

# LE LANGAGE DE REQUETES SQL 89

- ◆ Origines et Evolutions
- ◆ SQL1 86: la base
- ◆ SQL1 89: l'intégrité



Jim Melton (Oracle)  
Editeur de la norme SQL

# 1. Origines et Evolutions

- ◆ SQL est dérivé de l'algèbre relationnelle et de SEQUEL, une invention d'IBM
- ◆ Il existe plusieurs versions normalisées, du simple au complexe :
  - SQL-86 version minimale
  - SQL-89 addendum (intégrité)
  - SQL2 (92) langage complet
  - SQL3 (99) aspects objet, triggers
  - SQL:2003 introduction d'aspects XML
  - SQL:2006 intégration du début de XQuery
  - SQL:2008 modifications mineures (instead of, truncate)
  - SQL:2011 améliorations XQuery
- ◆ La plupart des systèmes supportent SQL2 ou SQL3

# Opérations

- ◆ Opérations de base
  - SELECT, INSERT, UPDATE, DELETE
- ◆ Opérations additionnelles
  - définition et modification de schémas
  - définition de contraintes d'intégrité
  - définition de vues
  - accord des autorisations
  - gestion de transactions

# Organisation du Langage

SQL comprend quatre parties :

1. **Le langage de définition de schéma (Tables, Vues, Droits)**
2. **Le langage de manipulation (Sélection et mises à jour)**
3. *La spécification de modules appelables (Procédures)*
4. *L'intégration aux langages de programmation (Curseurs)*

# SQL 86

- ◆ LANGAGE DE DEFINITIONS DE DONNEES
  - CREATE TABLE
  - CREATE VIEW
- ◆ LANGAGE DE MANIPULATION DE DONNEES
  - SELECT            OPEN
  - INSERT            FETCH
  - UPDATE            CLOSE
  - DELETE
- ◆ LANGAGE DE CONTROLE DE DONNEES
  - GRANT et REVOKE
  - BEGIN et END TRANSACTION
  - COMMIT et ROLLBACK

# Base de Données

- ◆ Collection de tables et de vues dans un schéma

## *TABLES*

RESPONSABLE (NR, NOM, PRENOM, DPT)

COURS (NC, CODE\_COURS, INTITULE, ECTS, NR, DPT)

ETUDIANT (NE, NOM, PRENOM, VILLE, AGE)

INSCRIT (NE, NC, ANNEE)

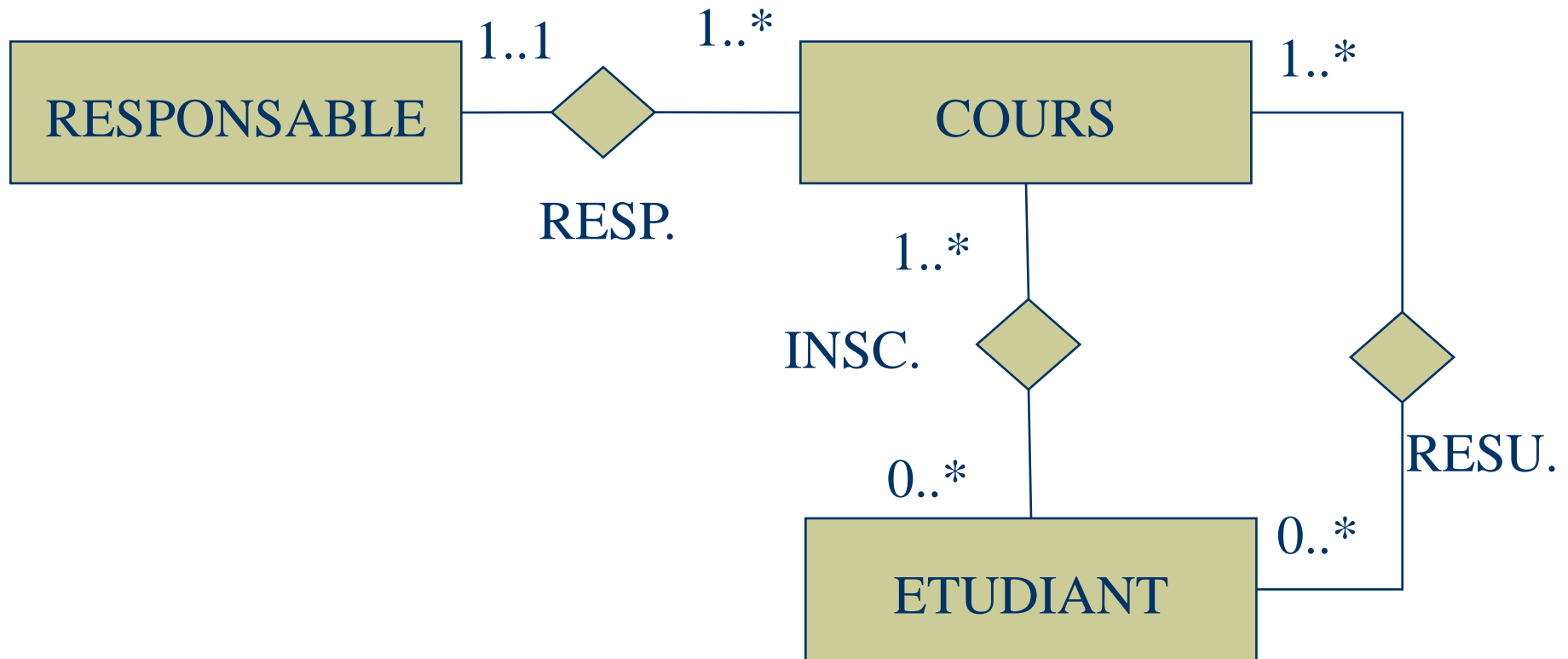
RESULTAT (NE, NC, ANNEE, NOTE)

