ChatGPT

Teil A – Videozusammenfassung und Best Practices

Wichtigste Learnings aus dem Video *(“Getting the most out of BigQuery… for patent research”)*

1. **BigQuery ermöglicht** Skalierbare Patentabfragen**: Man kann** Millionen von Patentdokumenten effizient per SQL analysieren*, statt lokal Daten zu laden* [1] [2]*. Dadurch lassen sich Fragen beantworten wie „Wie viele Patente haben mehr als einen Erfinder?“* direkt in BigQuery [3].

2. **Vielfältige** öffentliche Patent-Datensätze **stehen bereit**: Google’s Patent-Datasets auf BigQuery umfassen globale **Bibliografie-Daten zu 90+ Mio. Patentschriften** (aus 17 Ländern) sowie vollständige US-Patentinhalte [4]. Zusätzlich gibt es abgeleitete **Research-Daten** mit maschinellen Übersetzungen, Ähnlichkeits-Vektoren und Schlüsselbegriffen [5]. Weitere Datensätze (z.B. **PatentsView**, **CPC-Schema**, **USPTO**-Daten) sind ebenfalls frei verfügbar.

3. **Kombination mehrerer Datensätze**: Ein zentrales Learning ist das **gemeinsame Abfragen verschiedener Patentdatensätze** in einer SQL-Umgebung. BigQuery erlaubt **Joins über Bibliografie-, Forschungs-, und Prüfungsdaten**, sowie **Chemie- oder Rechtsdaten** [6] [7]. So kann man z.B. **Patente mit FDA-Wirkstoffdaten oder Klagedaten verknüpfen**, um komplexe Fragen (z.B. *„Welche patentierten Medikamente wurden mit Regierungsgeldern entwickelt?“*) in **einer Abfrage** zu beantworten [6].

4. **Zeitersparnis durch gehostete Daten**: Da die Daten bereits in BigQuery strukturiert vorliegen, entfällt das mühsame **Herunterladen, Parsen und Laden großer Patentdatensätze** [8]. Forscher können mehr Zeit für **Analyse statt Datenaufbereitung** verwenden [9].

5. **Leistungsfähigkeit für** Trend- und Landscape-Analysen***: BigQuery bewältigt komplexe Analysen sehr schnell. Beispielsweise kann man mit einer einfachen SQL-Abfrage** ***Patent-Trends** ermitteln (*„Nutzung des Begriffs ‘Internet of Things’ über Zeit“*) [10] oder **technologische Landscapes** erstellen (z.B. Patentanmeldungen in *Gen-Therapie* pro Jahr und Firma) [11]. Solche Abfragen wären lokal sehr zeitaufwändig, laufen aber in BigQuery dank verteiltem Rechnen in Sekunden [2].

6. **Einfache** Verteilung von Ergebnissen**: BigQuery-Resultate lassen sich direkt** in Google Data Studio/GSheets visualisieren oder als CSV exportieren** [12]. So können Analysten ihre Patent-Auswertungen leicht mit dem Team teilen, ohne komplexe Datenpipelines.

7. **BigQuery** Kostenmodell **verstehen**: Ein Learning ist, **Abfragekosten im Blick** zu behalten. BigQuery berechnet nach gescanntem Datenvolumen, daher sollte man Abfragen optimieren (siehe Best Practices unten). Die im Video gezeigten Beispiele demonstrieren, dass **selbst komplexe Patent-Analysen kostengünstig** ausgeführt werden können, wenn man gezielt filtert und nur benötigte Spalten abruft [2].

8. **Umgang mit** verschachtelten Feldern**:** Patentdaten enthalten oft **Arrays und Strukturen** (z.B. mehrsprachige Abstracts, CPC-Listen, Zitationslisten). Im Video lernt man, mit `UNNEST()` umzugehen, um solche Felder zu explodieren [13] [14]. Beispiel:

`patents.publications.citation` ist ein Array von Zitationen; man muss es mit `UNNEST(citation) AS c` auseinanderziehen, um jede einzelne Zitierung auszuwerten [13].

9. **Bestehende** Schema-Standards nutzen**: Die Datasets sind aufeinander abgestimmt.** Schlüssel wie `publication_number` **sind standardisiert (Land-Kode + Nummer + Kind-Code, z.B. "US-7499872-B1") und ermöglichen es,** über Tabellen hinweg zu joinen** [15] [16]. Im Video wurde betont, dass man diese Standard-IDs nutzen soll, statt eigene Varianten zu basteln.

10. **Praxisnahe** Beispiele**: Der Vortrag vermittelt das Wissen anhand konkreter Fragestellungen (Portfolio-Übersichten [17], einfache Landscapes [11], Zitierungsanalysen [18]). Dadurch wird klar,** welche Analysen direkt "out-of-the-box" möglich sind **– von der** Anmeldejahr-Auswertung **bis zur** Forward-Citation-Analyse** – alles mit einfachen SQL-Queries.

BigQuery-spezifische Best Practices (Kosten, Performance, Schema)

- **Gezielte Spaltenwahl statt** `SELECT *`: Im Video wird gewarnt, dass `SELECT *` auf Tabellen mit hunderten Spalten massiv Daten scannt – teuer und langsam. **Tipp**: Nur benötigte Felder auswählen. Beispiel: Für Patentanzahl pro Jahr reichen `filing_date` und `publication_number` [19]; andere Felder weglassen.

