リレーショナルデータベースのデータ構造と一貫性制約

URL: http://www.db.is.kyushu-u.ac.jp/rinkou/addb/2.html

- **直積集合**(「直積」とも呼ぶ)

  n 個の集合 S1, S2, … , Sn が与えられたとき, **{(X1, X2, … ,Xn) | X1∈S1, X2∈S2, …, Xn∈Sn}** を S1, S2, …, Sn の直積集合と呼び, S1 × S2 × … × Sn と記す. 例えば, S1 = {1, 2}, S2 = {a, b, c} とするとき S1 × S2 = {(1, a), (1, b), (1, c), (2, a), (2, b), (2, c)}

  ※ S2 × S1 = {(a, 1), (b, 1), (c, 1), (a, 2), (b, 2), (c, 2)} となり, S1×S2 とは違う集合である.

- **ドメイン (domain)**

  ドメインとは**属性が取りえる値の集合 (a set of permitted value)**のこと. 例えば, 属性「年齢」のドメインは, 整数値の集合 {0,1,2. … , 120} のようになるだろう.

- **リレーション (relation) (「関係」とも呼ぶ)**

  D1, D2, … , Dn をドメインとするとき, D1, D2, …, Dn 上のリレーションとは, **直積集合「D1 × D2 × … × Dn」の任意の有限部分集合**を言う.

  例えば, 8つの属性 id, year, month, day, customer_name, product_name, unit_price, qty があり, それぞれのドメインが次のようになっているとする.

  - id のドメイン: 整数
  - year のドメイン: {2009, 2010, ...}
  - month のドメイン: {1, 2, ..., 12}
  - day のドメイン: {1, 2, ..., 31}
  - customer_name のドメイン: 文字列
  - product_name のドメイン: 文字列
  - unit_price のドメイン: 0 よりも大きい実数
  - qty のドメイン: 0 よりも大きい整数

  このとき, 次の図で表される集合は 1つのリレーションである. ここでは, **ドメインの直積集合の要素を1つの行**として書いている.

1	2009	10	26	kaneko	orange A	1.2	10
2	2009	10	26	miyamoto	Apple M	2.5	2
3	2009	10	27	kaneko	orange B	1.2	8
4	2009	19	28	miyamoto	Apple L	3	1

- **多重集合 (multi-set)**

  要素の重複を許すような集合のことを**多重集合**という. リレーショナルデータベースにおいて多重集合という概念が導入されている理由を, 具体例で説明すると下記の通りである.

```
name        | score |  student_name
-------------------------------------
Database    |   80  |  KK
Database    |   95  |  AA
Database    |   80  |  LL
Programming |   85  |  KK
Programming |   75  |  LL
```

  name = Database (科目名が「Database」)である行の score (得点) は {80, 95, 80} という**多重集合**である. この多重集合を見て, 平均点85, 受講者3名ということが分かる. 多重集合という概念がない場合には, name (科目名) A の得点は {80, 95} という集合になり, 平均点や受講者数が分からず困ったことになる.

- **テーブル (table)**

  リレーショナルデータベースでは, データベースを**テーブル**の集まり (collection of tables) として記述する. リレーショナルデータベースでのテーブルは, ヘッダと本体からなる.

  - ヘッダ: 属性名が並ぶ.
  - 本体: 属性のドメインの直積集合の要素からなる多重集合

  【テーブルの例(本体に重複値がない)】

id	year	month	day	customer name	product name	unit price	qty	created at	updated at
1	2009	10	26	kaneko	orange A	1.2	10		
2	2009	10	26	miyamoto	Apple M	2.5	2		
3	2009	10	27	kaneko	orange B	1.2	8		
4	2009	19	28	miyamoto	Apple L	3	1		

※各行が挿入された日時 ※ 各行の最終更新日時

  【テーブルの例(本体に重複値がある)】

得点
80
95
80

以上のように，テーブルの本体には**リレーションを格納することができる**(リレーショナルデータベースに「リレーショナル」という言葉が付いているのはそのため)し，重複値を持つような多重集合を格納することもできる.

- **テーブル名 (table name)**

  テーブル名とは，**個々のテーブルに付けられた名前**のこと.

- **属性 (attribute)**

  リレーションデータベースのテーブルでは，**テーブルの各列が1つの属性をなす**．属性名は各列に付けられた名前であり，テーブルのヘッダに書かれる．例えば，下記のテーブルには，id, year, month, day, customer_name, product_name, unit_price, qty という名前の属性がある.

id	year	month	day	customer name	product name	unit price	qty	created at	updated at
1	2009	10	26	kaneko	orange A	1.2	10		
2	2009	10	26	miyamoto	Apple M	2.5	2		
3	2009	10	27	kaneko	orange B	1.2	8		
4	2009	19	28	miyamoto	Apple L	3	1		

※各行が挿入された日時 ※ 各行の最終更新日時

※ テーブルのタップルが更新(例えば，テーブルに新しいタップルが挿入されたり，テーブルからタップルが削除されたり，タップルの値が変化したり)されても，**属性名は不変**である(変化しない).

- **行 (row)**

  リレーショナルデータベースで「行」というときは，テーブルの本体の各行のこと.

- **リレーショナルデータモデル**

  リレーショナルデータモデルとは，次の特徴を持ったデータモデルである.

  1. データベースを，**テーブルの集まり**として記述
  2. データ操作の体系は，リレーショナル代数演算やリレーショナル論理(リレーショナル代数演算とリレーショナル論理は等価であるといってよい).
  3. リレーショナルデータモデルの一貫性制約には，主キー制約，一意制約，非空制約，ドメイン制約，参照整合性制約などがある.

