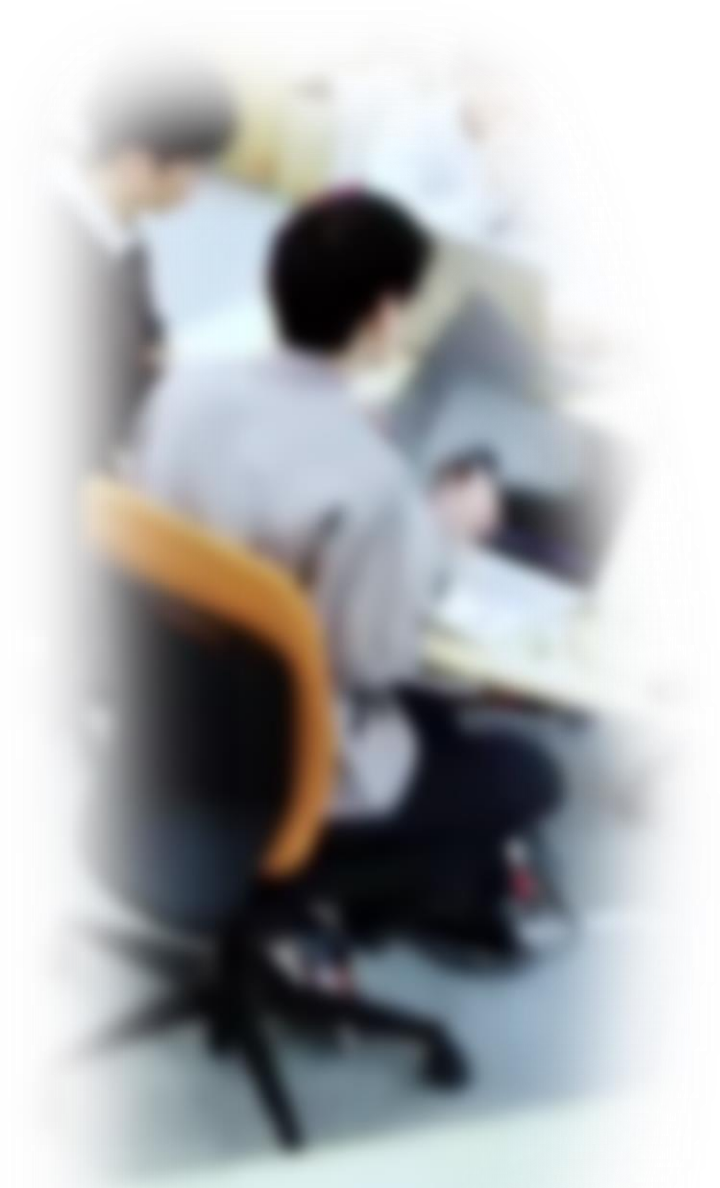
16. インデックス、セキュリティ、データベースの歴史と展望、データベースのバリエーション