- **Filter für Partitionen nutzen**: Viele Patenttabellen sind nach Datum partitioniert (z.B. nach Publikationsjahr). Ein *WHERE*-Filter auf das Partitionsfeld sorgt dafür, dass BigQuery **nur relevante Partitionen liest**. **Beispiel**: `WHERE publication_date >= 20200101` reduziert das Scanvolumen, indem Daten vor 2020 übersprungen werden (Partition Pruning).

- **Clustering verstehen**: Einige Tabellen sind nach häufig gefilterten Feldern geclustert (z.B. Land, CPC-Code). Das Video empfiehlt, bei großen Joins zuerst nach Ländern oder Zeiträumen einzuschränken, damit BigQuery dank Clustering **weniger Blöcke durchsuchen** muss. Beispielsweise ein *WHERE country_code='US'* nutzt den Cluster auf `country_code` optimal und spart Kosten [19].

- **Verschachtelte Felder effizient nutzen**: Patentdaten enthalten Arrays (z.B. **mehrere Erfinder, CPCs, Zitate**). Statt die Normalisierung auf zig Tabellen auszulagern, sind sie als Arrays im JSON-Stil gespeichert. Best Practice: **UNNEST nur bei Bedarf**. Beispiel: Will man nur die **Anzahl der Zitate** pro Patent, kann man `ARRAY_LENGTH(citation)` verwenden, ohne das ganze Zitations-Array zu entpacken – das ist deutlich günstiger als komplettes Unnesting.

- **Kostenabschätzung und Limits**: Im Video wird gezeigt, wie man im BigQuery UI das **Datenvolumen einer Abfrage** vorab sieht. So kann man eine Query abbrechen, falls sie z.B. 100 GB scannen würde, und stattdessen erst Filter oder Pre-Aggregationen einbauen. Merke: **1 TB frei pro Monat** (Stand bei Veröffentlichung) – *die gezeigten Patentabfragen lagen oft nur im zweistelligen GB-Bereich* und waren damit kaum kostenrelevant.

- **Materialisierte Sichten / Tabellen**: Für wiederkehrende komplexe Abfragen empfiehlt der Vortrag, **Zwischenergebnisse als Tabelle abzulegen**. Z.B. eine bereinigte **"filtered_publications"**-Tabelle, die nur Patentfamilien eines Technologiefelds enthält. Diese einmal zu erstellen und dann immer wieder zu nutzen ist effizienter, als jedes Mal den ganzen Datenbestand zu filtern. BigQuery unterstützt auch **materialized views**, die im Hintergrund aktualisiert werden und Abfragen beschleunigen, wenn sie auf aggregierten Teilmengen basieren (im Patentkontext z.B. Pre-Aggregation der jährlichen Fallzahlen).

- **Freigegebene** Views **für Teams**: Als Best Practice für **Ergebnisverteilung** schlägt das Video vor, **Views mit selbsterklärenden Schema** zu erstellen (z.B. eine View `portfolio_analysis` mit Feldern wie Jahr, Anzahl Anmeldungen, Top-Inhaber etc.). Diese können mit Berechtigungen versehen teamweit genutzt werden, ohne jedem Roheinzeldaten zeigen zu müssen.

- **Export großer Ergebnisse**: Wird eine Analyse sehr umfangreich (sagen wir >100MB Resultset), sollte man nicht alles durch den Browser ziehen. Stattdessen: Ergebnis in BigQuery als Table schreiben und z.B. als **Parquet auf Google Cloud Storage exportieren**. Das Video erwähnt, dass **Parquet-Exporte** insbesondere für Machine-Learning-Pipelines sinnvoll sind, da sie effizient und schema-beibehaltend sind.

- **Partitionierung eigener Tabellen**: Wenn man **eigene zusammengesetzte Datensätze** erstellt (z.B. durch Joins), sollte man beim Anlegen gleich Partitionierung definieren, etwa nach Jahr. So profitieren Folgeabfragen direkt davon. Beispiel: Eine eigene Tabelle "tech_landscape" mit Spalte `publication_year` als Partition – Abfragen für einzelne Jahrgänge sind dann sehr schnell.

- **Wiederholte Felder handhaben**: Patentdaten haben z.B. `inventor` als Wiederholfeld (mehrere Erfinder). Das Video betont, dass man hier aufpassen muss: **Joins oder Groupings ohne** `UNNEST` behalten das Wiederholfeld als Einheit. Will man je **einzelnen Erfinder zählen**, *muss* man entpacken. Aber wenn man nur pro Patent zählen will, sollte man *nicht* unnesten, um keine Vervielfachung der Zeilen zu verursachen. **Faustregel**: Unnest **gezielt und möglichst spät** in der Query, nur wenn wirklich nötig.

Typische Fallstricke bei Patentdaten in BigQuery

- **Unterschiedliche Nummernformate**: Patentnummern können je nach Quelle verschieden formatiert sein. Beispiel: **USPTO PTAB**-Datensatz hat Patentnummern rein numerisch (z.B. `7499872`), während der Hauptdatensatz `publication_number` mit Länderkürzel und Kindcode nutzt (`US-7499872-B1`) [15]. Ohne Anpassung *joinen diese nicht direkt*. Fallstrick: Nummernstrings vor Join vereinheitlichen oder die bereitgestellten *"match"*-Tabellen nutzen [20] [16].