- **リレーショナルデータベース**

  リレーショナルデータモデルで記述されたデータベース

- **リレーショナルデータベース管理システム**

  リレーショナルデータモデルの機能を持つデータベース管理システム

- **データ型 (data type)**

  データベースで「データ型」というときは，普通，データベースが扱う属性のデータ型のことをいう.

- **空値 (「NULL」とも呼ぶ)**

  空値はさまざまな意味で使われる.

  - **値が存在しない (does not exist)** ・・・ 例えば，自由席の場合には「予約座席番号」が存在しない.「送付を希望しない」ときは住所が空欄.
  - **未定 (not yet decided)** ・・・ 例えば，「データベース」の授業の担当者がまだ決まっていないようなとき
  - **値が未知く (unknown)** ・・・ 例えば，住所がまだ分からないとき(住所は存在するが、まだ知らない)

- **リレーションスキーマ (relation schema)**

  テーブルにはテーブル名と属性名が付与される． R をテーブル名，A1, A2, …, An を属性名とするとき，**リレーションスキーマを「R(A1, A2, …, An)」のように書く**.

  リレーションスキーマは，個々のテーブルの枠組みを記述している．**テーブルのタップルが更新**(テーブルに新しいタップルが挿入されたり，テーブルからタップルが削除されたり，タップルの値が変化したり)されても，**テーブル名と属性名は不変であるから，リレーションスキーマも不変**である.

- **キー(「候補キー」ともいう)**

  リレーショナルデータモデルで「キー」というときは，次の意味である.

  > **テーブルのタップルを唯一に識別するのに使える属性あるいは属性の組のうち極小なもの**をキーという.

  キーである属性，あるいは，属性の組については，テーブルの2つ以上のタップルが,いかなる場合であっても,同一の属性値を持つことはありえない．例えば，職員のデータベースでは，住所と氏名のペアがキーになるだろう(住所，氏名の片方だけではキーにはならない)．あるいは，職員に固有のIDが付いている場合には，そのIDがキーになるだろう.

- **主キー**

  リレーショナルデータモデルにおいて，ある1個のリレーションスキーマに対して，キーは複数種類ありえる．その中で，**データベース管理者が，データベースの管理上最も適当と判断し，かつ，空値をとることがありえないものを，リレーションスキーマの設計時に選んだもの**が主キーである．あるリレーションスキーマに対して「キー」が1種類しかないときは，それが必然的に主キーになる.

  ※ 主キーとして選ばれたキーの属性あるいは属性の組には，**空値を格納することはできない**という規則がある

- **リレーショナルデータモデルでの一貫性制約**

  一貫性制約とは，データベースが実世界を正しく反映しているときに満足せねばならない制約条件のこと．例えば，誕生年として「3009」が記録されているようなデータベースは，とても実世界を正しく反映しているとは言えず，信用出来ない．誕生年には，

例えば，「1889以上2009以下」のような一貫性制約があるはず．但し，一貫性制約の中身は．データベースの用途によっても変わってくることに注意．今後10年間データベースを使い続けると分かっているなら「1889以上2019以下」のようになるだろう．

リレーショナルデータモデルでの一貫性制約とは，リレーションスキーマのインスタンスの任意のタップル（言い換えると，リレーションスキーマのインスタンスの各要素）の各成分の値が，満足せねばならない制約条件のこと．リレーショナルデータベース管理システムには，普通，一貫性制約を記述できる機能，記述された一貫性制約に反するような更新を受け付けなくする機能がある．リレーショナルデータモデルでの一貫性制約は種々の種類がある．例示すると下記の通り．

- **主キー制約**
  主キーであると指定された属性（あるいは属性の組）については，**主キーとしての条件を満足せねばならない**という制約
- **一意制約**
  一意であると指定された属性（あるいは属性の組）については，**同じ値が2回以上現れない**という制約．
- **非空制約**
  非空であると指定された属性は，**空値（NULL）になってはいけない**という制約
- **ドメイン制約**
  テーブルのタップルの各成分の値は，対応する属性のドメインの要素でなければならないという制約．（例）「試験の点数」という属性のドメインが「{x|xは0以上100以下の整数}」であると定められているとき，試験の点数の値が，この範囲外になってはいけない．
- **参照整合性制約** (referential integrity)
  外部キーであると指定された属性（あるいは属性の組）が，外部キーとしての条件を満足せねばならないという制約

- **SQL**
  SQL は，リレーショナルデータベース言語の国際標準である．平易な英文として，簡単に読み下せるようになっている．
- **SQLite**
  SQLite とは，演習で使用するデータベース管理システムの名前である．

演習で行うこと

- SQLite データベースの新規作成 (Create a new SQLite database)
  データベース論理名: **C:¥SQLite¥mydb**
- SQL を用いたテーブル定義と一貫性制約の記述 (Table defintion and integrity constraint specification using SQL)
  リレーショナル・スキーマ (relational schema): **score_records(name, score, student_name, created_at, updated_at)**

  SQL プログラム:

```
CREATE TABLE score_records (
    name         TEXT     NOT NULL,
    score        INTEGER  NOT NULL CHECK ( score >= 0 AND score <=100 ),
    student_name TEXT     NOT NULL,
    created_at   DATETIME NOT NULL,
    updated_at   DATETIME,
    UNIQUE (name, student_name) );
```

- SQL を用いたテーブルへの行の挿入 (Insert rows into a table using SQL)

```
BEGIN TRANSACTION;
INSERT INTO score_records VALUES( 'Database',    80, 'KK', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Database',    95, 'AA', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Database',    80, 'LL', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Programming', 85, 'KK', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Programming', 75, 'LL', datetime('now'), NULL );
COMMIT;
```

  datetime('now') は現在日時の取得．DATETIME型は、YYYY-MM-DD HH:MM:SS形式．

- SQL 問い合わせの発行と評価結果の確認 (Issue SQL queries and inspect the results)

```
SELECT * FROM score_records;
SELECT * FROM score_records WHERE name = 'Database';
SELECT score FROM score_records WHERE name = 'Database';
```