URL: https://www.kkaneko.jp/de/ds/index.html

金子邦彦

謝辞：この資料では「いらすとや」のイラストを使用しています



アウトライン

1. イントロダクション
2. インデックス
3. セキュリティ
4. データベースの歴史と展望
5. データベースシステムとその周辺技術

SQLFiddle のサイトにアクセス

Webブラウザを使用

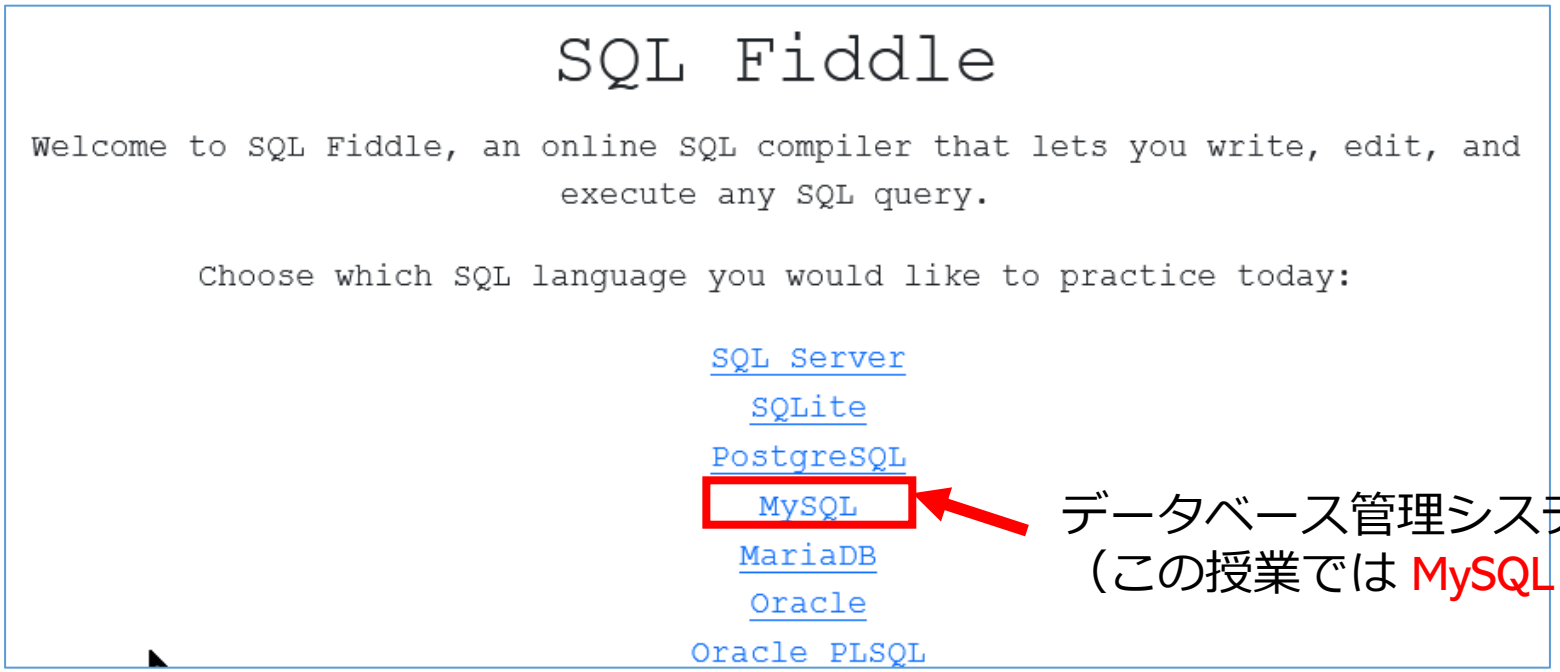
1. ウェブブラウザを開く
2. アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

3. **MySQL を選ぶ**

URLが分からないときは、Googleなどの**検索エンジン**を利用。
「**SQLFiddle**」と**検索**し、表示された結果からSQLFiddleの
ウェブサイトをクリック。

SQLFiddle でのデータベース管理システムの選択



データベース管理システムの選択
（この授業では MySQL を使用）

SQLFiddle の画面

上のパネル: SQLの入力（複数可能）

- **テーブル定義** CREATE TABLE
- **データの追加** INSERT INTO
- **SQL問い合わせ**。SELECT, FROM, WHERE など

16-1. イントロダクション

リレーショナルデータベースの仕組み

- データを**テーブル**と呼ばれる**表形式で保存**
- **テーブル間**は**関連**で結ばれる。複雑な構造を持ったデータを効率的に管理することを可能に。

商品

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

関連

購入

購入者	商品番号
X	1
X	3
Y	2

商品テーブルと購入テーブル

商品

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

関連

購入

購入者	商品番号
X	1
X	3
Y	2

Xさんは、**1** の**みかん**と,
3 の**メロン**を買った
Yさんは、 **2** の**りんご**を買った

購入テーブルの情報

商品テーブルの情報

リレーショナルデータベースの重要性

1. **データの整合性**: リレーショナルデータベースは、**データの整合性を保持するための機能**を有する。これにより、誤ったデータや矛盾したデータが保存されるのを防ぐことができる。
2. **柔軟な問い合わせ（クエリ）能力**: リレーショナルデータベースの**SQL**（Structured Query Language）の使用により、**複雑な検索やデータの抽出**が可能になる。
3. **トランザクション**の機能: 一連の操作全体を一つの単位として取り扱うことができる機能。これにより、**データの一貫性と信頼性が向上**する。
4. **セキュリティ**: **アクセス権限の設定**などにより、セキュリティを確保。

データの安全な保管、効率的なデータ検索・操作、ビジネスや研究の意思決定をサポート。

SQL 理解のための前提知識

○ テーブル

データを**テーブル**と呼ばれる**表形式で保存**

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

購入者	商品番号
X	1
X	3
Y	2

○ 問い合わせ（クエリ）

- **問い合わせ（クエリ）**は、**データベース**から**必要なデータを検索、加工するための指令**
- SELECT, FROM, WHERE など、**多様**なコマンドが存在。
- **結合、集計、ソート、副問い合わせ**など、高度な操作も可能

SQL によるテーブル定義

- **テーブル名**：**商品**
- **属性名**：**ID、商品名、単価**
- **属性のデータ型**：**数値、テキスト、数値**
- データの整合性を保つための**制約**：**主キー制約**

```
CREATE TABLE 商品 (
ID INTEGER PRIMAY KEY,
商品名 TEXT,
単価 INTEGER);
```

