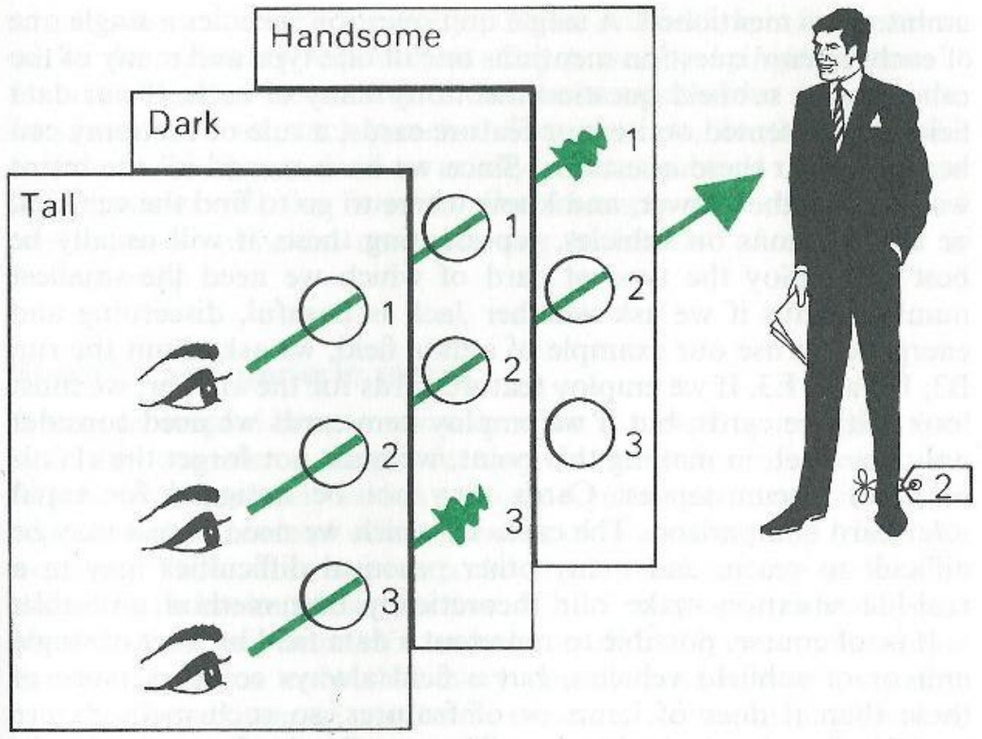
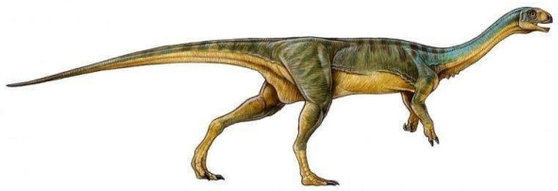
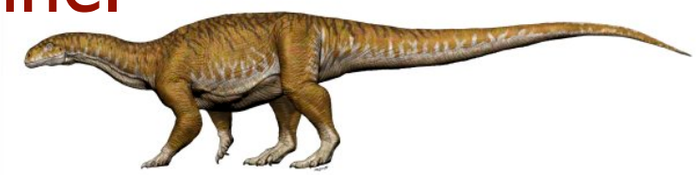


Figure 9. Answering a feature question – a search by stacking punched feature cards. Item two is shown as possessing all three qualifications.





La préhistoire du modèle relationnel



- Puis, vinrent les systèmes de fichiers (ERMA Electronic Recording Machine Accounting, 1955) :
  - les données sont rangées dans des fichiers, non connectés entre eux

Flat-File Database Model

Fruit	Vegetables
Bananas	Broccoli
Grapes	Kale
Lemons	Onions
Limes	

The flat-file database model has different data in separate files.