- SQL を用いたテーブル定義と一貫性制約の記述 (Table defintion and integrity constraint specification using SQL)
  リレーショナル・スキーマ (relational schema): **order_records(id, year, month, day, customer_name, product_name, unit_price, qty)**

  SQL プログラム:

```
CREATE TABLE order_records (
    id            INTEGER  PRIMARY KEY AUTOINCREMENT NOT NULL,
    year          INTEGER  NOT NULL CHECK ( year > 2008 ),
    month         INTEGER  NOT NULL CHECK ( month >= 1 AND month <= 12 ),
    day           INTEGER  NOT NULL CHECK ( day >= 1 AND day <= 31 ),
    customer_name TEXT  NOT NULL,
    product_name  TEXT  NOT NULL,
    unit_price    REAL     NOT NULL CHECK ( unit_price > 0 ),
    qty           INTEGER  NOT NULL DEFAULT 1 CHECK ( qty > 0 ),
    created_at    DATETIME NOT NULL,
    updated_at    DATETIME,
    CHECK ( ( unit_price * qty ) < 200000 ) );
```

- SQL を用いたテーブルへの行の挿入 (Insert rows into a table using SQL)

```
BEGIN TRANSACTION;
INSERT INTO order_records VALUES( 1, 2009, 10, 26,  'kaneko', 'orange A', 1.2, 10, datetime('now'), NULL );
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, qty, created_at) VALUES( 2009, 10, 26,  'miy
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, qty, created_at) VALUES( 2009, 10, 27,  'kan
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, created_at) VALUES( 2009, 10, 28,  'miyamoto
COMMIT;
```

- SQL 問い合わせの発行と評価結果の確認 (Issue SQL queries and inspect the results)

```
SELECT * FROM order_records;
SELECT * FROM order_records WHERE day = 26;
SELECT * FROM order_records WHERE customer_name = 'kaneko';
SELECT * FROM order_records WHERE unit_price > 2;
```

- 一貫性制約に違反する更新ができないことの確認 (Integrity constraint violation is not permitted when database update)

演習を行うために必要になる機能や文法

SQL はリレーショナルデータベース言語の標準である. ここでは SQLite が持つ **SQL の機能**のうち今回の演習に関係する部分を紹介する。

SQLite の SQL の説明は http://www.hwaci.com/sw/sqlite/lang.html (English Web Page) にある.

1. **SQLite のデータ型**

   データ型の種類は, データベース管理システムごとに違う. SQLite では, 扱えるデータ型として次の 5 種類がある. 詳しい説明は http://www.sqlite.org/datatype3.html にある.

   ※ SQlite のデータ型と, SQL の標準が定めるデータ型の定義は異なる. 大胆にまとめると, SQlite のデータ型の方がより大きな範囲のデータを扱える.

   - **NULL**: 空値 (a NULL value)
   - **INTEGER**: 符号付きの整数 (signed integer) ※ SQLite では BIGINT と書いても INTEGER と書いても同じ「8バイトの整数」という意味
   - **REAL**: 浮動小数点値 (floating point value)
   - **TEXT**: 文字列 (text string)
   - **DATETIME**: 日時 ※ SQLite では DATETIME と書いても TEXT と書いても同じ「可変長文字列」の意味. だけど使い分けるべき(その方が分かりやすい)
   - **BLOB**: バイナリ・ラージ・オブジェクト (Binary Large Object). 入力がそのままの形で格納される (stored exactly as it was input). ※ SQLite では LARGEBLOB と書いても BLOB と書いても同じ「長大なバイナリ・ラージ・オブジェクト」という意味

2. **SQL の文字列定数**

   SQL の規格では, 文字列定数はシングルクオーテーションマーク「'」で囲むことになっている.

3. **SQL テーブル定義文 (create-table-statement) の例**

   CREATE TABLE <table-name> **(**<column-name> <type-name> [<column constraint> ...], ...**);**

   ※「 [<column constraint> ...]」は省略可能であることに注意

4. **SQL 列制約 (column-constraint) の例**

   create-table-statement の中に含める一貫性制約やデフォルト値の指定

   - **PRIMARY KEY** ・・・ 主キー制約
   - **NOT NULL** ・・・ 非空制約
   - **UNIQUE** ・・・ 一意制約
   - **CHECK (**<expression>**)** ・・・ 更新時にチェックされる式 (expression) ※ SQLite 固有の機能
   - **REFERENCES** <foreign-table> **(**<column-name>, ...**)** ・・・ 参照整合性制約
   - **DEFAULT (**<expression>**)** ・・・ デフォルト値 (default value) の指定
   - **AUTOINCREMENT** ・・・ 自動インクリメント (auto increment)

5. **SQL テーブル制約 (table-constraint) の例**

   create-table-statement の中に含める一貫性制約のうち**複数の属性に関わるもの**は table-constraint の形で記述することになる.

   - **PRIMARY KEY(**<indexed-column>, ...**)**
   - **UNIQUE(**<indexed-column>, ...**)**
   - **CHECK(**<expression>**)**

6. **SQL 挿入文 (insert statement) の例**

   テーブルへの行の挿入

   - **INSERT INTO** <table-name> **VALUES (**<expression>, ...**);**
   - **INSERT INTO** <table-name> **(**<column-name>, ...**) VALUES (** <expression>, ...**);**

7. **BEGIN TRANSACTION, COMMIT, ROLLBACK**

   SQL 挿入文 (insert statement) などでのデータベース更新を行うときは, 最初に「BEGIN TRANSACTION;」を実行する. データベース更新が終わったら「COMMIT;」または「ROLLBACK;」を実行する.