データ追加のSQL

商品

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

```
INSERT INTO 商品 VALUES(1, 'みかん', 50);
INSERT INTO 商品 VALUES(2, 'りんご', 100);
INSERT INTO 商品 VALUES(3, 'メロン', 500);
```

主キー

- **主キー**は、**テーブルの各行を識別する**ためのキー

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

主キー

PRIMARY KEY

PRIMARY KEY はテーブル定義時に使用し、「**主キー制約**」を示す

```
CREATE TABLE テーブル名 (
   列名1 データ型 PRIMARY KEY,
   列名2 データ型,
   …
);
```

SQL の書き方

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500

主キー

```
CREATE TABLE 商品 (
   ID INTEGER PRIMAY KEY,
   商品名 TEXT,
   単価 INTEGER);
```

演習１．テーブル定義とデータの追加、主キー制約

【トピックス】

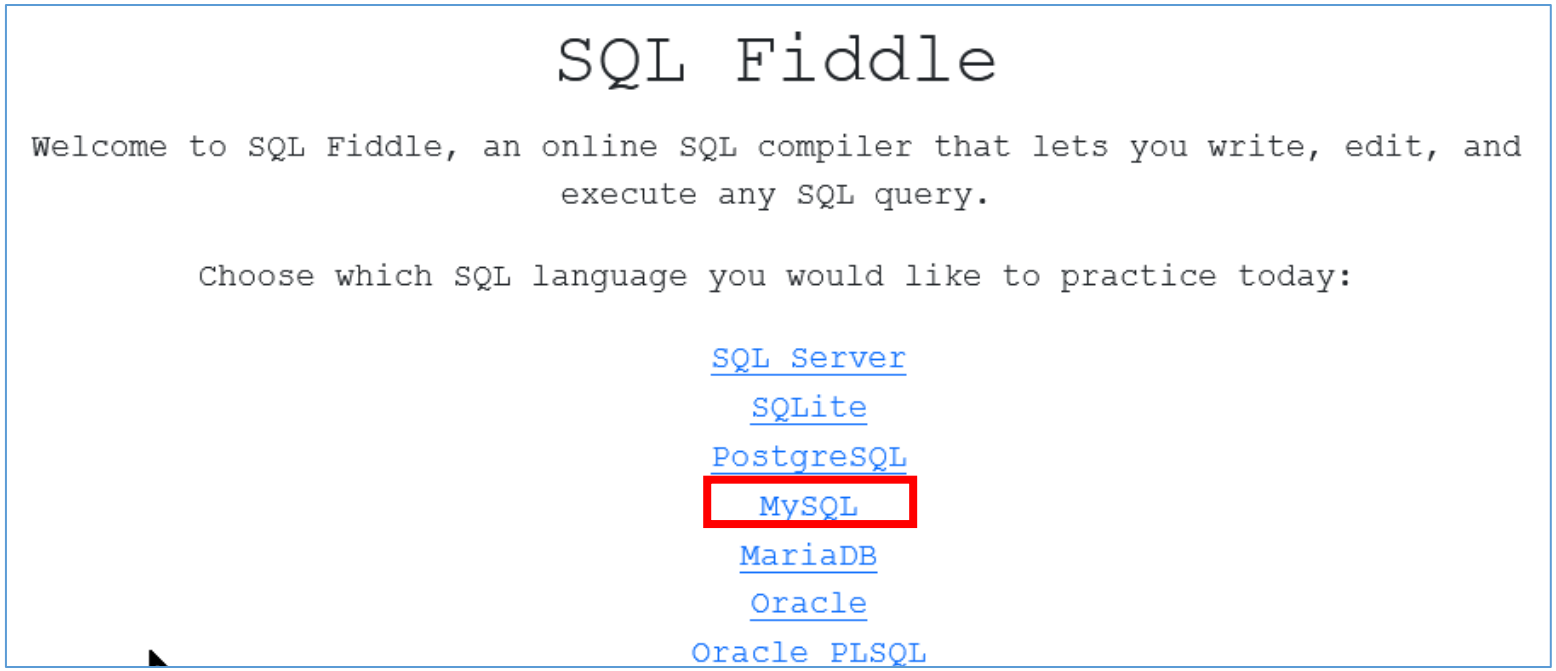
1. **SQL によるテーブル定義**
2. **主キー制約 PRIMARY KEY**
3. **SQL によるデータの追加**
4. **問い合わせ（クエリ）による確認**