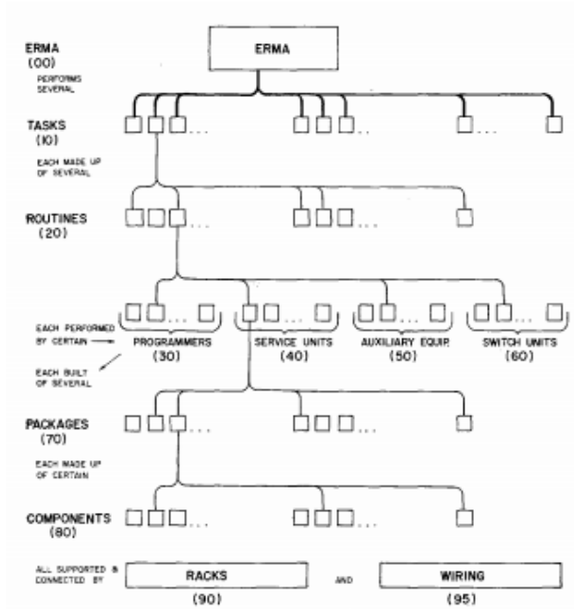


Fig. 2. The ERMA Mark I hierarchical structure

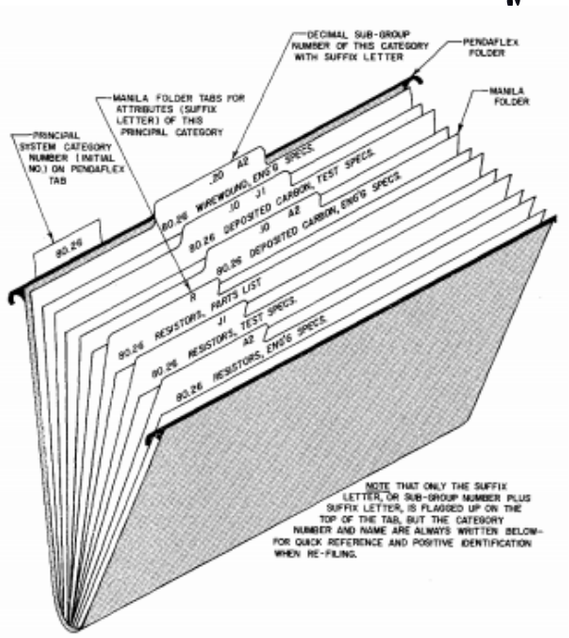
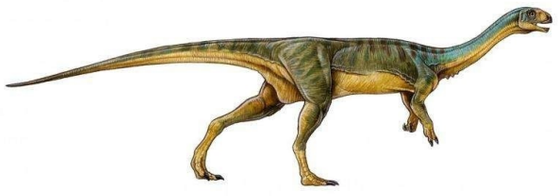
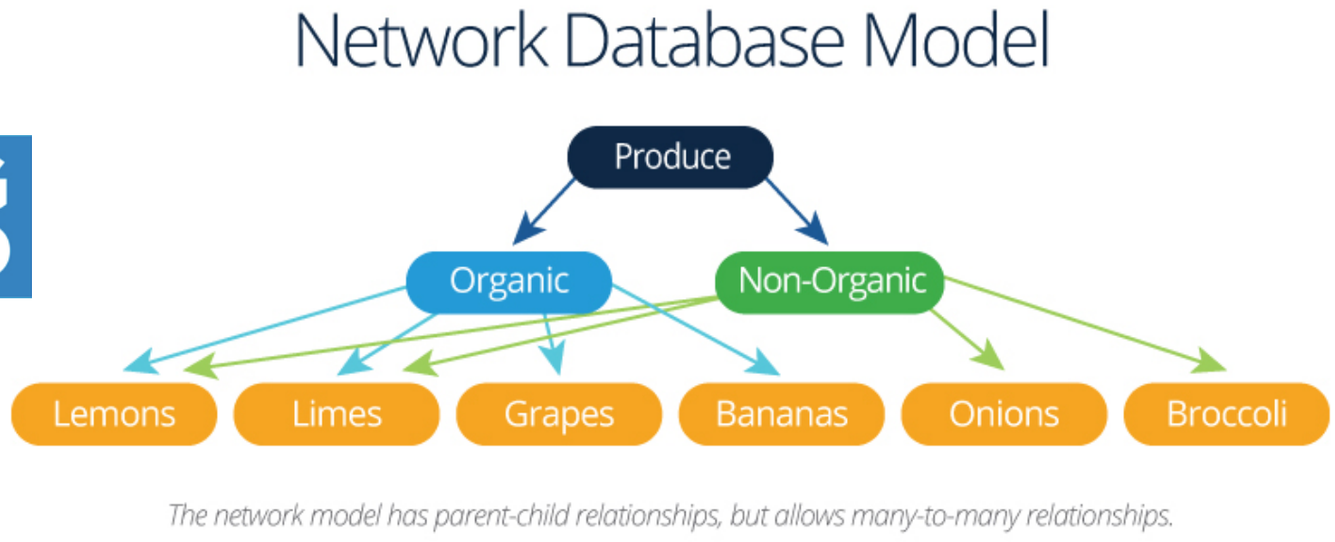