- **Double Counting**: Ein Patent kann mehrfach in Daten erscheinen (z.B. als **Anmeldung und erteiltes Patent**). Im globalen Datensatz ist beides unter derselben `application_number`, aber mit unterschiedlichem `publication_number` (A1 vs. B1). Wer z.B. *USA-Patente zählt*, muss entscheiden: Will ich *Anmeldungen + Grants zählen* oder nur einzigartige Familien? Sonst zählt man evtl. doppelt. **Best Practice**: Den **Familienbegriff** oder die `family_id` nutzen, um pro Erfindung zu zählen [21].

- **Begrenzte Felder für Nicht-US-Patente**: Im *Google Patents Public Data*-Table sind **Ansprüche und Beschreibung nur für US-Dokumente** vorhanden (volltext) [22]. Für andere Länder gibt es i.d.R. nur bibliographische Daten, Titel/Abstract (oft nur auf Landessprache). Fallstrick: Eine Query, die `claims` durchsucht, liefert außerhalb US **NULL-Ergebnisse** – das ist erwartbar (kein Fehler in den Daten). Hier muss man evtl. auf Übersetzungsdaten (Research-Table) zurückgreifen, die **englische Abstracts für alle Länder** bieten [23].

- **Ländercodes und Normierungen**: In verschiedenen Datensätzen können Länder und Dokumentarten unterschiedlich codiert sein. Z.B. **PatentsView** nutzt `country = 'US'` und separates Feld für "patent vs application", Google-Daten haben alles im `publication_number`

integriert. Wenn man **USPTO-Daten mit EPO-Daten** mischt, gilt es auf Konsistenz zu achten (z.B. *EP vs. WO vs. national* – hier lieber über *country_code* unterscheiden).

- **Verwechslung von Schlüsseln**: USPTO-Datensätze haben verschiedene IDs: `application_number` (Anmeldenummer), `publication_number` (Veröffentlichungsnummer der A-Schrift), `patent_number` (Nummer der erteilten B-Schrift). Beispiel: **PatEx** nutzt `application_number` als Primärschlüssel je Eintrag (ein Antrag), **PatentsView** hat `patent_id` (interne ID) und `number` (Patentnummer). Ein häufiger Fehler ist, `application_number` mit `patent_number` zu verwechseln – die überschneiden sich numerisch *nicht*. Immer den Kontext der ID prüfen (Antrag vs. erteiltes Patent).

- **Große Arrays falsch behandeln**: Zitations- und CPC-Arrays können sehr groß sein (eine Patentschrift mit 100+ Zitaten). **Fehler**: CROSS JOIN mit UNNEST ohne Filter kann zu *Explosions* der Ergebniszeilen führen. Besser: vor dem Unnesting die relevanten Dokumente filtern (z.B. erst auf eine *Patentliste WHERE assignee = 'X'* einschränken, dann in dieser Teilmenge `UNNEST(citation)`).

- **Veraltete Datenannahmen**: Patentdaten ändern sich laufend (z.B. neue CPC-Klassen, Rechtsstandänderungen). Ein Fallstrick ist, *anzunehmen, die Datensätze seien immer up-to-date*. **Prüfen**: Update-Frequenzen (viele BigQuery-Datensätze werden **vierteljährlich** aktualisiert [24], manche jährlich, manche gar nicht mehr). Bei zeitkritischen Analysen (z.B. *Anmeldetrend in 2025*) sicherstellen, dass der Datenstand aktuell genug ist.

- **Textsuche Performance**: BigQuery beherrscht zwar SQL, aber für **Volltextsuche in Abstracts/ Claims** ist es nicht wie ElasticSearch optimiert. Ein Fallstrick ist, sehr breite LIKE-Queries (z.B. `WHERE abstract_text LIKE '%machine learning%'`) über zig Millionen Zeilen laufen zu lassen – das kann teuer werden. Besser: voraggregierte **Top-Terms oder Embeddings** nutzen (die *Research Data* enthalten 10 wichtigste Begriffe pro Patent [25], was die Suche nach Themen erheblich beschleunigt). Oder mit Volltext-Filtern kombiniert vorgehen (z.B. erst CPC filtern, dann Textsuche innerhalb kleinerer Menge).

- **Fehlinterpretation von Feldern**: Einige Felder klingen ähnlich, bedeuten aber Unterschiedliches. Beispiel: In **PatentsView** gibt es `rawassignee` vs. `assignee` (einer ist normalisiert). Oder `application_type` (kann "provisional", "PCT", "utility" bedeuten). Hier im Zweifel die Dokumentation zu Rate ziehen, um korrekte Filter zu setzen (z.B. nur `application_type = 'utility'` wenn nur US-Nonprovisionals gewünscht).