Webブラウザを使用

① アドレスバーにSQLFiddleのURLを入力

http://sqlfiddle.com/

② 「**MySQL**」を選択



② **上のパネル**に、**テーブル定義**と**データの追加**と**問い合わせ（クエリ）**を行う SQL を入れる。

元からある SQL は不要なので消す

```
CREATE TABLE 商品 (
     ID INTEGER PRIMARY KEY,
     商品名 TEXT,
     単価 INTEGER);
INSERT INTO 商品 VALUES(1, 'みかん', 50);
INSERT INTO 商品 VALUES(2, 'りんご', 100);
INSERT INTO 商品 VALUES(3, 'メロン', 500);

select * FROM 商品;
```

④ 「**Execute**」をクリック

SQL 文が**実行**され、結果が表示される。

⑤ 下側のウインドウで、**結果を確認**。

```
+----+------------+--------+
| ID | 商品名     | 単価   |
+----+------------+--------+
| 1  | みかん     | 50     |
| 2  | りんご     | 100    |
| 3  | メロン     | 500    |
+----+------------+--------+
```

16-2. インデックス

インデックス

- **データベースの検索性能向上のためのもの**

商品名
いちご
すいか
みかん
りんご
メロン

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500
4	いちご	200
5	すいか	400

属性「商品名」の
インデックス

インデックスの機能

- **データ検索時**に、全データをスキャンするのでなく、**インデックスのみのスキャン**を行うことで、効率的な検索を可能にする

商品名「みかん」で検索のときは、インデックスのスキャン

商品名
いちご
すいか
みかん
りんご
メロン

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500
4	いちご	200
5	すいか	400

単価で検索のときは、全データのスキャン

属性「商品名」のインデックス

インデックスにより検索性能が向上する仕組み

インデックスの種類

B-tree ： 20以上30以下のような範囲検索向き

ハッシュ：「みかん」で検索のような完全一致検索向き

商品名
いちご
すいか
みかん
りんご
メロン

属性「商品名」の
インデックス

インデックスは、
単純にデータを並べたものではなく、
検索性能が向上する仕組みが付いている

インデックスの作成方法

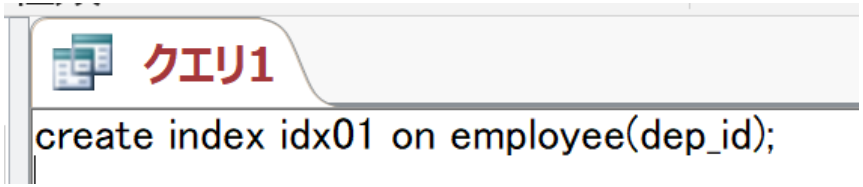
- SQL 文を用いて作成可能
- 作成時に、**インデックス名**、そして、インデックス化したい**テーブル名**と**属性名**を指定

MySQL

```
create index idx01 on employee(dep_id);
```

※ USING BTREE, USING HASH でインデックスの種類を指定可能

Access



※ Access では B-tree インデックスが利用可能

インデックスの特性

1. **メリット**（性能向上）と**デメリット**（性能低下）
2. **インデックスの有無**で、**SQLのプログラムは変化しない。**
   同じSQLを使うことができる
3. 通常、「**主キー**」の属性には**自動的にインデックスが作成される。**他の属性についてもインデックスを追加可能