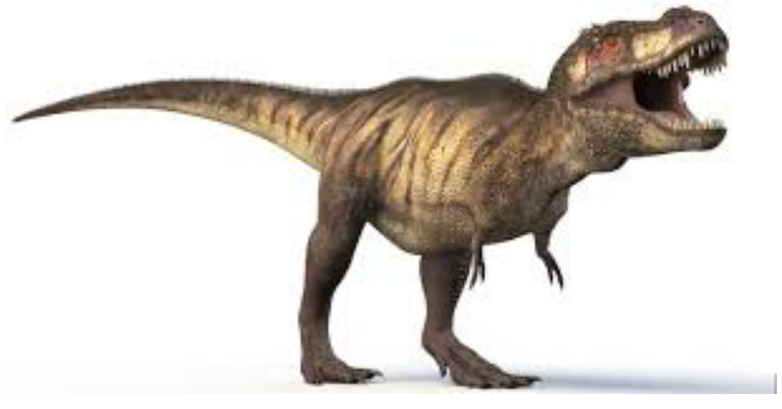
Fig. 3. Records storage and arrangement

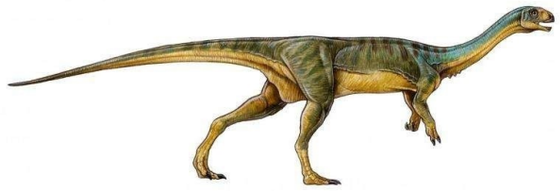


La préhistoire du modèle relationnel : les premiers SGBD

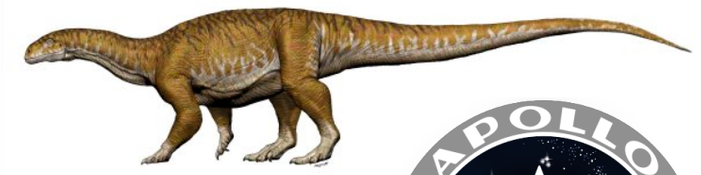
- Le modèle réseau
  - similaire au modèle hiérarchique, mais les associations n-n sont admises
- CoDaSyl DBTG (Data Base Task Group), système IDS (Integrated Data Store)
- SGBD développé chez General Electrics 1964 par Bachman (Turing Award en 1973)