## *VUES*

ADMIS (NE, NC, ANNEE)

COURS\_DPT (DPT, NC, INTITULE)

# Schéma E/A “allégé” (sans attributs, rôles)



## 2. SELECT: Forme Générale

SELECT <liste de projection>  
FROM <liste de tables>  
[WHERE <critère de jointure> AND <critère de restriction>]  
[GROUP BY <attributs de partitionnement>]  
[HAVING <critère de restriction>]

### ◆ Restriction :

- arithmétique (=, <, >, <>, >=, <=)
- textuelle (LIKE)
- sur intervalle (BETWEEN)
- sur liste (IN)

### ◆ Possibilité de blocs imbriqués par :

- IN, EXISTS, NOT EXISTS, ALL, SOME, ANY



# Forme générale de la condition

**<search condition> ::= [NOT]**  
    **<nom\_colonne>  $\theta$  constante | <nom\_colonne>**  
    **<nom\_colonne> LIKE <modèle\_de\_chîne>**  
    **<nom\_colonne> IN <liste\_de\_valeurs>**  
    **<nom\_colonne>  $\theta$  (ALL | ANY | SOME) <liste\_de\_valeurs>**  
    **EXISTS <liste\_de\_valeurs>**  
    **UNIQUE <liste\_de\_valeurs>**  
    **<tuple> MATCH [UNIQUE] <liste\_de\_tuples>**  
    **<nom\_colonne> BETWEEN constante AND constante**  
    **<search condition> AND | OR <search condition>**

avec

**$\theta ::= < | = | > | \geq | \leq | <>$**

Remarque: **<liste\_de\_valeurs>** peut être dynamiquement déterminée par une requête

# Exemples de Questions (1)

- ♦ Q1: Liste des nom, prenom des étudiants, sans doublons

```
SELECT DISTINCT NOM, PRENOM  
FROM ETUDIANT
```

- ♦ Q2: Noms des étudiants inscrits en IN311 en 2011 ou 2012

Π	SELECT DISTINCT NOM	Projection
×	FROM ETUDIANTS E, COURS C, INSCRIT I	Produit cartésien
WHERE		
⋈	E.NE = I.NE AND I.NC = C.NC	ATTRIBUTS DE JOINTURE
σ	AND C.CODE_COURS LIKE '%IN311%' AND I.ANNEE IN (2011, 2012)	ATTRIBUTS DE RESTRICTION

# Avec la jointure dans le FROM

- ◆ Q2: Noms des étudiants inscrits en IN311 en 2011 ou 2012

SELECT DISTINCT NOM



FROM (ETUDIANTS E JOIN INSCRIT I ON E.NE  
=I.NE) JOIN COURS C ON I.NC=C.NC

WHERE



C.CODE\_COURS LIKE '%IN311%'  
AND I.ANNEE IN (2011, 2012)

# Exemples de Questions (2)

- ♦ Q3 : Noms et prénoms des étudiants inscrits à des cours dont le code commence par IN, entre 2009 et 2012.