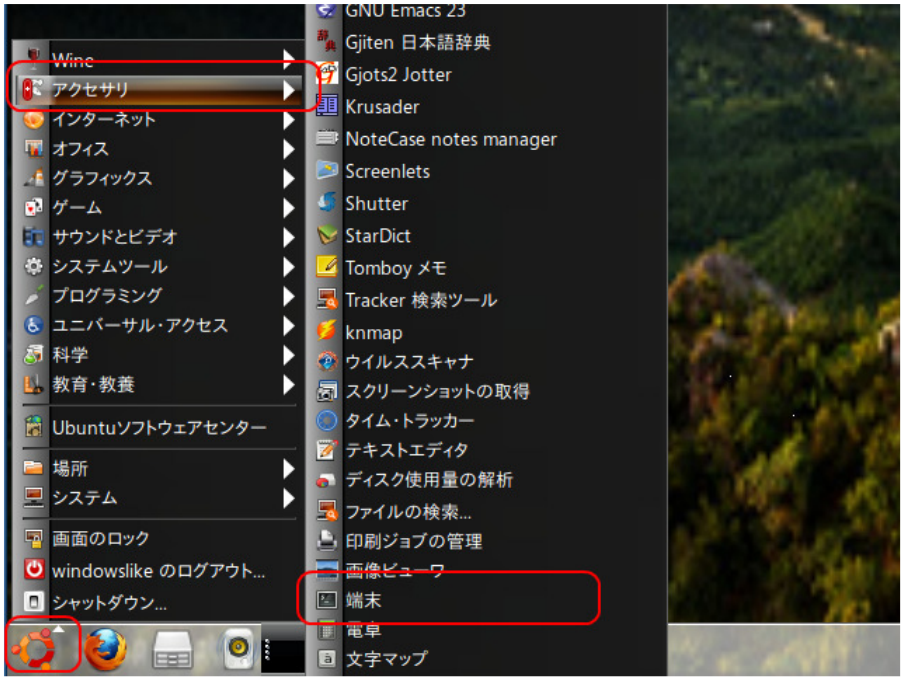
   - COMMIT: ・・・「BEGIN TRANSACTION;」以降の全てのデータベース更新操作を**確定**したいとき
   - ROLLBACK: ・・・「BEGIN TRANSACTION;」以降の全てのデータベース更新操作を**破棄**したいとき

8. **SQL の SELECT, FROM, WHERE の例**
   SQL での問い合わせ には SELECT, FROM, WHERE 句が多用される.
   - **SELECT * FROM** <table-name>;
     table-name で指定したテーブルの**全ての行**を表示
   - **SELECT * FROM** <table-name> **WHERE** <expression>;
     table-name で指定したテーブルのうち **expression で指定した条件を満足する行**だけを抽出して表示

SQLite バージョン 3 の起動と終了 (Start and end SQLite version 3)

1. 端末(あるいは Windows のコマンドプロンプト)を開く (Open e a Linux Terminal or the Windows command prompt
   ■ **Ubuntu の場合**

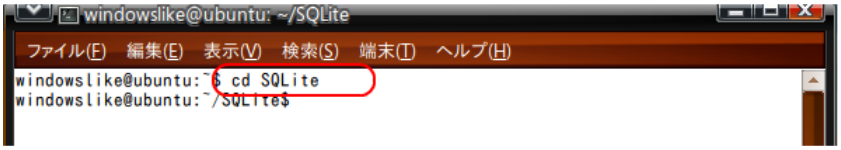
   端末を開きたいので,「**アクセサリ**」→「**端末**」と操作する



2. **SQLite データベース・ディレクトリに移る**. (Move to the database directory).
   ■ **Ubuntu の場合**

   SQLite データベース・ディレクトリ **SQLite** に移る.

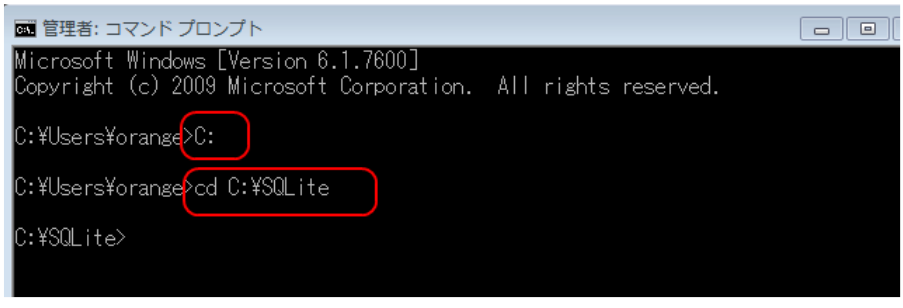
```
cd SQLite
```



   ■ **Windows の場合**

   SQLite データベース・ディレクトリ **C:¥SQLite** に移る.

```
C:
cd C:¥SQLite
```

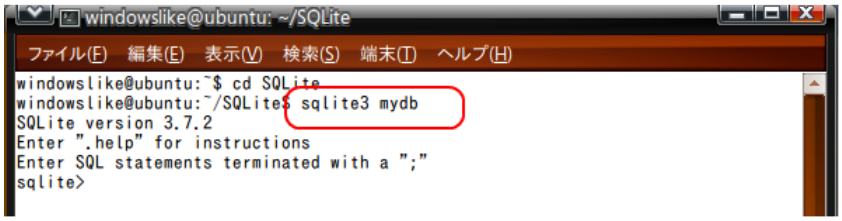


3. **SQLite の起動** (Start the SQLite).
   ■ **Ubuntu の場合**

   このとき, **データベース論理名**として **mydb** を指定する. (The logical database name is 'mydb').