- **Zitationsrichtung verwechseln**: Der *citations* Array in der Google-Tabelle enthält **backward citations** (Referenzen, *die das Patent macht*). Möchte man die Forward Citations (wer *zitiert dieses Patent später*), muss man die Rolle umdrehen (selbst Join auf Citation-Tabelle oder invertierte Abfrage) [26] [27]. Ein häufiger Fehler ist, die Rohdaten ohne Anpassung zu verwenden und dann falsche Schlüsse über „Zitiert-von" zu ziehen.

Mit diesen Learnings und Hinweisen im Hinterkopf ist man gut gerüstet, die nachfolgenden Details zu Datensätzen und Analysen nachzuvollziehen.

Teil B – Patent-Datasets in BigQuery: Katalog

Im Folgenden ein Katalog wichtiger Patent-bezogener Datensätze, die in BigQuery verfügbar sind. Die Tabelle gibt einen Überblick zu Inhalt, Quelle, Abdeckung etc. für jeden Datensatz. (Weitere themennahe Datasets – z.B. Chemie oder Litigation – sind unten angefügt.)

Legende: *"BigQuery-Pfad"* gibt das Projekt und Dataset an (Tabellen darin siehe Teil C). *Join-Schlüssel* nennt wichtige Felder für Verknüpfungen. *Lizenz* ist indikativ (kein Rechtsrat).

Dataset (Name)	**BigQuery-Pfad**	**Anbieter/Quelle**	**Inhalt**
Google Patents Public Data (IFI/ Google)	`patents-public-data.patents`	IFI CLAIMS & Google [28] [29]	Bibliographische **Metadaten z** **>90 Mio. Patentschriften** welt (mehr als 100 Länder; inkl. EP, V CN, JP, DE, etc.) sowie **komplet** **Volltext für US-Patente** [4] . Enthält u.a. Titel, Abstracts (mehrsprachig), Anmelde- und Veröffentlichungsdaten, Prioritäten, **IPC/CPC**-Klassen, **Zitationen** (Patentzitate), **Erfin** **& Anmelder** (roh und harmonisiert), **Familien-ID** usv
Google Patents Research Data	`patents-public-data.google_patents_research`	Google Global Patents Team [33]	**Abgeleitete Patent-Features**: Enthält pro Patentdok. vorverarbeitete Daten: **maschinelle Übersetzungen** a Titel und Abstracts ins Englisch (mittels Google Translate) [33] , **Embedding-Vektoren** (64-dimensionale numerische Vektoren, die Patentinhalt repräsentieren) [34] , **"Top Term** (die 10 wichtigsten Schlagwört pro Patent, 1-2-gramme aus Volltext) [25] , **Ähnlichkeits-Sco** und evtl. Links (z.B. `url` zur Google Patents Seite).

Dataset (Name)	**BigQuery-Pfad**	**Anbieter/Quelle**	**Inhalt**
Disclosed Standard Essential Patents (SEP)	`patents-public-data.standard_essential_patents` (vermutl.)	OEIDD / Academic (Std-Patente DB) [38]	**Datenbank deklarierter Standardessentieller Patente** Sammlung aller bei **Standardisierungsorganisati** (insb. **ETSI, IEEE, ITU, etc.**) gemeldeten Patentdokumente (Stand ~2011). Enthält bereinig Listen von **Patentnummern (U und EP)**, zugehörige **Standard Dokumente/Standards** und Angaben zu meldenden Firmen SSOs [38] .
USPTO PatentsView (Disambiguated US Data)	`patents-public-data.uspto_patentsview`	USPTO OCE & PatentsView consortium [39]	**Umfassende US-Patentdatenbank** mit **Bereinigung & Vernetzung**: Enthält alle US-Patente und -Veröffentlichungen (Grant ab 1 Applications ab 2001) mit **normalisierten Entitäten** (z.B **Assignee und Erfinder disambiguiert** – gleiche Perso Firma hat eindeutige ID) [39] . Mehrere Tabellen: **patent** (Stammdaten pro erteiltem Pat **application** (Veröffentlichte Anmeldungen), **assignee**, **inventor**, **lawyer**, **locations**, **citations** (uspatentcitation, foreigncitation, npl), **classes** (C IPC, USPC, NBER, WIPO fields + Lookup Tabellen) [39] [40] , **government_interest** (Angabe Regierungsförderung).

Dataset (Name)	**BigQuery-Pfad**	**Anbieter/Quelle**	**Inhalt**
USPTO Patent Examination Research Dataset (PatEx)	`patents-public-data.uspto_oce_pair` (Dataset, Tabellen z.B. `application_data`)	USPTO OCE (Office of Chief Economist) [43]	**Detaildaten zum Prüfungsver** aller öffentlich einsehbaren US Patentanmeldungen. Enthält p **Patentanmeldung** (Antrag) bibliografische Angaben und **Prüfungsmetriken**: z.B. *Eingangsdatum, Art der Anmeldu (non-provisional, PCT national…) technische Hauptklasse, Prüfer-I Art Unit, aktuelle Statuscodes, Ausstellungsdatum falls erteilt, Abandon-Date falls verworfen*, Anzahl RCEs, Interviews etc. Zusätzlich gibt es eine Tabelle m **allen Erfindern je Anmeldung** (Name, Ort, Reihenfolge) [44] [4 PatEx deckt ~11,9 Mio. Anmeldungen (inkl. Prov.) bis ~ ab [43] .
USPTO Patent Examination Data System (PEDS)	*kein voller Bulk-Dataset* (API-Zugang; evtl. teilweise in obigen enthalten) [47] [48]	USPTO via API/ Web (PAIR Ersatz) [47]	**Live-Datenbank** aller laufende **Anmeldungen** beim USPTO se 1981 (teilw. Daten ab 1935). En ähnlich PatEx **Bibliographisch Status** (jedoch keine **Verlaufsdetails** wie Office Acti Im BigQuery-Kontext keine eig Tabelle – eher via API nutzbar o Einzelexporte.
USPTO OCE Patent Claims **Research Data**	`patents-public-data.uspto_oce_patent_claims`	USPTO OCE [50]	Enthält den **Wortlaut der Patentansprüche** für alle **US-Patente (Granted 1976–2014)** **US Publikationen (2001–2014)** Jede Anspruchsfassung ist einz erfasst (mit Claim-Nummer, unabhängig/abhängig-Markier Text). Drei Tabellen: wahrschein **granted_claims**, **pub_claims** u evtl. Metadaten.