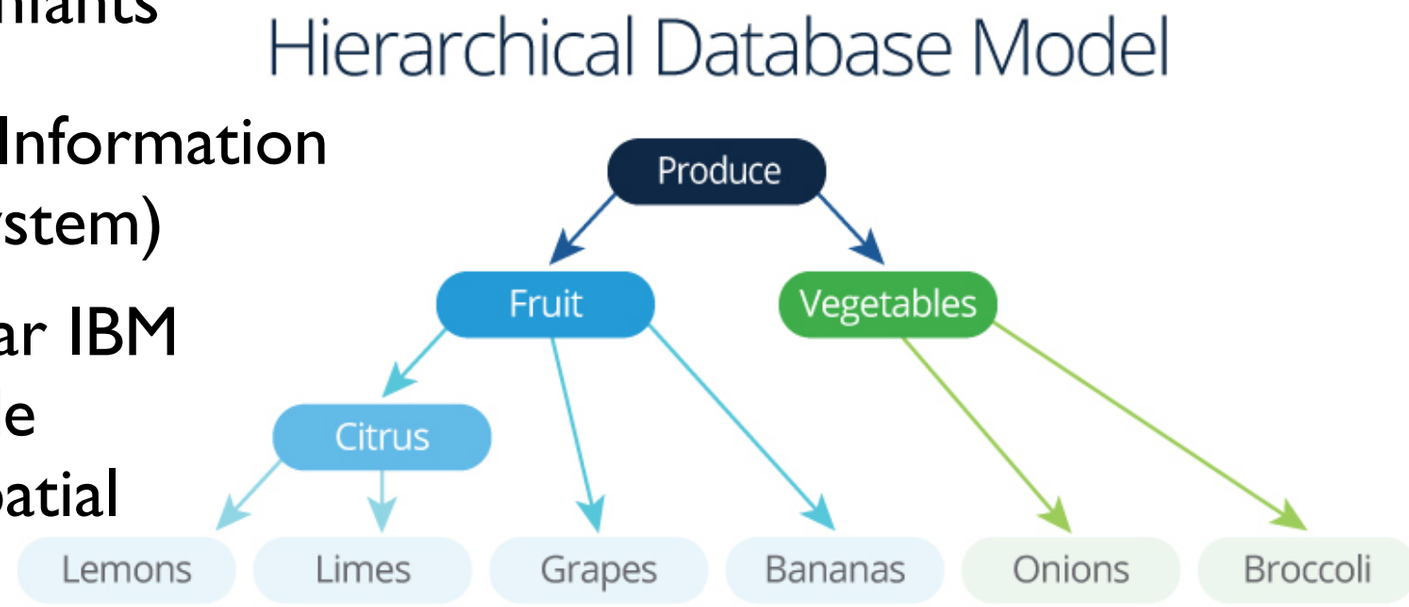




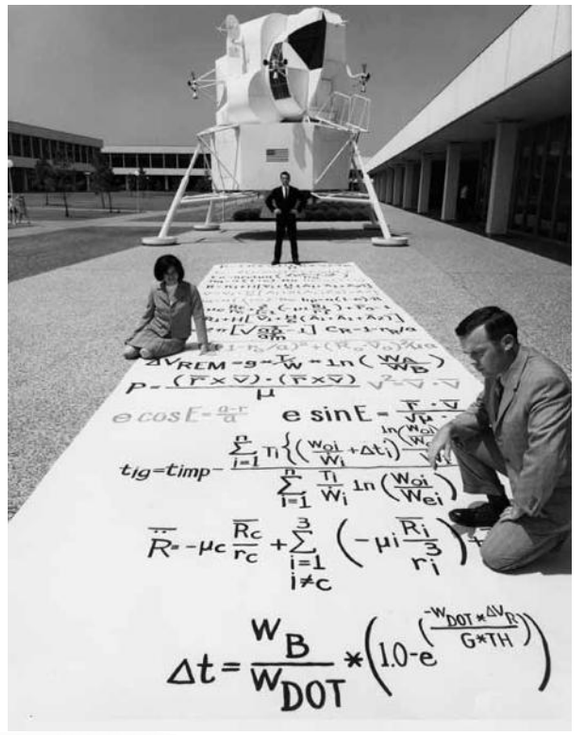
La préhistoire du modèle relationnel : les premiers SGBD

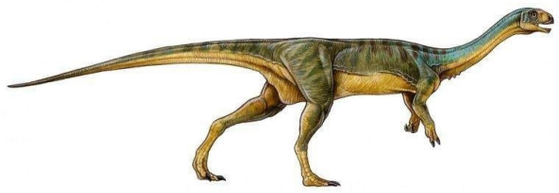


- Le modèle hiérarchique
  - chaque élément a un unique enregistrement parent, mais peut avoir plusieurs enregistrements enfant, il ne peut être en lien qu'avec son parent ou ses enfants