※ データベース論理名はなんでも良いが、アルファベットのみを使うのが良い.

```
sqlite3 mydb
```

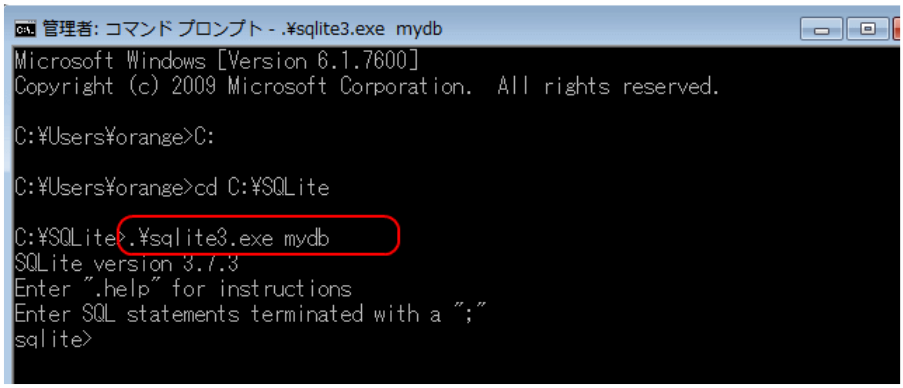


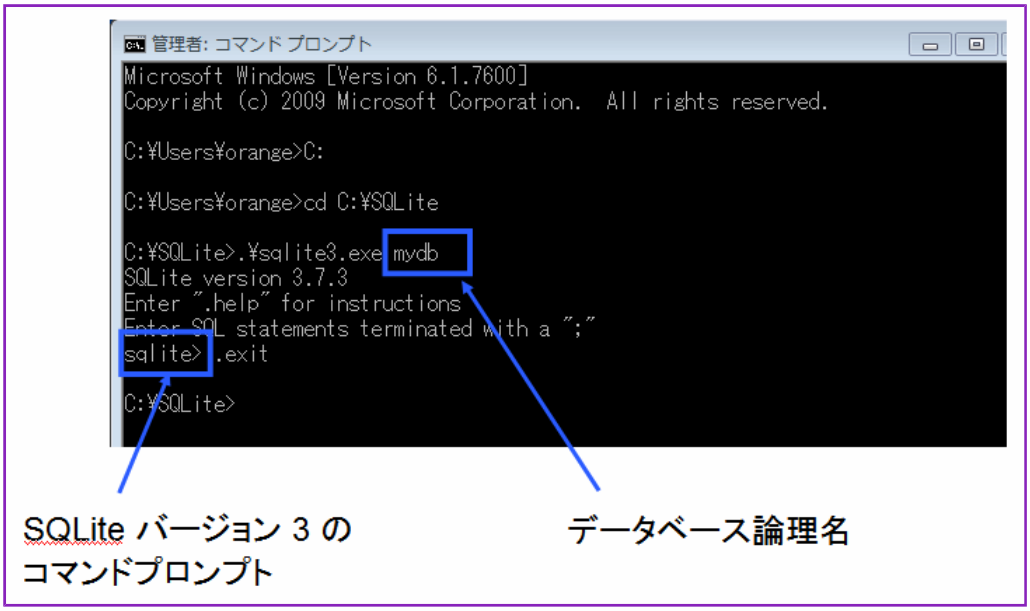
■ **Windows の場合**

このとき, **データベース論理名**として **mydb** を指定する. (The logical database name is 'mydb').

※ データベース論理名はなんでも良いが、アルファベットのみを使うのが良い.

```
.¥sqlite3.exe mydb
```





4. **ヘルプの表示** (display the help)
   「**.help**」で, ヘルプが表示されます.

5. **SQLite の終了** (End SQLite)
   「**.exit**」で終了.

SQLite データベースの新規作成 (Create a new SQLite database), SQL を用いたテーブル定義と一貫性制約の記述 (Table defintion and integrity constrant specification using SQL)

前準備: **SQLite3 のインストールを終えていること**

SQL を用いて, **score_records テーブルを定義**し, **一貫性制約を記述**する. (Define 'score_records' table and specify integrity constrants of the table using SQL)

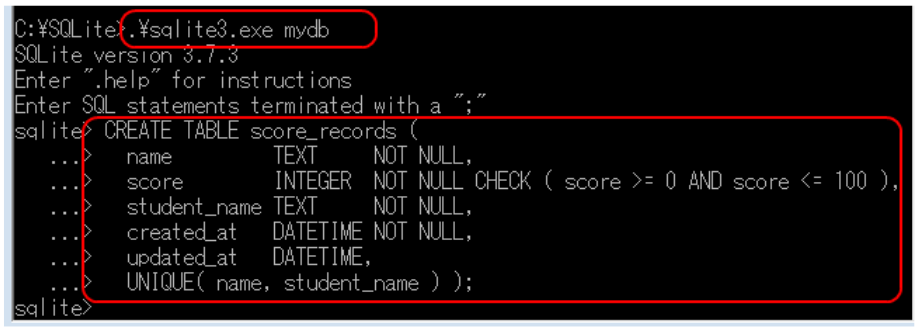
リレーショナル・スキーマ (relational schema): **score_records(name, score, student_name, created_at, updated_at)**

```
CREATE TABLE score_records (
    name         TEXT     NOT NULL,
    score        INTEGER  NOT NULL CHECK ( score >= 0 AND score <=100 ),
    student_name TEXT     NOT NULL,
    created_at   DATETIME NOT NULL,
    updated_at   DATETIME,
    UNIQUE (name, student_name) );
```

TEXT は「文字列」という意味になるので，数値に関する演算を実行することはできません．

SQL のキーワード(CREATE, TABLE, TEXT, NOT, NULL, INTEGER, DATETIME など)は，大文字でも小文字でもよいというルールがあります．但し，大文字と小文字を混ぜると混乱するので，この演習では大文字を使うことにします．