$\Pi$	SELECT NOM, PRENOM
$\times$	FROM ETUDIANT E, INSCRIT I, COURS C
$\bowtie$	WHERE E.NE = I.NE AND I.NC = C.NC
$\sigma$	AND C.CODE_COURS LIKE "IN%" AND (I.ANNEE BETWEEN 2009 AND 2012)

- ♦ Q4 : Code des modules suivis par au moins un etudiant. (requête imbriquée)

```
SELECT C.CODE_COURS  
FROM COURS C  
WHERE EXISTS (
```

```
    SELECT *  
    FROM ETUDIANT E, INSCRIT I  
    WHERE E.NE = I.NE AND I.NC = C.NC )
```

Sous  
requête

# Exemples de requêtes agrégat

- ◆ Q5 : Calculez la moyenne de chaque étudiant, référencé par son NE

$\Pi$  et  $\gamma_{AVG(R.NOTE)}$  SELECT E.NE, E.NOM, AVG(R.NOTE)

$\bowtie$  FROM ETUDIANT E, RESULTAT R  
WHERE E.NE = R.NE

$\gamma_{E.NE}$  GROUP BY E.NE

$\sigma$  HAVING COUNT(DISTINCT R.NC)  $\geq 5$

Pour les étudiants ayant suivi plus de 5 modules ...

# Exemples de requêtes agrégat

- ◆ Q5' : Calculez la note maximale de chaque module, référencé par son NC

Π et

SELECT C.NC, C.INTITULE, MAX(R.NOTE)

γ<sub>MAX(R.NOTE)</sub>



FROM COURS C, RESULTAT R

WHERE C.NC = R.NC

E.NE γ

GROUP BY C.NC



HAVING COUNT(DISTINCT R.NE) >= 15

Pour les modules de plus de 15 étudiants...

# Exemples de Requêtes agrégat

- ♦ Q6: Calculer l'âge du plus jeune étudiant

$\Pi$  et  $\gamma_{\text{MIN(AGE)}}$  **SELECT MIN(AGE)**  
**FROM ETUDIANT**

- ♦ Q7 : Calculer le nombre d'étudiants, ainsi que l'âge de l'étudiant le plus vieux reçus par module, pour les modules du dpt INFO dont la moyenne globale est supérieure à 12.

$\Pi$  et  $\gamma_{\text{COUNT(*), MAX(AGE)}}$  **SELECT C.NC, COUNT(\*), MAX(E.AGE)**  
**FROM ETUDIANT E, RESULTAT R, COURS C**  
**WHERE E.NE = R.NE AND R.NC = C.NC**  
**AND C.DPT = "INFO"**  
**GROUP BY C.NC**  
**HAVING AVG(R.NOTE) > 12**

# Requêtes agrégat et fonctions

- ♦ Q8 : Donnez le nombre d'ECTS obtenus par chaque étudiant, référencé par son NE, dans une colonne appelée CREDITS.

```
SELECT E.NE, E.NOM,  
FROM ETUDIANT E, RESULTAT R, COURS C  
WHERE E.NE = R.NE AND R.NC = C.NC
```

```
σ AND R.NOTE >= 10
```

```
GROUP BY E.NE
```

CALCUL DE FONCTION  
SUM(C.ECTS)

P<sub>SUM(R.ECTS)</sub>  
→ CREDITS  
AS CREDITS

!/ ICI ON NE PREND PAS EN COMPTE LA COMPENSATION !



# Requêtes agrégat et fonctions

- ♦ Q8' : Donnez le nombre d'ECTS obtenus par chaque étudiant, référencé par son NE, dans une colonne appelée CREDITS.

```
SELECT E.NE, E.NOM, SUM(C.ECTS) AS CREDITS
FROM ETUDIANT E, RESULTAT R, COURS C
WHERE E.NE = R.NE AND R.NC = C.NC
GROUP BY E.NE
```

```
σ HAVING AVG(R.NOTE) >= 10
```

!/ COMBIEN D'ECTS A L'ETUDIANT S'IL N'A PAS LA MOYENNE  
AU SEMESTRE ?