- Système IMS (Information Managment System)
- SGBD lancé par IBM en1966, pour le programme spatial Apollo





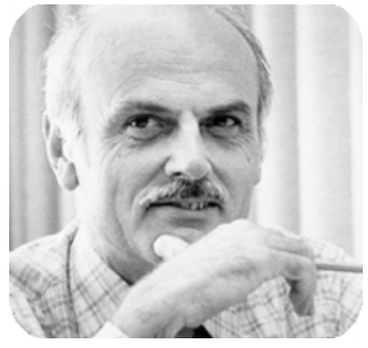
La préhistoire du modèle relationnel : avant les 70’

- Modèles hiérarchique et réseau, avénement des SGBD modernes, avancées considérables par rapport aux systèmes de fichiers :
  - l’organisation des données est dévolue à un sous-système spécialisé
  - très bien adapté aux données naturellement représentées sous forme d’arbres ou de graphe… mais pas aux autres
- Défaut principal : manque d’indépendance des données :
  - langages de requêtes compliqués
  - séparation données / navigation très difficile
  - tout changement dans la structure des données peut mener à l’obsolescence des programmes

```
SQL: select name from student where dept = “EECS”

DBTG:
student.dept = “EECS”;
find any student using dept;
while DB-status = 0 do
   begin
      get student;
      print (student.name);
      find duplicate student using dept;
   end
```

L'histoire du modèle relationnel



- 1970 « A relational Model of Data for Large Shared Data Banks »
- 1972 « Relational Completeness of Data Base Sublanguages » (Codd, IBM)

This was as I say a revelation for me because Codd had a bunch of queries that were fairly complicated queries and since I'd been studying CODASYL, I could imagine how those queries would have been represented in CODASYL by programs that were five pages long that would navigate through this labyrinth of pointers and stuff. Codd would sort of write them down as one-liners. These would be queries like, "Find the employees who earn more than their managers." He just whacked them out and you could sort of read them, and they weren't complicated at all, and I said, "Wow." This was kind of a conversion experience for me, that I understood what the relational thing was about after that.
(Chamberlin, The 1995 SQL Reunion: People, Projects, and Politics)

Codd critique les modèles existants et propose le modèle relationnel.

D'abord peu d'écho : il n'y a pas de système, seulement des requêtes très simples… Et puis :

- Ingres (Berkeley, 1973-1980+)
- System R (IBM, 1974-1980+)
- 1974 « SEQUEL: A Structured English Query Language » (IBM, Chamberlin & Boyce)

Codd, Turing award, 1981

L'histoire du modèle relationnel

- **Le modèle relationnel :**
  les données sont stockées dans des tables et les relations entre les données sont établies grâce aux clefs.

- Véritable révolution à l'époque

- A l'origine du modèle relationnel : la logique mathématique

Artist Table

ID	Band
DEP01	Department of Eagles
DEP02	Depeche Mode
DES01	Destiny's Child
DES02	Despite the Rain
DEV01	Devo
DEV01	Devotchka
DEX01	Dexy's Midnight Runners

Album Table

Band	Title	Year	Label	Catalog #
DES02	Despite the Rain	2000	Rain Records	RR1234
DES02	Lost in the Clouds	2003	Record Conglomerate	2003-DTR01

Song Table

Album	Song	Time	Composer
RR1234	Raincloud Blues	3:33	Smith/Jones
RR1234	Umbrella	2:24	Smith/Jones
RR1234	Dead Battery	2:58	Smith/Jones
RR1234	Fish Eye Lens	3:03	Smith/Jones
RR1234	Open Up	3:11	Jones
2003-DTR01	Blue Skies. Red Eyes	4:02	Smith
2003-DTR01	Stop and Smell the Roses	4:21	Jones
2003-DTR01	Freedom Skies	0:25	Smith/Jones
2003-DTR01	Twenty-Six Hours from Home	4:12	Smith/Jones
2003-DTR01	Hallelujah	6:59	Leonard Cohen
2003-DTR01	End of the Rain	3:06	Smith/Jones

The relational model allows one-to-one, one-to-many, and many-to-many relationships.