Dataset (Name)	**BigQuery-Pfad**	**Anbieter/Quelle**	**Inhalt**
USPTO OCE Office Actions **Research Data**	`patents-public-data.uspto_oce_office_actions`	USPTO OCE [51]	Datensatz zu **Prüferbescheide (Office Actions)**: Details zu *4,4 Prüfungsbescheiden* in *2,2 Mio. Patentanmeldungen* von 2008 b 2017 [52] . Aufgeteilt in **drei Tab** 1) **office_actions** (Basisinfos pr Bescheid: Appl.-ID, Mail-Datum Non-Final/Final, Event-Code), 2 **rejections** (Ablehnungsgründe Anspruch/Bescheid: enthält Zitathinweise, §§ etc.), 3) **citati** (die im Bescheid zitierten Referenzen, inkl. Patent- und N Zitate) [52] .
USPTO PTAB Data (Trial & Appeal Board)	`patents-public-data.uspto_ptab`	USPTO PTAB via Google [15]	**PTAB Verfahrensdaten**: Enthä Infos zu **Patentstreitverfahre vor der PTAB**. Schwerpkt. AIA-T (Inter Partes Review, Post-Gran Review, CBM) seit 2012: Tabelle **trials** mit Feldern wie *Trial Num (Case-ID), Petitioner (Herausford Patent Owner, angefochtene Patentnummer, Art des Verfahren Status/Outcome* [55] [56] . Möglicherweise auch **Appeals** (Prüfer-Entscheidungsinstanz) integriert – unklar. Zusätzlich e **match**-Tabelle, um Patentnummern/ Anmeldenummern mit den Pul Data-Format zu verknüpfen (z.I numeric -> US-B1 Format) [15]
Cooperative Patent Classification (CPC) Schema	`patents-public-data.cpc_scheme` (o.ä.)	USPTO & EPO (öffentlich) [58]	**Nachschlagetabelle aller CPC Klassen**: Hierarchische Struktu aller **CPC Codes mit Benennu und Definition**. Jede Zeile ist e Klassifikationseinheit (z.B. *"A61K48/00"* mit Titel *"Gen thera* Enthält Felder: CPC **Symbol**, **Tit (Kurztext)**, ggf. **Definition** (Langtext), **Level** (Sektion, Klas Subklasse etc.), **Parent-Code** (übergeordnete Klasse).

Dataset (Name)	**BigQuery-Pfad**	**Anbieter/Quelle**	**Inhalt**
Matrixware Research Collection (MAREC)	`patents-public-data.marec`	IRF/MATRIXWARE (EU-FP7 Projekt) [59]	**Historische Patent-Volltextsamlung (1976–2008)**: Mio. Patentdokumente aus EPO (EP), WIPO (WO), USPTO (US) u JPO (JP) [59] in **vereinheitlichte Format**. Enthält wahrscheinlich **Volltext (Titel, Abstract, Ansprüche, Beschreibung) in durchsuchbarer Form** sowie Metadaten (Nummer, Datum, Herkunft). MAREC war ursprün ein Benchmark-Datensatz für Information Retrieval (CLEF-IP)
UCB Fung Institute Patent Data (US)	`patents-public-data.ucb_fung_patents` (vermutet)	UC Berkeley Fung Institute [60]	**Alternative aufbereitete US-Patentdatenbank**: Mit moder ML-/NLP-Methoden wurden USPTO-Patente automatisch geparst, disambiguiert und in e aktualisierte DB überführt [60] . Fokus vermutlich auf **Inventor und Firmenentitätsauflösung** (ähnlich PatentsView, evtl. verbessert) und **Text-Mining**-Features. Könnte z.B. zusätzlich **Wortfeld-Analysen oder technologische Vernetzung** enthalten (Projektbeschreibung erwähnt neue Tools).
Patent PDF Samples (mit extrahierten Strukturdaten)	`patents-public-data.patent_pdf_samples`	Google ML (Beispieldatensatz)	Kleine Sammlung von **Patent-P und deren automatisch extrahierten Daten** (Claims, Zeichnungsbeschriftungen, chemische Strukturen etc.), als Demonstration für ML-Extrakti Dient als **Beispiel**, wie man aus PDF-Dokumenten strukturierte Infos gewinnt (z.B. Tabellen aus Beschreibungen).

