SQL

リレーショナルデータベース言語

SQL

標準化されたリレーショナルデータベース言語

- 1986年：ANSIで最初に標準化
- 1987年：ISO で第1版，JIS としても制定
- 1989年：ISO，ANSI で整合性制約などの機能を拡張した改訂版が規格化

  JIS でも1990年に規格化
- 1992年：SQL2として検討された規格が ISO，ANSI で制定

  JIS でも1995年に制定
- 1999年：オブジェクト指向拡張が施されたSQL:1999が規格化

標準化のメリット

- ユーザは1つのデータベース言語を学ぶことで各種DBMSを利用できる
- 異なるDBMS間でのアプリケーションプログラムの移植や連携が容易になる
- 汎用性のあるツールやユーティリティの開発や利用が容易になる
- DBMS利用技術のノウハウの共有が促進される
- ・・・

機能の概略

- 生成 (データベースやテーブルなど)
- アクセス (入力，問合せ，修正，削除 など)
- 制御 (トランザクション処理など)
- 保護 (アクセス権など)

規格化されている機能

- データ型などの諸概念
- 問合せ式
- スキーマ定義およびスキーマ操作
- モジュール
- データ操作言語
- トランザクション管理
- 動的SQL
- 情報スキーマおよび定義スキーマ
- 適合性
- ・・・

用語定義

リレーショナルデータモデル	SQL
リレーション (relation)	表 (table)
属性 (attribute)	列 (column)
タップル (tuple)	行 (row)

表の種類

- 実表 (base table)
- ビュー表 (view table)
- 導出表 (derived table)

教科書のSQLの実行例で用いられている表

部門

部門番号	部門名	部門長
K55	データベース	0650
K41	ネットワーク	1508

社員

社員番号	社員名	給与	所属
0650	山田太郎	50	K55
1508	鈴木花子	40	K41
0231	田中桃子	60	K41
2034	佐藤一郎	40	K55
0713	渡部美咲	60	K55

表の作成

実行例で用いる表の定義 (文法については次回)

(教科書のテーブル例をそのまま定義しているので漢字を使っているが，漢字を使うことは推奨しない)

```
CREATE TABLE 部門 (
  部門番号 char(3) PRIMARY KEY,
  部門名 varchar(20),
  部門長 char(4)
)
CREATE TABLE 社員 (
  社員番号 char(4) PRIMARY KEY,
  社員名 varchar(20),
  給与 integer,
  所属 char(3),
  FOREIGN KEY (所属) REFERENCES 部門(部門番号)
)
```

主キーの定義

外部キーの定義

データ入力

実行例で用いるデータ入力の命令列 (文法については次回)

部門では1つずつ入力し，社員では複数個をまとめて入力している

```
INSERT INTO 部門 VALUES ('K55','データベース','0650')
INSERT INTO 部門 VALUES ('K41','ネットワーク','1508')

INSERT INTO 社員 VALUES ('0650','山田太郎',50,'K55'),
('1508','鈴木花子',40,'K41'),('0231','田中桃子',60,'K41'),
('2034','佐藤一郎',40,'K55'),('0713','渡部美咲',60,'K55')
```

問合せ指定

- 基本構文は教科書 図5.1 参照
- 基本形

  SELECT 〈値式$_1$〉,〈値式$_2$〉, ･･･,〈値式$_n$〉
  FROM 〈表参照$_1$〉,〈表参照$_2$〉, ･･･,〈表参照$_n$〉
  WHERE 〈探索条件〉

- その他の句
  - 結果をソートしたい場合は ORDER BY を用いる
  - 集約関数 (COUNT, SUM, AVG, ･･･) を使えるが, 対象の列を GROUP BY でグループ化する必要がある
  - 集約関数を条件にする場合は WHERE ではなく HAVING を用いる
- その他
  - 数値や特殊な語句以外の値はシングルクオートで囲む

単純質問

SELECT文のFROM句にただ1つの〈表参照〉が指定され，
WHERE句の〈探索条件〉中にSELECT文が入れ子構造で入らない

全データを閲覧する操作例 （WHERE句がない）

```
SELECT ＊
FROM 社員
```

社員番号	社員名	給与	所属
0650	山田太郎	50	K55
1508	鈴木花子	40	K41
0231	田中桃子	60	K41
2034	佐藤一郎	40	K55
0713	渡部美咲	60	K55

選択演算に対応する操作例 (所属が K55 の社員一覧)

```
SELECT ＊
FROM 社員
WHERE 所属='K55'
```

社員番号	社員名	給与	所属
0650	山田太郎	50	K55
2034	佐藤一郎	40	K55
0713	渡部美咲	60	K55

射影演算に対応する操作例 (給与の一覧)