テーブルの名前や，列の名前も大文字でも小文字でもよいというルールがあります．これも，大文字や小文字を混ぜると混乱するので，この演習では小文字を使うことにします．空白文字は使えないので，単語と単語の区切りに「_」を使っています．



このとき **C:¥SQLite** に，データベースファイル **mydb** ができる．(At the time, database file is generated)

※ SQLite では，データベースが始めて使うときに，**自動的にデータベースファイルが生成**される．データベースファイル名は，データベース論理名と同じになる．

※ データベースファイルが生成されるのは，テーブルを定義するなど，データベースの更新を行ったときなので，最初，sqlite3 を起動したとき，データベースファイルが無くてもあわてないこと．

SQL を用いたテーブルへの行の挿入 (Insert rows into a table using SQL)

次のような **score_records** テーブルを作る．(Construct table 'score_records)

```
name        | score | student_name | created_at | updated_at
-------------------------------------------------------------
Database    |   80  | KK           |            |
Database    |   95  | AA           |            |
Database    |   80  | LL           |            |
Programming |   85  | KK           |            |
Programming |   75  | LL           |            |
```

以下の手順で，SQL を用いて**score_records テーブルへの行の挿入**を行う (Insert rows into table 'score_records' using SQL)

挿入の前に **BEGIN TRANSACTION;** を実行し，一連の挿入が終わったら **COMMIT;** を実行する．(Issue "BEGIN TRANSACTION" before database update and "COMMIT" after database update).

datetime('now') は現在日時の取得．DATETIME型は、YYYY-MM-DD HH:MM:SS形式．

```
BEGIN TRANSACTION;
INSERT INTO score_records VALUES( 'Database',    80, 'KK', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Database',    95, 'AA', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Database',    80, 'LL', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Programming', 85, 'KK', datetime('now'), NULL );
INSERT INTO score_records VALUES( 'Programming', 75, 'LL', datetime('now'), NULL );
COMMIT;
```

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO score_records VALUES( 'Database',    80, 'KK', datetime('now
'), NULL );
sqlite> INSERT INTO score_records VALUES( 'Database',    95, 'AA', datetime('now
'), NULL );
sqlite> INSERT INTO score_records VALUES( 'Database',    80, 'LL', datetime('now
'), NULL );
sqlite> INSERT INTO score_records VALUES( 'Programming', 85, 'KK', datetime('now
'), NULL );
sqlite> INSERT INTO score_records VALUES( 'Programming', 75, 'LL', datetime('now
'), NULL );
sqlite> COMMIT;
sqlite>
```

この挿入では，属性の値を，テーブル定義の順に全て並べる (List all attribute values. The order is the same as its table definition)

SQL 問い合わせの発行と評価結果の確認 (Issue SQL queries and inspect the results)

SQL 問い合わせの詳細については，別の Web ページで説明する．ここでは，テーブルの中身を確認して欲しい．

テーブルの全ての行の表示 (List all rows of a table)

```
SELECT * FROM score_records;
```

```
sqlite> SELECT * FROM score_records;
Database|80|KK|2009-12-08 15:58:42|
Database|95|AA|2009-12-08 15:58:53|
Database|80|LL|2009-12-08 15:59:04|
Programming|85|KK|2009-12-08 15:59:22|
Programming|75|LL|2009-12-08 15:59:34|
sqlite>
```

条件を満足する行のみの表示 (List the rows which satisfy a given condition)

```
SELECT * FROM score_records WHERE name = 'Database';
```

```
sqlite> SELECT * FROM score_records WHERE name = 'Database';
Database|95|AA|2009-12-08 15:58:53|
Database|80|KK|2009-12-08 15:58:42|
Database|80|LL|2009-12-08 15:59:04|
sqlite>
```

```
SELECT score FROM score_records WHERE name = 'Database';
```

```
sqlite> SELECT score FROM score_records WHERE name = 'Database';
95
80
80
sqlite>
```

SQL を用いたテーブル定義と一貫性制約の記述 (Table defintion and integrity constrant specification using SQL)

SQL を用いて，**order_records テーブルを定義**し，**一貫性制約を記述**する．(Define 'order_records' table and specify integrity constrants of the table using SQL)

リレーショナル・スキーマ (relational schema): **order_records(id, year, month, day, customer_name, product_name, unit_price, qty)**

```
CREATE TABLE order_records (
    id            INTEGER  PRIMARY KEY AUTOINCREMENT NOT NULL,
    year          INTEGER  NOT NULL CHECK ( year > 2008 ),
    month         INTEGER  NOT NULL CHECK ( month >= 1 AND month <= 12 ),
    day           INTEGER  NOT NULL CHECK ( day >= 1 AND day <= 31 ),
    customer_name TEXT     NOT NULL,
    product_name  TEXT     NOT NULL,
    unit_price    REAL     NOT NULL CHECK ( unit_price > 0 ),
    qty           INTEGER  NOT NULL DEFAULT 1 CHECK ( qty > 0 ),
    created_at    DATETIME NOT NULL,
    updated_at    DATETIME,
    CHECK ( ( unit_price * qty ) < 200000 ) );
```