Dataset (Name)	**BigQuery-Pfad**	**Anbieter/Quelle**	**Inhalt**
SureChEMBL – Chemische Patentauszüge	`patents-public-data.surechembl`	EMBL-EBI (SureChEMBL) [61]	**Große extrahierte Chemie-Datenbank aus Patenten**: Ent >15 Mio. **chemische Struktur Verbindungen**, die aus Volltex von Patentdokumenten automatisch extrahiert wurden (chem. Namen, Strukturbilder). Verknüpft Patentreferenz → chemische Verbindung (InChI K etc.).
ChEMBL (Pharmadatenbank, verlinkt mit Patenten)	`bigquery-public-data.chembl`	EMBL-EBI (ChEMBL) [62] [63]	**Wirkstoff-Datenbank**: Informationen zu ~2 Mio. **chemischen Substanzen**, inkl. **Arzneimittel-Zulassungen (FD Orange Book)** und Patentreferenzen. Ermöglicht Verknüpfung von **Wirkstoff → zugehörige Patente** (z.B. Patentnummern aus Orange B
USITC Section 337 Data (Patent Litigations)	`patents-public-data.usitc_337`	USITC (via Google) [64] [65]	Datenbank der **US Internation Trade Commission** zu *"337 Investigations"* (Import-Untersagungsverfahren, oft Patentstreitigkeiten). Enthält p Verfahren: **Investigation Nr**, beteiligt Firmen, **betroffene Patente** (Nummern) sowie **Industrie-Sektor** (WIPO Technologiefeld).

> **Hinweis:** Einige Datasets (v.a. USPTO OCE) sind umfangreich. Bei unsicherer Tabellenstruktur empfiehlt sich ein Blick in `INFORMATION_SCHEMA` oder die Google Cloud Marketplace-Beschreibung. So lässt sich verifizieren, ob z.B. `uspto_oce_office_actions.citations` existiert. Details dazu im nächsten Teil.

Teil C – Datenstrukturen und Schema-Details der Kern-Datensätze

In diesem Abschnitt beleuchten wir für jedes Kern-Dataset wichtige Tabellen und Felder. Wir zeigen, wie man mit BigQuerys `INFORMATION_SCHEMA` die Struktur ermittelt und heben pro Dataset ~10 zentrale Felder (mit Datentyp) hervor. Zudem erklären wir, welche Felder **verschachtelt** sind (Arrays/Structs) und wie man sie entpackt.

Beispiel: Um alle Tabellen im *Patents Public Data*-Dataset aufzulisten, kann man ausführen:

```
SELECT table_name
FROM `patents-public-data.patents.INFORMATION_SCHEMA.TABLES`;
```

Ähnlich liefert `INFORMATION_SCHEMA.COLUMNS` Details zu Spalten. So könnten wir für `patents.publications` die Spalten abfragen:

```
SELECT column_name, data_type
FROM `patents-public-data.patents.INFORMATION_SCHEMA.COLUMNS`
WHERE table_name = 'publications';
```

(In der Praxis kann man auch die BigQuery-Weboberfläche nutzen, die Schemata anzeigt.)

Im Folgenden die wichtigen Tabellen je Dataset und ihre Hauptfelder:

Google Patents Public Data – Tabelle `patents.publications`

- **Tabellenübersicht**: Alle Patent-Publikationen weltweit in *einer großen Tabelle* `publications`. **Wichtig:** Es handelt sich um **eine Zeile pro veröffentlichtem Dokument** (d.h. *A1, B1, C1 etc. jeweils eigene Zeile*). Für eine Anmeldung erscheinen mehrere Zeilen, wenn es z.B. eine A1 und später B1 gibt – erkennbar am gemeinsamen `application_number`.

- **Wichtige Felder (Spalten) & Typen**:

- `publication_number` (STRING): Eindeutige Publikations-ID mit Land und Kindcode, z.B. `"US-8616463-B2"` [42]. *Schlüssel* für Joins, identisch mit Patentnummer für Grants (mit Code) [15].
- `application_number` (STRING): Anmeldenummer (mit Seriencode bzw. Ländercode-Präfix). Für US z.B. `"12/345678"`; für EP oft Format `"EP-2008-XYZ..."`. Alle Dokumente einer Familie teilen sich diese Nummer [16]. Nutzen für Gruppierung von Veröffentlichungs- und Erteilungsdokumenten.
- `country_code` (STRING): 2-stelliger Länderkode (z.B. "US", "EP") [67].
- `kind_code` (STRING): Publikationsart (A1, A2, B1, B2, etc.).
- `filing_date`, `publication_date`, `grant_date` (INT64): Datumsangaben im Format YYYYMMDD als Zahl [19]. Beispiel: 20131231 → 31.12.2013. *Partitioniert* vermutlich nach `publication_date`.
- `priority_date` (INT64): Frühestes Prioritätsdatum (falls berechnet vorhanden).
- `title_localized` (RECORD/STRUCT): Struktur für Titel mit Sprachinfos. Enthält z.B. Felder `text` (STRING) und `language` (STRING). Meist nur 1 Eintrag – außer wenn mehrsprachige Veröffentlichungen (EP bilingual etc.). **Nicht verschachtelt** (einzeln).
- `abstract_localized` (ARRAY<STRUCT>): **Array** von Abstracts, da ein Patent mehrere Abstracts in unterschiedlichen Sprachen haben kann [68]. Strukturfelder: `text` (STRING), `language` (STRING). **Beispiel**: `SELECT ... FROM patents.publications, UNNEST(abstract_localized) AS abs WHERE abs.language='en'` filtert den englischen Abstract [68].
- `claims_localized` (ARRAY<STRUCT>): Enthält für US-Dokumente die Ansprüche (ggf. ein einziger String mit alle Ansprüche, Sprache). Bei nicht-US meist leer. Kann großen Text enthalten.
- `description_localized` (ARRAY<STRUCT>): Analog: die Beschreibung (Detailed Description) – i.d.R. nur US.