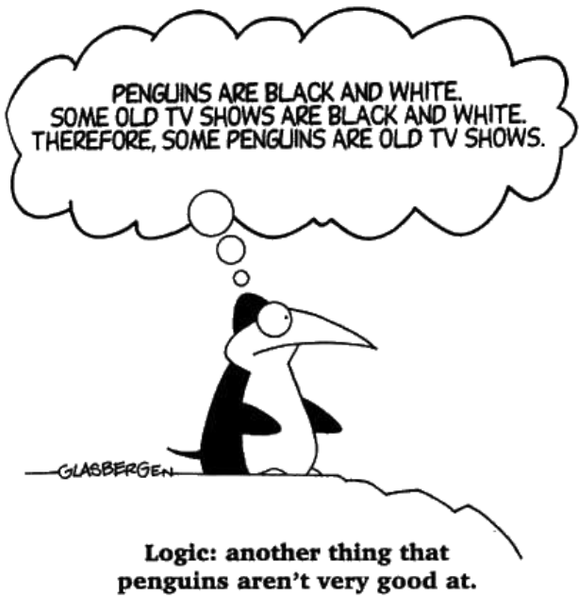
Calcul relationnel

- En introduisant le modèle relationnel, Codd a introduit non seulement l'algèbre relationnelle, mais également le calcul relationnel
- Le calcul relationnel est un langage de requête déclaratif pour les bases de données, basé sur la logique du premier ordre (FO, first-order logic)
- Il existe deux variantes :
  - Tuple relational calculus
  - Domain relational calculus (variante abordée ici)
- Il existe une traduction simple entre les deux formalismes
- Le résultat technique principal de Codd est que l'algèbre relationnelle et le calcul relationnel ont « essentiellement » le même pouvoir expressif

Aux origines du modèle relationnel : la logique du premier ordre

When I was a student, even the topologists regarded mathematical logicians as living in outer space. Today the connections between logic and computers are a matter of engineering practice at every level of computer organization.
(Martin Davis, The universal Turing machine. A half-century survey)

- Histoire (au tableau) :
  - Aristote : les syllogismes
  - Leibniz : lingua characteristics & calculus ratiocinator
  - Crise des fondements : géométries non Euclidiennes et paradoxes de la théorie des ensembles, programme de Hilbert
  - Hilbert et Ackerman 1928 : l'ensemble des énoncés universellement valides du calcul des prédicats du premier ordre est-il décidable ? (problème de la décision)
  - Church Turing 1936 : réponse négative



Le Calcul Relationnel
(logique du premier ordre pour les BD)

- Variables du premier ordre : $x, y, z, \ldots, x_1, \ldots, x_k$
  - ont pour domaine les valeurs susceptibles d'apparaitre dans les tables
- Symboles de relation: $R, S, T, \ldots$ d'arité fixée (noms de relations)
- Formules du calcul relationnel :

connecteurs booléens

$$\varphi := P(t_1, \ldots, t_n) \mid t_1 \text{ op } t_2 \text{ avec op} \in \{ =, \neq, <, >, \leq, \geq \} \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi \mid$$

formules atomiques (de base)

$$\varphi_1 \vee \varphi_2 \mid \neg\varphi \mid \varphi_1 \rightarrow \varphi_2 \mid \exists x\varphi \mid \forall x\varphi$$

où $\varphi, \varphi_1, \varphi_2$ sont des formules

quantificateur existentiel

quantificateur universel

Le Calcul Relationnel
comme langage de requête pour les bases de données

- Définition :
  - une expression du calcul relationnel est de la forme

$$\{(x_1, \ldots, x_k) \mid \varphi(x_1 \ldots, x_k)\}$$

où $\varphi(x_1 \ldots, x_k)$ est une formule du calcul relationnel ayant pour variables libres $x_1, \ldots, x_k$ (i.e., dans la portée d'aucun quantificateur)