```
sqlite> CREATE TABLE order_records (
   ...>   id           INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
   ...>   year         INTEGER NOT NULL CHECK ( year > 2008 ),
   ...>   month        INTEGER NOT NULL CHECK ( month >= 1 AND month <= 12 ),
   ...>   day          INTEGER NOT NULL CHECK ( day >= 1 AND day <= 31 ),
   ...>   customer_name TEXT   NOT NULL,
   ...>   product_name  TEXT   NOT NULL,
   ...>   unit_price   REAL    NOT NULL CHECK ( unit_price > 0 ),
   ...>   qty          INTEGER NOT NULL DEFAULT 1 CHECK ( qty > 0 ),
   ...>   created_at   DATETIME NOT NULL,
   ...>   updated_at   DATETIME,
   ...>   CHECK ( ( unit_price * qty ) < 200000 ) );
sqlite>
```

SQL を用いたテーブルへの行の挿入 (Insert rows into a table using SQL)

次のような order_records テーブルを作る. (Construct table 'order_records)

id	year	month	day	customer name	product name	unit price	qty	created at	updated at
1	2009	10	26	kaneko	orange A	1.2	10		
2	2009	10	26	miyamoto	Apple M	2.5	2		
3	2009	10	27	kaneko	orange B	1.2	8		
4	2009	19	28	miyamoto	Apple L	3	1		

※各行が挿入された日時 ※ 各行の最終更新日時

以下の手順で, SQL を用いて**order records テーブルへの行の挿入**を行う (Insert rows into table 'order_records' using SQL)

挿入の前に **BEGIN TRANSACTION;** を実行し, 一連の挿入が終わったら **COMMIT;** を実行する. (Issue "BEGIN TRANSACTION" before database update and "COMMIT" after database update).

```
BEGIN TRANSACTION;
INSERT INTO order_records VALUES( 1, 2009, 10, 26,  'kaneko', 'orange A', 1.2, 10, datetime('now'), NULL );
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, qty, created_at) VALUES( 2009, 10, 26,  'miyamoto', 'Apple
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, qty, created_at) VALUES( 2009, 10, 27,  'kaneko',    'orange
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, created_at) VALUES( 2009, 10, 28,  'miyamoto',    'Apple L',
COMMIT;
```

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO order_records VALUES( 1, 2009, 10, 26, 'kaneko', 'orange A',
 1.2, 10, datetime('now'), NULL );
sqlite> INSERT INTO order_records (year, month, day, customer_name, product_name
, unit_price, qty, created_at) VALUES( 2009, 10, 26, 'miyamoto', 'Apple M', 2.5,
 2, datetime('now') );
sqlite> INSERT INTO order_records (year, month, day, customer_name, product_name
, unit_price, qty, created_at) VALUES( 2009, 10, 27, 'kaneko', 'orange B', 1.2,
8, datetime('now') );
sqlite> INSERT INTO order_records (year, month, day, customer_name, product_name
, unit_price, created_at) VALUES( 2009, 10, 28, 'miyamoto', 'Apple L', 3, dateti
me('now') );
sqlite> COMMIT;
sqlite>
```

INSERT INTO には 2つの方法がある. (Two styles of "INSERT INTO")

■ 属性の値を, テーブル定義の順に全て並べる (List all attribute values. The order is the same as its table definition)

```
INSERT INTO order_records VALUES( 1, 2009, 10, 26,  'kaneko', 'orange A', 1.2, 10, datetime('now'), NULL );
```

■ 属性の値の並び方を, 属性名を使って明示的に指定する (Specify the order of attribute values using attribute name list)

このとき, 属性値を省略すると, テーブル定義のときに指定されたデフォルト値が使われる (defaults values are used)

```
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, qty, created_at) VALUES( 2009, 10, 26,  'miyamoto', 'Apple
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, qty, created_at) VALUES( 2009, 10, 27,  'kaneko',    'orange
INSERT INTO order_records (year, month, day, customer_name, product_name, unit_price, created_at) VALUES( 2009, 10, 28,  'miyamoto',    'Apple L',
```

SQL 問い合わせの発行と評価結果の確認 (Issue SQL queries and inspect the results)

SQL 問い合わせの詳細については, 別の Web ページで説明する. ここでは, テーブルの中身を確認して欲しい.

テーブルの全ての行の表示 (List all rows of a table)

```
SELECT * FROM order_records;
```

```
sqlite> SELECT * FROM order_records;
1|2009|10|26|kaneko|orange A|1.2|10|2009-12-08 16:42:48|
2|2009|10|26|miyamoto|Apple M|2.5|2|2009-12-08 16:42:59|
3|2009|10|27|kaneko|orange B|1.2|8|2009-12-08 16:43:35|
4|2009|10|28|miyamoto|Apple L|3.0|1|2009-12-08 16:44:33|
sqlite>
```

条件を満足する行のみの表示 (List the rows which satisfy a given condition)

```
SELECT * FROM order_records WHERE day = 26;
```

```
sqlite> SELECT * FROM order_records WHERE day = 26;
1|2009|10|26|kaneko|orange A|1.2|10
2|2009|10|26|miyamoto|Apple M|2.5|2
sqlite>
```

```
SELECT * FROM order_records WHERE customer_name = 'kaneko';
```

```
sqlite> SELECT * FROM order_records WHERE customer_name = 'kaneko';
1|2009|10|26|kaneko|orange A|1.2|10
3|2009|10|27|kaneko|orange B|1.2|8
sqlite>
```

```
SELECT * FROM order_records WHERE unit_price > 2;
```

```
sqlite> SELECT * FROM order_records WHERE unit_price > 2;
2|2009|10|26|miyamoto|Apple M|2.5|2
4|2009|10|28|miyamoto|Apple L|3.0|1
sqlite>
```

一貫性制約に違反する更新ができないことの確認 (Integrity constraint violation is not permitted when database update)

ここでは，一貫性制約に違反するような更新を試みる．データベース管理システムソフトウエアが一貫性を維持するので，一貫性制約に違反するような更新はできない．

一意制約 (UNIQUE)

```
BEGIN TRANSACTION;
INSERT INTO score_records VALUES( 'Database', 90, 'KK', datetime('now'), NULL );
ROLLBACK;
```