- `assignee_harmonized` (ARRAY<STRUCT>): **Wichtig für Analysen** – normalisierte Anmelderdaten. Struktur z.B.: `name` (STRING – vereinheitlichter Name, z.B. "International Business Machines Corp." statt diverse Schreibweisen) [29], `city`, `state`, `country` (alle STRING). Oft nur 1 Eintrag, kann aber mehrere enthalten (bei Co-Assignees, Regierungsinteresse Statements etc.). Nutzung: `UNNEST(assignee_harmonized)` bei Bedarf, z.B. Top-Anmelder zählen.
- `inventor` (ARRAY<STRUCT>): Erfinder-Rohdaten – Felder z.B. `name_first`, `name_last` (oder ein Kombinationsfeld `name`), plus Ort. *Nicht disambiguiert*, einfach so wie am Dokument.
- `examiner` (STRUCT): Prüfername (nur bei US-Grants relevant).
- `family_id` (STRING/INT): Interne Familien-ID (vom Datenlieferanten IFI) [21]. Gleiche ID für Patente aus derselben prioritär verknüpften Familie. Nützlich, um *eine Repräsentant-Publikation pro Familie* auszuwählen [21].
- `cpc` (ARRAY<STRUCT>): **Wichtig** – Liste aller CPC-Klassifikationen des Dokuments. Struktur etwa: `section` (STRING, z.B. "A"), `group` / `subgroup` etc., evtl. zusammengesetztes Feld `code`. Meist kann man `UNNEST(cpc) AS c` und dann `c.code` nutzen, das den vollständigen Klassifikationscode (z.B. "A61K 48/00") enthält.
- `ipc` (ARRAY<STRUCT>): Ähnlich CPC, für IPC-Klassen (wenn vorhanden).
- `us_class` (ARRAY<STRUCT>): US-Klassifikation (alte).
- `citation` (ARRAY<STRUCT>): Liste **zurückzitierter Patente** [42]. Jedes Struct hat z.B. `publication_number` (STRING des zitierten Patents) und `category` (STRING, z.B. "cited by examiner" vs "cited by applicant"). Eventuell weitere Felder wie `date`, `country`. Nutzung: `UNNEST(citation)`. Für **Forward Citations** muss man invertieren (siehe Teil E).
- `priority_claim` (ARRAY<STRUCT>): Listet Prioritätsbezüge (Anmeldenummern, Länder, Daten).

- `child` (ARRAY<STRUCT>): Weiterführende Anmeldungen (Continuation, CIP etc., nur bei US).

- **Nested Fields & Unnest**: Wie oben sichtbar, sind viele Felder Arrays. Faustregeln:

- **Einebene-Structs** (z.B. examiner) kann man mit `publications.examiner.name` direkt ansprechen.
- **Arrays** (z.B. citation, cpc, assignee_harmonized) erfordern `UNNEST`, wenn man innerhalb des Arrays aggregieren oder filtern will. Z.B. um **jede Zitierung als Zeile** zu bekommen: `SELECT p.publication_number, c.publication_number AS cited FROM patents.publications p, UNNEST(p.citation) c ...` [42].
- **Mehrfach-Arrays**: In der Regel unnested man eins nach dem anderen. Achtung: Unnesting multipliziert Zeilenanzahl (Cross Join Verhalten), daher wenn man z.B. *Assignee und CPC gleichzeitig analysieren* will, sollte man erst z.B. pro Patent aggregieren, oder mit subqueries arbeiten, um Mehrfachzählung zu vermeiden.

Google Patents Research Data – Tabelle `google_patents_research.publications`

- **Tabellenübersicht**: Gleiche Granularität wie `patents.publications` (jede Patentveröffentlichung einmal). Gleicher Primärschlüssel `publication_number`.

- **Wichtige Felder**:

- `publication_number` (STRING): Schlüsselelement, identisch zum Public Data Set [37].