# Requêtes agrégat et fonctions

```
Q9 : SELECT CALCUL.NE, MAX(CALCUL.CREDITS)
FROM
( SELECT E1.NE, SUM(C1.ECTS)  AS CREDITS
FROM ETUDIANT E1, RESULTAT R1, COURS C1
WHERE E1.NE = R1.NE AND R1.NC = C1.NC
AND R1.NOTE >= 10
GROUP BY E1.NE
UNION
SELECT E2.NE, SUM(C2.ECTS)  AS CREDITS
FROM ETUDIANT E2, RESULTAT R2, COURS C2
WHERE E2.NE = R2.NE AND R2.NC = C2.NC
GROUP BY E2.NE
HAVING AVG(R2.NOTE) >= 10 ) AS CALCUL
GROUP BY CALCUL.NE
```

Peut on faire sans union ?

# Requêtes agrégat et fonctions

## CALCUL DE FONCTION

```
SELECT NORMAL.NE, GREATEST(NORMAL.CREDITS, COMPENSE.CREDITS)
```

```
AS CREDITS
```

$\rho \rightarrow \text{CREDITS}$

```
FROM
```

```
( SELECT E1.NE, SUM(C1.ECTS) AS CREDITS  
FROM ETUDIANT E1, RESULTAT R1, COURS C1  
WHERE E1.NE = R1.NE AND R1.NC = C1. NC  
AND R1.NOTE >= 10
```

```
GROUP BY E1.NE ) AS NORMAL, Table « NORMAL »
```

```
(SELECT E2.NE, SUM(C2.ECTS) AS CREDITS  
FROM ETUDIANT E2, RESULTAT R2, COURS C2  
WHERE E2.NE = R2.NE AND R1.NC = C1. NC  
GROUP BY E2.NE
```

```
Table « COMPENSE »  
HAVING AVG(R2.NOTE) >= 10 ) AS COMPENSE
```

```
WHERE NORMAL.NE = COMPENSE.NE
```



#!/ Problème des étudiants qui n'ont pas la moyenne des notes >= 10

# Requêtes imbriquées (1)

- ♦ Q10: Donner les CODE\_COURS des cours qui n'ont aucun inscrit

```
SELECT CODE_COURS
FROM COURS C
WHERE C.NC NOT IN
( SELECT I.NC
  FROM INSCRIT I )
```


```
SELECT CODE_COURS
FROM COURS C
WHERE C.NC <> ALL
( SELECT I.NC
  FROM INSCRIT I )
```

On ne se ressort pas forcément de la requête extérieure

# Requêtes imbriquées (2)

- ♦ Q11 : Donner le NE des étudiants qui suivent tous les cours

```
SELECT NE
FROM ETUDIANT E
WHERE NOT EXISTS (
  SELECT *
  FROM COURS C
  WHERE NOT EXISTS (
    SELECT *
    FROM INSCRIT I
    WHERE C.NC = I.NC
    AND I.NE = E.NE) )
```



Requête doublement imbriquée !

# Requêtes imbriquées (3)

- ♦ Q11' : Donner le NE des étudiants qui suivent tous les cours

```
SELECT E.NE
FROM ETUDIANT E, INCRIT I, COURS C
WHERE E.NE = I.NE AND I.NC = C.NC
GROUP BY E.NE
HAVING COUNT(DISTINCT C.NC) = (SELECT COUNT(*) FROM COURS C)
```

# Utilisation de SQL depuis un langage de prog.

- ◆ Il est très fréquent d'utiliser SQL à partir de programmes
  - Applications C++/Java/etc.
  - Applications Web (PHP, etc.)
- ◆ L'utilisation de bibliothèques JDBC est conseillée
- ◆ ... plus de détails dans le cours sur PHP et Java.

# 3. Les Mises à Jour

## ◆ INSERT

- Insertion de lignes dans une table
- Via formulaire où via requêtes

## ◆ UPDATE

- Modification de lignes dans une table

## ◆ DELETE

- Modification de lignes dans une table



# Commande INSERT

- ◆ INSERT INTO <relation name>  
[( attribute [,attribute] ... )]  
{VALUES <value spec.> [, <value spec.>] ...| <query spec.>}
- ◆ Exemples

```
INSERT INTO ETUDIANT (NE, NOM, PRENOM, VILLE, AGE)  
VALUES (112, 'MARTIN', 'THOMAS', 'VERSAILLES', 20)
```

```
INSERT INTO RESULTAT (NC, NE, ANNEE, NOTE)  
SELECT C.NC, I.NE, 2013 AS ANNEE, 20 AS NOTE  
FROM COURS C, INSCRIT I  
WHERE C.CODE_COURS = 'IN311'  
AND C.NC = I.NC
```

# Commande UPDATE

UPDATE <relation name>

SET <attribute = {value expression | NULL}

[<attribute> = {value expression | NULL}] ...