メリットとデメリット

- 検索時に、頻繁に使用される属性についてのみインデックスを作成

商品名「**みかん**」で**検索**のときは、インデックスのスキャン

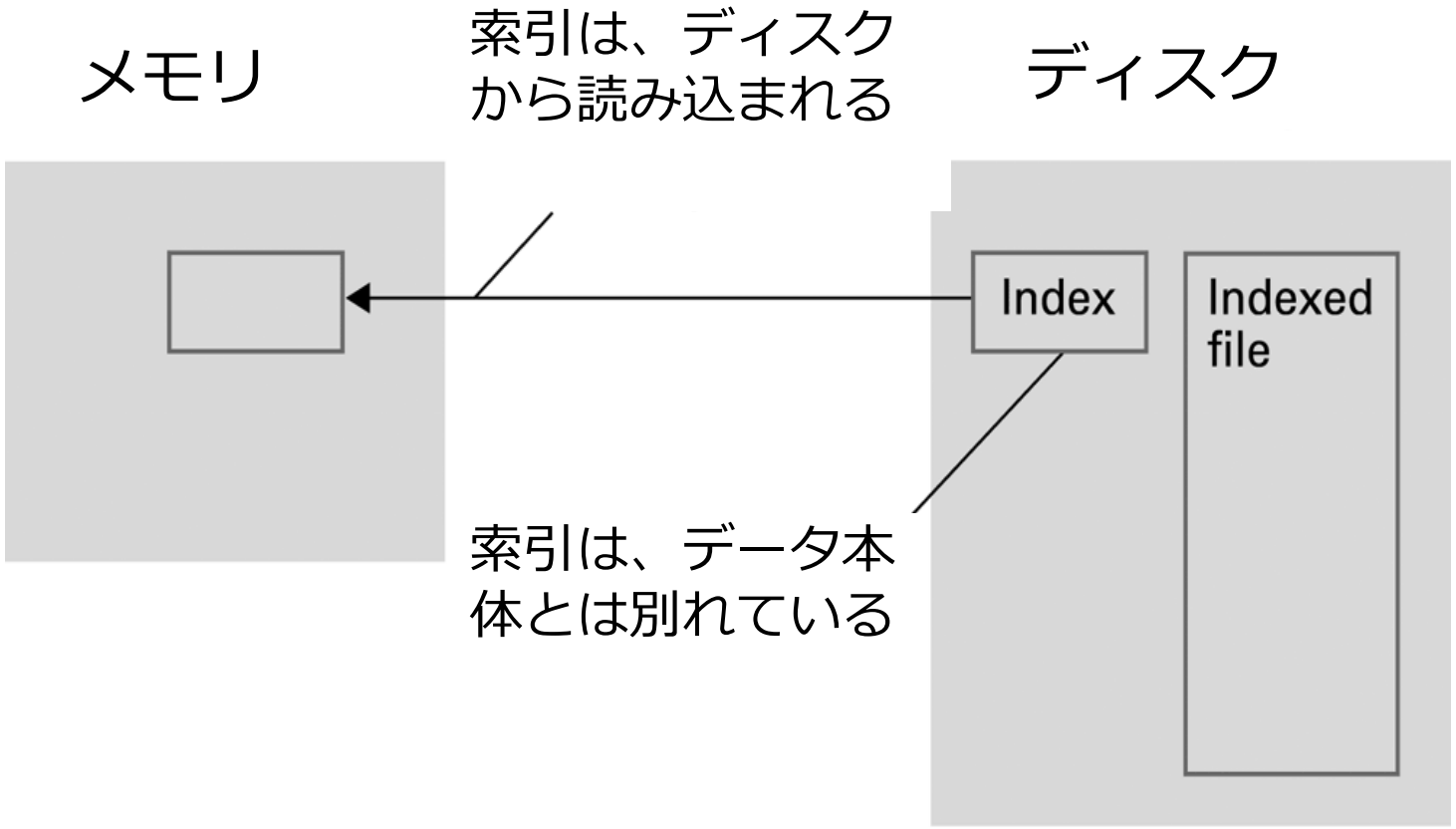
商品名
いちご
すいか
みかん
りんご
メロン

属性「商品名」のインデックス

ID	商品名	単価
1	みかん	50
2	りんご	100
3	メロン	500
4	いちご	200
5	すいか	400

単価で検索のときは、全データのスキャン

索引（インデックス）の仕組み



全体まとめ

- データの一元管理により、多様なデータを組み合わせた分析が可能になる。
- **オンライントランザクション**では、**リアルタイムのデータ処理**により、**オンラインでの情報共有**を行う。
- **データウェアハウス**は**「履歴データ」を重視**し、**データは一度格納されると削除、変更されることなく保存**される。
- 購入テーブル（属性は、ID、購入者、商品ID、数量、購入日時）では、「購入日時」の属性を用いて購入履歴を管理できるようになる。
- 商品テーブル（属性は、ID、名前、単価、改訂日時）では、「改訂日時」の属性を持ちいて価格の履歴を管理できるようになる。
- **SQL**では、「**DATETIME**」を使って**日時**や**履歴**を扱うことができる。
- **日時のデータ**は SQL では、「**'2024-01-05 09:00:00'**」のような形式で表される。
- MySQL や Access では、「**select now();**」により、現在の日時が表示される

16-4 データベースシステムの歴史と展望

データベースシステムの基礎

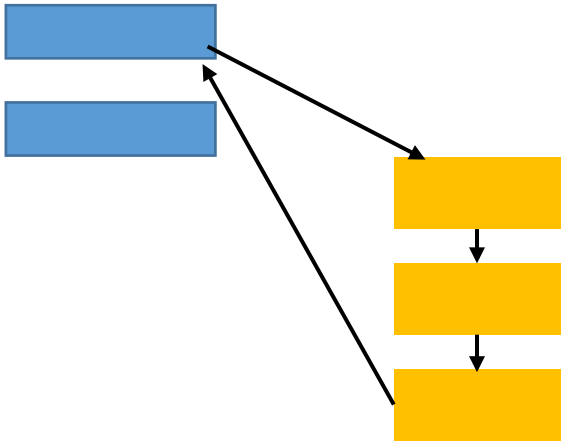
- **データの集積、管理、利用の効率化**: データベースはデータを効率的に整理、管理、利用するための仕組み。
- データは**テーブル形式**で整理
- **SQL**（Structured Query Language）を使用してデータの操作が行われます。
- データベースは**データの整合性を保つ仕組み**を提供。
- **複数のユーザー**が**同時にデータにアクセス**できる。
- **データの安全性と信頼性の向上**

リレーショナルデータベースシステムの歴史

- １９６９年（E.F. Codd at IBM research）リレーショナルデータベースの提唱
- １９７０年代前半から　実際のシステムが登場

従来のプログラミング言語や，他のデータベースシステムで出来なかったことができるようになり，大きく普及.