※ すでに「'Database', 80, 'KK'」という行がある．一意制約「UNIQUE(name, student_name)」に違反．

```
sqlite> SELECT * FROM score_records;
Database|80|KK|2009-12-10 04:29:40|
Database|95|AA|2009-12-10 04:29:49|
Database|80|LL|2009-12-10 04:29:54|
Programming|85|KK|2009-12-10 04:30:06|
Programming|75|LL|2009-12-10 04:30:12|
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO score_records VALUES( 'Database', 90, 'KK', datetime('now'),
 NULL );
Error: constraint failed
sqlite> ROLLBACK;
sqlite>
```

主キー制約 (PRIMARY KEY)

```
BEGIN TRANSACTION;
INSERT INTO order_records VALUES( 3, 2009, 10, 29, 'kaneko', 'orange C', 1.1, 6, datetime('now'), NULL );
ROLLBACK;
```

※ すでに属性 id には 3 という値がある．主キー制約「PRIMARY KEY」に違反．

```
sqlite> SELECT * FROM order_records;
1|2009|10|26|kaneko|orange A|1.2|10|2009-12-10 04:32:27|
2|2009|10|26|miyamoto|Apple M|2.5|2|2009-12-10 04:32:27|
3|2009|10|27|kaneko|orange B|1.2|8|2009-12-10 04:32:27|
4|2009|10|28|miyamoto|Apple L|3.0|1|2009-12-10 04:32:27|
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO order_records VALUES( 3, 2009, 10, 29, 'kaneko', 'orange C',
 1.1, 6, date('now'), NULL );
Error: constraint failed
sqlite> ROLLBACK;
sqlite>
```

非空制約 (NOT NULL)

```
BEGIN TRANSACTION;
INSERT INTO order_records VALUES( 5, 2009, 10, 30, NULL, 'melon', 10, 3, datetime('now'), NULL );
ROLLBACK;
```

※ 非空制約「NOT NULL」．属性 customer_name には NULL を入れることができない．

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO order_records VALUES( 5, 2009, 10, 30, NULL, 'melon', 10, 3,
 datetime('now'), NULL );
Error: constraint failed
sqlite> ROLLBACK;
sqlite>
```

その他の一貫性制約

一貫性制約に違反する例

```
BEGIN TRANSACTION;
INSERT INTO order_records VALUES( 6, 1009, 10, 30, kaneko, 'melon', 10, 3, datetime('now'), NULL );
ROLLBACK;
```

※ 制約「CHECK (year > 2008)」に違反

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO order_records VALUES( 6, 1009, 10, 30, 'kaneko', 'melon', 10
, 3 );
SQL error: constraint failed
sqlite> ROLLBACK;
```

一貫性制約に違反する例

```
BEGIN TRANSACTION;
INSERT INTO order_records VALUES( 7, 2009, 10, 31, kaneko, 'strawberry', 4.6, 100000, datetime('now'), NULL );
ROLLBACK;
```

※ 制約「CHECK ((unit_price * qty) < 200000))」に違反

```
sqlite> BEGIN TRANSACTION;
sqlite> INSERT INTO order_records VALUES( 7, 2009, 10, 31, 'kaneko', 'strawberry
', 4.6, 100000, datetime('now'), NULL );
Error: constraint failed
sqlite> ROLLBACK;
sqlite>
```

演習問題と解答例

次の問いに答えよ. その後, 下記の解答例を確認せよ. Answer the following questions. Then, inspect answers described below.

問い (Questions)

1. SQLite の SQL を使い, 下記のリレーションスキーマに合致するテーブルを定義しなさい (Define tables below using SQL)

   (1) employees(id, employeename, street, city)

   (2) companies(id, companyname, city)

   (3) works(employeename, companyname)

2. SQLite を使い, 下記の SQL を評価させなさい (Evaluate the following SQL)

```
CREATE TABLE SS (
    name         TEXT     NOT NULL,
    score        INTEGER  NOT NULL CHECK ( score >= 0 AND score <=100 ),
    student_name TEXT     NOT NULL,
    UNIQUE (name, student_name) );
```

   その後, SQLite の SQL を使い次のテーブルを作りなさい (Create the following table using SQLite)

```
table-name: SS
name       | score | student_name
-----------------------------------------------
Database   |   80  | KK
Database   |   95  | AA
Database   |   80  | LL
Programming|   85  | KK
Programming|   75  | LL
```

3. 次の SQL を評価させるとエラーが発生する. 確認しなさい (Examine runtime errors of the following SQL)

```
(1)
        create table AA (
                id  INTEGER PRIMARY KEY,
                amount INTEGER NOT NULL CHECK ( amount > 1000 ) );
        BEGIN TRANSACTION;
        insert into AA values ( 1, 100 );
        ROLLBACK;
(2)
        create table BB (
                id  INTEGER PRIMARY KEY,
                name TEXT NOT NULL,
                course TEXT NOT NULL );
        BEGIN TRANSACTION;
        insert into BB values ( 1, 'A', NULL );
        ROLLBACK;
```

解答例 (Answers)

1.

   1. employees(id, employeename, street, city)

```
CREATE TABLE employees (
  id INTEGER PRIMARY KEY NOT NULL,
  employeename TEXT,
  street TEXT,
  city TEXT );
```

   2. company(id, companyname, city)

```
CREATE TABLE companies (
  id INTEGER PRIMARY KEY NOT NULL,
  companyname TEXT,
  city TEXT );
```

   3. works(employeename, companyname)

```
CREATE TABLE works (
  employeename TEXT,
  companyname TEXT );
```

2.

```
INSERT INTO SS VALUES('Database', 80, 'KK');
INSERT INTO SS VALUES('Database', 95, 'AA');
INSERT INTO SS VALUES('Database', 80, 'LL');
INSERT INTO SS VALUES('Database', 85, 'KK');
INSERT INTO SS VALUES('Database', 75, 'LL');
```