[WHERE <search condition>]

## ◆ EXAMPLE

```
UPDATE RESULTAT
```

```
SET NOTE = NOTE * 1.2
```

```
WHERE RESULTAT.NC IN
```

```
( SELECT NC
```

```
  FROM COURS C
```

```
  WHERE C.CODE_COURS = 'IN311' )
```

# Commande DELETE

```
DELETE FROM <relation name>  
[WHERE <search condition>]
```

## ◆ EXEMPLE

```
DELETE FROM RESULTAT  
WHERE NC IN  
  SELECT C.NC  
  FROM COURS C  
  WHERE C.CODE_COURS = 'IN311'
```

## 4. Contraintes d'intégrité

- ◆ Contraintes de domaine
  - Valeurs possibles pour une colonne
- ◆ Contraintes de clés primaires
  - Clé et unicité
- ◆ Contraintes référentielles (clé étrangères)
  - Définition des liens inter-tables

# SQL1 - 89 : INTEGRITE

- ◆ VALEURS PAR DEFAUT

CREATE TABLE ETUDIANT

( NE INT(5) PRIMARY KEY,

NOM VARCHAR(128),

PRENOM VARCHAR(128),

VILLE VARCHAR(128),

AGE INT(3) CHECK BETWEEN 10 AND 120)

- ◆ CONTRAINTES DE DOMAINES

# SQL1 - 89 :

## Contrainte référentielle

- ◆ Clé primaire et contrainte référentielle

```
CREATE TABLE INSCRIT
```

```
( NC INT(5),
```

```
  NE INT (5),
```

```
  ANNEE INT(4),
```

```
  PRIMARY KEY (NC, NE, ANNEE),
```

```
  FOREIGN KEY (NC) REFERENCES COURS(NC),
```

```
  FOREIGN KEY (NE) REFERENCES ETUDIANT(NE) )
```

- ◆ Référence en principe la clé primaire

- celle de COURS et celle de ETUDIANT

# SQL1 – 89 : Création de table

```
CREATE TABLE <nom_table>  
(<def_colonne> *  
[<def_contrainte_table>*]) ;
```

< def\_colonne > ::=

<nom\_colonne> < type | nom\_domaine >

[CONSTRAINT nom\_contrainte

< NOT NULL | UNIQUE | PRIMARY KEY |

CHECK (condition) | REFERENCES nom\_table (colonne) > ]

< def\_contrainte\_table > ::= CONSTRAINT nom\_contrainte

< UNIQUE (liste\_colonnes) | PRIMARY KEY (liste\_colonnes) |

CHECK (condition) |

FOREIGN KEY (liste\_colonnes) REFERENCES nom\_table (liste\_colonnes) >

## 5. CONCLUSION

- ◆ SQL1 est un standard minimum
- ◆ Les versions étendues:
  - SQL2 = Complétude relationnelle
  - SQL3 = Support de l'objet
  - SQL:2006 = Extension à XQuery
  - Par la suite ... pas encore de grande révolution :  
rajout de fonctions mineures
- ◆ Sont aujourd'hui intégrées dans les grands SGBD



# Les grandes bases de données (sept 2012)

## ◆ « Commerciaux »

- Oracle (11g) version 11.2
- IBM DB2 version 10
- Microsoft SQL Server 2012
- Sybase ASE 15.7 (SAP)

## ◆ « Open Source »

- MySQL v. 5.5 (<http://www.mysql.org/>)
- Postgres v. 9.2 (<http://www.postgresql.org/>)

# LA NORMALISATION DE SQL

- ◆ Groupe de travail ANSI/X3/H2 et ISO/IEC JTC1/SC2
- ◆ Documents ISO :
  - SQL1 - 86 : Database Language SQL X3.135 ISO-9075-1987)
  - SQL1 - 89 : Database Language SQL with Integrity Enhancement X3.168 ISO-9075-1989
  - SQL:1999 (SQL3)
  - SQL2 - 92 : Database Language SQL2 X3.135 ISO-9075-1992
  - SQL:2003 ISO/IEC 9075:2003
  - SQL:2006 ISO/IEC 9075-14:2006
  - SQL:2008 ISO/IEC 9075:2008

Etc..