  - appliquée à une BD D, cette expression retourne la relation k-aire formée par l'ensemble des k-tuples $a_1, \ldots, a_k$ t.q. $\varphi(a_1 \ldots, a_k)$ est vraie sur D

- Exemple : l'expression du calcul relationnel

$$\{(x, y) \mid \exists z \; Vol(x, z) \wedge Vol(z, y)\}$$

retourne l'ensemble de toutes les paires de villes t.q. la seconde est accessible depuis la première en une escale

Le Calcul Relationnel
comme langage de requête pour les bases de données

- Exemple : Studio(nom, titre), Film(titre, benefice)

(trouver les recettes des films de Dreamworks)

$$\{x \mid \exists y\ Studio(Dreamworks, x) \wedge Film(y, x)\}$$

une requête équivalente :

$$\{x \mid \exists zy\ Studio(z, x) \wedge Film(y, x) \wedge z = Dreamworks\}$$

Le Calcul Relationnel
comme langage de requête pour les bases de données

- Exemple : Studio(nom, titre), Film(titre, benefice)

  trouver les films ayant fait le plus de recettes

$$\{x \mid \exists y(Film(x,y) \wedge \forall zu(Film(z,u) \rightarrow y \geq u))\}$$

- Exercice : trouver les films de Dreamworks ayant fait le plus de recettes, écrire aussi cette requête en algèbre relationnelle et en SQL

La jointure naturelle dans le Calcul Relationnel

- Exemple : R(a, b, c), S(b, c, d)

  en algèbre relationnelle la jointure naturelle $R \bowtie S$ est donnée par

$$\Pi_{R.a,R.b,R.c,S.d}(\sigma_{R.b=S.b \wedge R.c=S.c}(R \times S))$$

en calcul relationnel

$$\{(x_1, x_2, x_3, x_4) \mid R(x_1, x_2, x_3) \wedge S(x_2, x_3, x_4)\}$$

Remarque : la jointure naturelle peut être exprimée par une formule sans quantificateur

La division dans le calcul relationnel

- La division R ÷ S de deux relations R et S est la relation d'arité r – s formée par l'ensemble des tuples (a_1,…,a_{r-s}) t.q. pour tout tuple (b_1,…,b_s) dans S, (a_1,…,a_{r-s}, b_1, …,b_s) est aussi dans R.

- Les films qui passent dans tous les cinémas :

$$Seance \div \Pi_{cinema}(Seance) = \{x \mid \forall z(\exists y\ Seance(z, y) \rightarrow Seance(z, x)\}$$

- Beaucoup plus simple que la traduction dans l'algèbre relationnelle sans ÷

Algèbre relationnelle vs Calcul relationnel

- Théorème de Codd (énoncé informel) :
- l'algèbre relationnel et le calcul relationnel ont « essentiellement » le même pouvoir expressif, ie, ils peuvent exprimer les mêmes requêtes

- Attention : toutes les expressions du calcul relationnel n'ont pas une traduction dans l'algèbre relationnelle

- Exemples :
  - $\{(x_1, \ldots, x_k) \mid \neg R(x_1, \ldots, x_k)\}$
  - $\{x \mid \forall y \; Seance(y, x)\}$

Algèbre relationnelle vs Calcul relationnel

- Attention : de telles expressions du calcul relationnel peuvent générer des réponses différentes en fonction des domaines sur lesquels les variables sont interprétées
- Example : si les variables $x_1,\ldots,x_k$ prennent leurs valeurs sur le domaine D, alors

$$\{(x_1, \ldots, x_k) \mid \neg R(x_1, \ldots, x_k)\} = D^k - R$$

- Faits :
- L'expression du calcul relationnel $\{(x_1, \ldots, x_k) \mid \neg R(x_1, \ldots, x_k)\}$ n'est pas "domaine-indépendante".
- L'expression du calcul relationnel $\{(x_1, \ldots, x_k) \mid S(x_1, \ldots, x_k) \wedge \neg R(x_1, \ldots, x_k)\}$ est "domaine indépendante".(énoncé informel)