いまも，**リレーショナルデータベースは，データベースシステムの主流**



関係のあるレコード同士を
ポインタでつないでおく

デーブルは**ダイナミック**に
結合する

リレーショナル
データベースシステム**以前**

リレーショナル
データベースシステム

大きな変化．データ管理が簡単に

データベースシステムの進化

- **リレーショナルモデルの提案**

E.F. Coddによって1970年に提唱。その後のデータベースの基盤になった。

- **リレーショナルデータベースシステムの誕生**

その後、リレーショナルモデルを実現したリレーショナルデータベースシステムが登場

- **新しいデータベース技術の発展**

NoSQLデータベースなど、新しいデータベース技術が登場し、巨大なデータ処理や実時間処理が容易に。

データベース関連技術の未来

- **人工知能（AI）とデータベースの統合**

分析や意思決定での、AIとデータベース利用が普及

- **巨大なデータ処理や実時間処理の進展**

より高性能なシステムが求められるように。

- **セキュリティ技術の深化**

ブロックチェーンによるデータの不変性（改ざんされていないことの保証）、セキュリティが向上。

- **クラウドコンピューティングの普及**

ITシステムは、より使いやすく、安価で、高性能なものになる。

16-5. データベースシステム とその周辺分野

データベースシステム

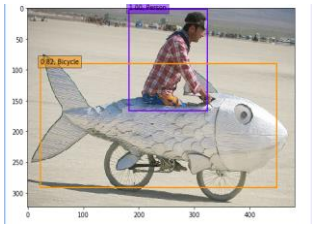
・データベース管理システムは，データベースの管理等の機能を持ったソフトウエア
・オンラインでデータを共有するときに，特に適する