結果に重複する行があっても1つにしない
(SQLでは許される)

```
SELECT 給与
FROM 社員
```

給与
50
40
60
40
60

結果に重複する行があれば1つにする

```
SELECT DISTINCT 給与
FROM 社員
```

給与
50
40
60

選択演算結果に射影演算を行った例
(所属が'K55'の社員の社員番号，社員名，給与)

```
SELECT 社員番号，社員名，給与
FROM 社員
WHERE 所属='K55'
```

社員番号	社員名	給与
0650	山田太郎	50
2034	佐藤一郎	40
0713	渡部美咲	60

AND条件の例
('K55'に所属 かつ 給与が50以上)

```
SELECT 社員番号，社員名
FROM 社員
WHERE 所属='K55' AND
      給与>=50
```

社員番号	社員名
0650	山田太郎
0713	渡部美咲

選択リストに式を指定した例

```
SELECT 社員番号, 社員名, 給与, 給与*0.8
FROM 社員
```

導出表の値式 給与*0.8 に対応する，列名がないが，gin.eng.kagawa-u.ac.jp 上の mysql では 給与*0.8 と表示された (2022/10/27 調べ)

社員番号	社員名	給与	
0650	山田太郎	50	40
1508	鈴木花子	40	32
0231	田中桃子	60	48
2034	佐藤一郎	40	32
0713	渡部美咲	60	48

条件に BETWEEN, IN, LIKE, NULL, EXISTS などの述語を使えるが，以下は，BETWEEN の例

```
SELECT *
FROM 社員
WHERE 給与 BETWEEN 40 AND 50
```

社員番号	社員名	給与	所属
0650	山田太郎	50	K55
1508	鈴木花子	40	K41
2034	佐藤一郎	40	K55

IN を使った例 (給与が40か60)

```
SELECT 社員番号, 社員名, 給与
FROM 社員
WHERE 給与 IN (40, 60)
```

社員番号	社員名	給与
1508	鈴木花子	40
0231	田中桃子	60
2034	佐藤一郎	40
0713	渡部美咲	60

ORDER BY, GROUP BY, HAVING 句や
集約関数 (COUNT, SUM, AVG, MAX, MIN) が使える

集約関数を用いた例 (3人以上の部門の給与の平均)
GROUP BY で対象をまとめる

```
SELECT 所属, AVG(給与)
FROM 社員
GROUP BY 所属
HAVING COUNT(*)>=3
```

所属	
K55	50

結合操作

FROM句に2つ以上の表を指定することで直積が行われ，
その中からWHERE句で結合の条件を指定して選択する
(2つの表で属性名が同じものがなければ，ドット表記にしなくても良い)

社員と部門の全ての組み合わせの取得 (=直積)

```
SELECT 社員.*, 部門.*
FROM 社員, 部門
```

```
SELECT *
FROM 社員, 部門
```

でも結果は同じ

社員番号	社員名	給与	所属	部門番号	部門名	部門長
0231	田中桃子	60	K41	K41	ネットワーク	1508
0231	田中桃子	60	K41	K55	データベース	0650
0650	山田太郎	50	K55	K41	ネットワーク	1508
0650	山田太郎	50	K55	K55	データベース	0650
0713	渡部美咲	60	K55	K41	ネットワーク	1508
0713	渡部美咲	60	K55	K55	データベース	0650
1508	鈴木花子	40	K41	K41	ネットワーク	1508
1508	鈴木花子	40	K41	K55	データベース	0650
2034	佐藤一郎	40	K55	K41	ネットワーク	1508
2034	佐藤一郎	40	K55	K55	データベース	0650

データベース部に所属している社員の社員番号と社員名の取得

```
SELECT X.社員番号, X.社員名
FROM 社員 X, 部門 Y
WHERE X.所属=Y.部門番号 AND
      Y.部門名='データベース'
```

社員番号	社員名
0650	山田太郎
2034	佐藤一郎
0713	渡部美咲

上司よりも高給の社員の社員番号と社員名の取得

```
SELECT X.社員番号, X.社員名
FROM 社員 X, 部門 Y, 社員 Z
WHERE X.所属=Y.部門番号 AND
      Y.部門長=Z.社員番号 AND
      X.給与>Z.給与
```

社員番号	社員名
0231	田中桃子
0713	渡部美咲

FROM句で結合の条件を指定できる (結果は前ページの1つめと同じ)
2つの表の間に INNER JOIN を置き，WHERE の代わりに ON を用いる

```
SELECT X.社員番号, X.社員名
FROM 社員 X INNER JOIN 部門 Y
  ON X.所属=Y.部門番号 AND
     Y.部門名='データベース'
```