取引

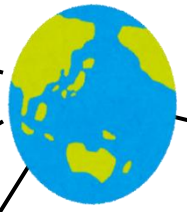
記入

データ保存

センサー連携



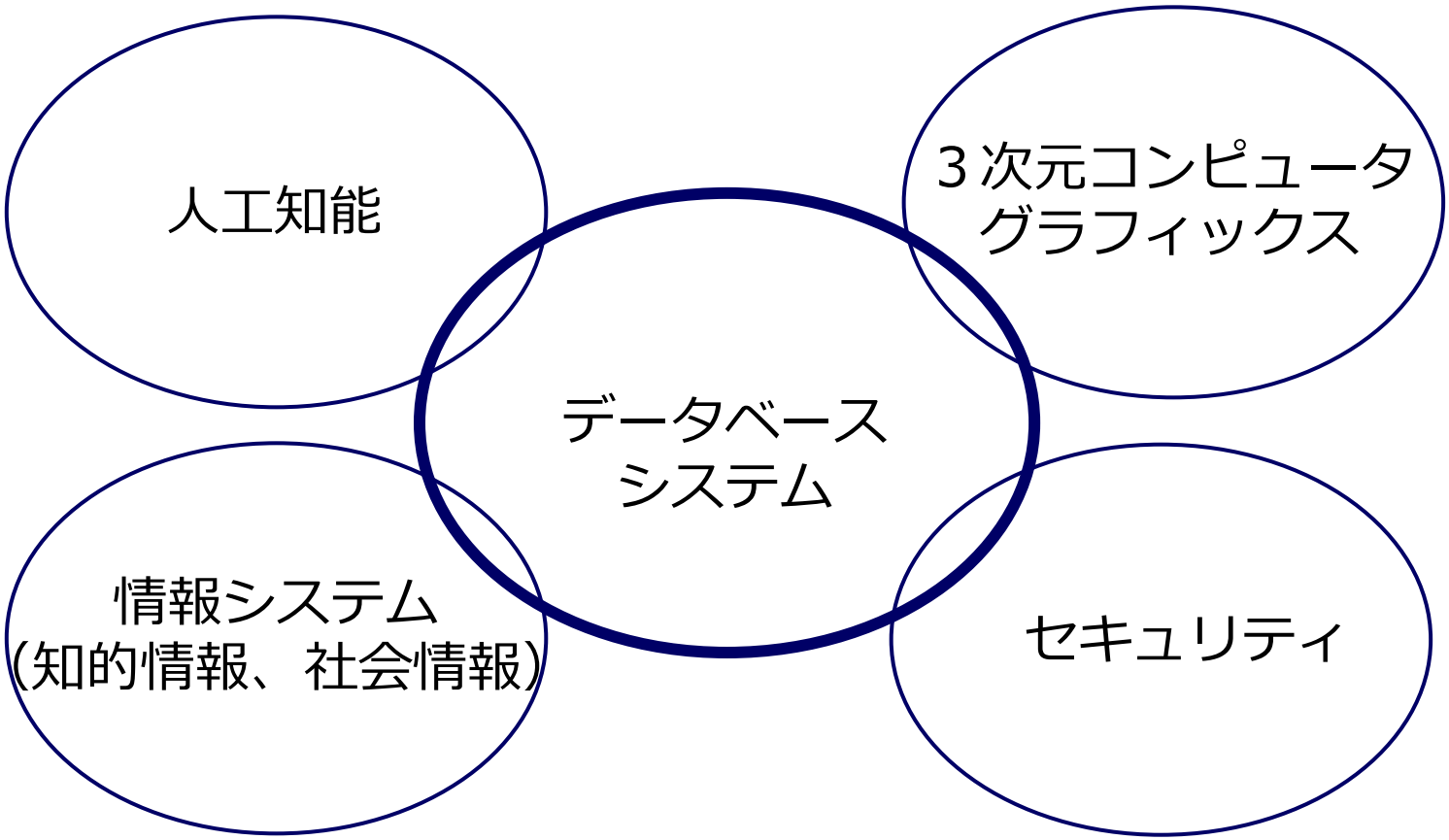
人工知能応用



ネットワーク

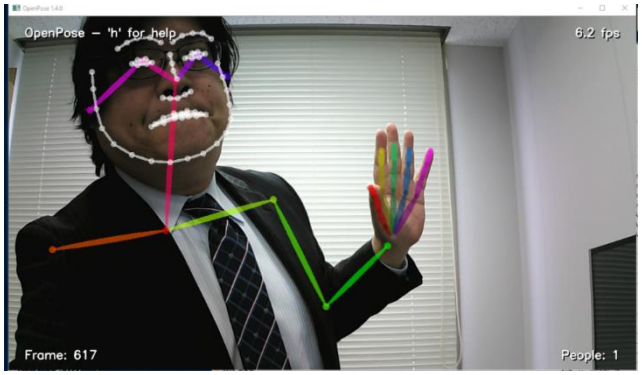
データベースシステム

「データベースシステム」と周辺研究領域

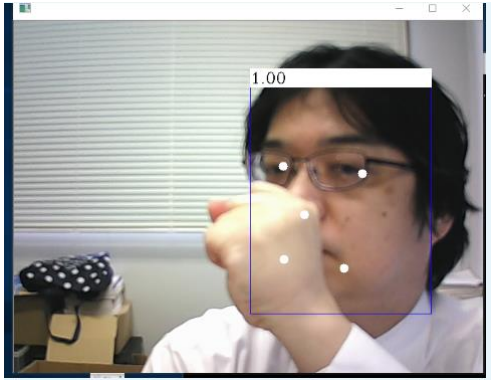


データベースシステムとAI（人工知能）の融合
人工知能の学習にはデータが必要

モーションキャプチャ実験（マーカーレス）

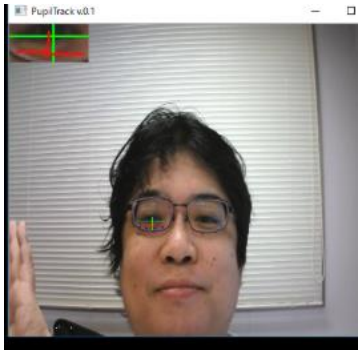


顔検出実験



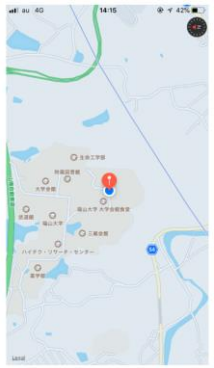
ナンバープレート自動読み取り実験

眼球運動、顔の動きの自動抽出実験



データベースシステムと社会情報
データの収集，蓄積，活用で，データベースシステムが役に立つ

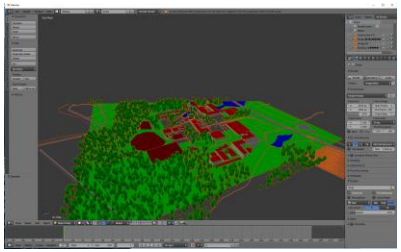
情報共有型地図アプリの製作

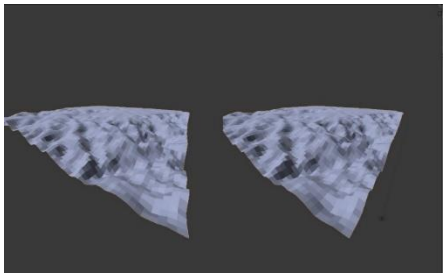


プライベートな
グループ向けアプリ
- 地図表示
- 位置表示
- 写真投稿
- メッセージやり取り

車両観測システムの製作

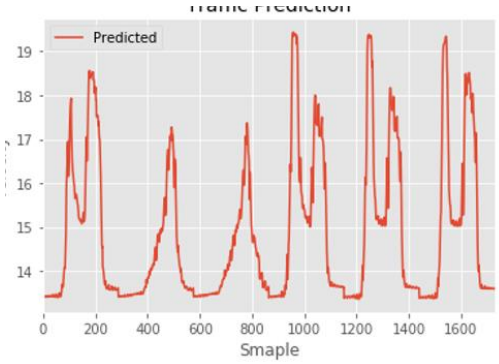
広域３次元地図の製作





ヴァーチャル・リアリティ表示も可能

センサーからの交通密度プロット



16-6. キー・バリューのデータベース

データベースシステムの種類

① リレーショナルデータベースシステム

	路線コード番号	事業者コード番号	路線名称一般	路線名称一般カナ
1	1001	3	中央新幹線	チュウオウシンカンセン
2	1002	3	東海道新幹線	トウカイドウシンカンセン
3	1003	4	山陽新幹線	サンヨウシンカンセン
4	1004	2	東北新幹線	トウホクシンカンセン

テーブル

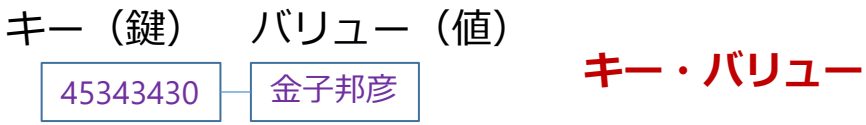
② ドキュメント指向のデータベースシステム

XMLなどの**ドキュメント**に特化.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap
0.0.2">
 <bounds minlat="54.0889580"
minlon="12.2487570" maxlat="54.0913900
maxlon="12.2524800"/>
 <node id="298884269" lat="54.0901746"
lon="12.2482632" user="SvenHRO"
uid="46882" visible="true" version="1"
changeset="676636" timestamp="2008-09-
```

ドキュメント

③ キー・バリュー形式のデータベースシステム（キー・バリュー・ストア）



キー・バリュー

その他さまざま

キー・バリュー形式のデータの例

ID　　パスワード，氏名，住所

キー　　バリュー

キー・バリュー形式のデータの例

	name	price
0	apple	10
1	orange	20

0, name　　apple

キー　　バリュー

キー・バリュー形式のデータを扱う２つの方法

- ファイル

  JSON などの方法

- キー・バリュー形式のデータベースシステム

JSON とは

「**タグ**」と「**値**」を**並べる**ことを特徴とするデータ形式

```
{"name": "apple", "price": 10}
```

{タグ: 値, タグ: 値}

JSON の例

```
In[25]: x = {"name": "apple", "price": 10}
In[26]: print(json.dumps(x))
{"name": "apple", "price": 10}

In[27]:
```

Pythonで扱うことも簡単

JSON のデータ型

- 配列　　例 [1, 2, 3]
- 数値　　例 4.56
- 文字列　　例 "hello"　※　「"」で囲む
- ブール値　　true　false
- null

```
{"name": "apple", "price": 10}
```

- **タグ**　**必ず文字列**
- **値**　　さまざまな**データ型**でありえる.
  JSON オブジェクトを値とすることも

JSON での入れ子

入れ子： **値**が **JSON オブジェクト**になること

id	name	price
0	apple	10
1	orange	20

リレーショナルデータベースのテーブル

同じ意味

```
{"name":{"0":"apple","1":"orange"},"price":{"0":10,"1":20}}
```

JSON での入れ子

JSON とキー・バリュー形式のデータベースシステム Cloud FireStore の違い

	JSON	Google Firebase
オブジェクトの種類	JSONオブジェクト	**コレクション**，ドキュメント
入れ子	OK	OK
「値」の種類	文字列 (string), 数値 (number), ブール値 (boolean), 配列 (array), null	文字列 (string), 数値 (number), ブール値 (boolean), 配列 (array), null, **日時 (timestamp),** **geopoint,** **参照 (reference)**