結合した結果から WHERE句で条件に合う行を選択できる

```
SELECT X.社員番号, X.社員名
FROM 社員 X INNER JOIN 部門 Y
  ON X.所属=Y.部門番号 AND
     Y.部門名='データベース'
WHERE X.給与 > 50
```

社員番号	社員名
0713	渡部美咲

入れ子型質問 (nested query)

- WHERE句内に**問合せが入れ子で出現する**問合せ
- 入れ子になっている問合せをSQLでは副問合せ (subquery) と言う
- 入れ子が多段になっても良い
- 副問合せの結果は，比較や IN で用いる事ができるものでなければならない
- 入れ子の問合せは，それを入れ子にしている外側の問合せと独立している場合や，相関を有する場合がある

平均給料より高給の社員の取得 (外側の問合せと独立)

```
SELECT ＊
FROM 社員
WHERE 給与 ＞
        (SELECT AVG(給与)
         FROM 社員)
```

社員番号	社員名	給与	所属
0231	田中桃子	60	K41
0713	渡部美咲	60	K55

上司よりも高給の社員の社員番号と社員名の取得 (外側の問合せと相関を有する)

```
SELECT X.社員番号, X.社員名
FROM 社員 X
WHERE X.給与 ＞
        (SELECT Z.給与
         FROM 部門 Y, 社員 Z
         WHERE X.所属=Y.部門番号 AND
               Y.部門長=Z.社員番号)
```

社員番号	社員名
0231	田中桃子
0713	渡部美咲

NULL値の検査方法

NULL値であるかどうかの検査は

```
IS NULL
```

で行う

空値を含む行を入力してみる

```
INSERT INTO 社員 VALUES ('0999',NULL,100,'K41')
```

条件式にIS NULLを含む例

```
SELECT *
FROM 社員
WHERE 社員名 IS NULL
```

社員番号	社員名	給与	所属
0999		100	K41

社員番号	社員名	給与	所属
000	山田太郎	50	K55
1508	鈴木花子	40	K41
0231	田中桃子	60	K41
2034	佐藤一郎	40	K55
0713	渡部美咲	60	K55
0999		100	K41

パターンマッチによる検索

パターンを指定して特定の文字列を含むものを調べることができる

同じパターンを含むデータを入力しておく

```
INSERT INTO 社員 VALUES ('0123','香川照之',100,'K55')
INSERT INTO 社員 VALUES ('0555','香川真司',100,'K55')
```

社員番号	社員名	給与	所属
000	山田太郎	50	K55
1508	鈴木花子	40	K41
0231	田中桃子	60	K41
2034	佐藤一郎	40	K55
0713	渡部美咲	60	K55
0123	香川照之	100	K55
0555	香川真司	100	K55

’香川’を含む

```
SELECT *
FROM 社員
WHERE 社員名 LIKE '%香川%'
```

’%’ は0文字以上にマッチ
’_’ は1文字にマッチ

社員番号	社員名	給与	所属
0123	香川照之	100	K55
0555	香川真司	100	K55

データ更新

UPDATE を用いて行う

```
UPDATE 表名 SET 列名={式 | DEFAULT} [, ...]
[WHERE 検索条件]
```

WHERE の検索条件で選択された行の SET で指定した列のみ変更される

社員の所属が変わる場合の例

```
UPDATE 社員 SET 所属='K51'
WHERE 社員番号='0650'
```

所属の異動と同時に給与が上がる場合の例

```
UPDATE 社員 SET 所属='K51',給与=給与+5
WHERE 社員番号='0650'
```

データの削除

指定した行のみ削除する場合は DELETE を用いる

```
DELETE FROM 表名
[WHERE 検索条件]
```

WHERE の検索条件で選択された行を削除する

社員番号が’0650’の社員が退職していなくなった場合の例

```
DELETE FROM 社員 WHERE 社員番号='0650'
```

表のデータ全てを削除する場合，

```
DELETE FROM 表名
```

で削除できるが TRUNCATE を用いることもできる

```
TRUNCATE [TABLE] 表名
```

埋込みSQL

- SQL単体では通常のプログラミング言語が有している計算能力がない

↓

- 既存のプログラミング言語にSQLを埋込み，リレーショナルデータベースへのアクセス部分はSQLが，計算の部分はプログラミング言語が行う言語システムが考えられた

Webシステム開発（3年前期）の前半では，埋込みSQLではないが，プログラミング言語であるPHPからSQLを呼び出してデータベースを操作するWebアプリケーションを作成する演習を行う