- `embedding_v1` (ARRAY<FLOAT64>): 64-dimensionaler Vektor (Float-Werte). Dieser ist als Array gespeichert. Bei neueren Versionen könnte es auch `embedding_v2` etc. geben. Nutzung: Man kann diesen Vektor z.B. in **Vergleichen** nutzen (siehe Teil E, semantische Suche).
- `top_terms` (ARRAY<STRING>): Bis zu 10 Begriffe [25], repräsentativ für den Inhalt. Keine weitere Struktur, einfach Strings (können Unigramme oder Bigrams sein wie "neural network") [25].
- `translated_title` / `translated_abstract` (STRING): Die englische Übersetzung, falls Originalsprache nicht Englisch. (In manchen Releases in `title_localized` / `abstract_localized` bereits enthalten, aber hier sind sie **einsprachig vereinheitlicht**).
- `similarity_score` oder `nearest_neighbor` Felder: Möglicherweise enthält die Tabelle Precomputed Similarities (z.B. Patent X ähnlich zu Patent Y Score 0.xx). Sollte aus Beschreibung "and more" folgen [33], aber aktuell sind *embedding_v1* + *top_terms* die Hauptbestandteile.
- `url` (STRING): Der Link zur Patentansicht (z.B. "https://patents.google.com/patent/XYZ"). Praktisch, um Ergebnisse klickbar zu machen.

- Eventuell Felder für Bilder/Diagramme? (Nicht bestätigt, aber in Patentlandschaften oft relevant – hier vermutlich nur als Link im `url`).

- **Nested?**: Meist *keine komplex verschachtelten Felder* hier außer das `ARRAY` selbst (embedding und top_terms sind Arrays primitiver Typen, kein Struct). Also simpler.

- `embedding_v1` erfordert keine Unnest, außer man möchte einzelne Dimensionen als Zeilen (selten nötig). Eher berechnet man Distanz zwischen zwei embeddings z.B. via **Vectors** in SQL (COSINE_SIM etc. – es gibt BigQuery Funktionen/Modell dafür) [69].

- `top_terms` kann per `UNNEST(top_terms)` entpackt werden, z.B. um Häufigkeiten der Begriffe zu zählen.

- **Schema-Ermittlung**: Auch hier kann `INFORMATION_SCHEMA.COLUMNS` helfen. Würde man das ausführen, sähe man z.B. dass `embedding_v1` als ARRAY<FLOAT64> deklariert ist.

USPTO PatentsView – wichtigste Tabellen & Felder

PatentsView hat ein relationales Schema mit mehreren verknüpften Tabellen:

- `patent`: Enthält erteilte US-Patente (Utility, Design, Plant, Reissue).

- Felder: `id` (INT, PatentsView intern), `number` (TEXT, Patentnummer mit Kind, z.B. "5123456" oder "RE12345"), `country` (="US"), `date` (DATE, Erteilungsdatum), `title` (TEXT), `num_claims` (INT), `abstract` (TEXT), `type` (CHAR, e.g. "utility"), `kind` (CHAR, e.g. "B2"), `organization` (government interest organization, optional), `patent_year` (INT).

- `application`: US-Veröffentlichungen (Pre-Grant Publications).

- Felder ähnlich: `id`, `number` (Publikationsnr, z.B. "US20030123456A1"), `country`, `date` (Publikationsdatum), `abstract`, etc., plus `patent_id` falls schon erteilt zugeordnet.

- `assignee`: Patentinhaber/Firmen.

- Felder: `id` (INT, disambiguierte ID), `type` (TEXT, z.B. "Company"), `name_first`, `name_last`, `organization` (Wenn Firma, hier Name), `sequence` (Nummer falls mehrere Assignees per Patent).

- `inventor`: Erfinder.

- Felder: `id` (INT, disambiguiert), `name_first`, `name_last`, `location_id` (Verweis).

- `location`: Ort (Städte, Bundesstaat, Ländercode).

- Felder: `id`, `city`, `state`, `country`, `latitude`, `longitude` (bei US teils gefüllt).

- `lawyer`: Patentanwälte (mit Namen, aber idR nicht disambiguiert, da selten genutzt).

- **Verknüpfungstabellen**:

- `patent_assignee` (verbindet Patent <-> Assignee mit patent_id und assignee_id, inkl. `assignment_date` und `sequence`).
- `patent_inventor` (verbindet Patent <-> Inventor mit patent_id, inventor_id, `sequence`).
- `application_inventor`, `application_assignee` analog für Pre-grant.

- **Citations**:

- `uspatentcitation`: jeder Datensatz = Zitierung einer US-PatentID durch eine andere US-PatentID. Felder: `patent_id` (citing), `citation_id` (cited, referenziert `patent_id` in derselben Table), `date`, `category` ("cited_by_examiner"/"cited_by_applicant").
- `foreigncitation`: Zitate von US-Patenten an ausländische Patente. Felder: `patent_id` (US citing), `foreign_document_id` (Nummer string), `foreign_document_kind`, `country`, etc.
- `nplcitation`: Zitate von US-Patenten an Nicht-Patent-Literatur. Felder: `patent_id`, `citation_text` etc.
- `usapplicationcitation`: Zitate von Granted US-Patenten an US-Anmeldungen.

- **Classification Tables**:

- `cpc_current` / `cpc_group`: CPC Codes pro Patent. Enthält `patent_id`, `cpc_id` (Verknüpft zu `cpc` Tabelle mit Definition).
- `cpc` (Lookup der CPC Codes mit text).
- Ähnlich `uspc` (alte Klass.), `nber` (NBER Kategorie pro Patent), `wipo` (WIPO tech fields pro Patent).

- **Verschachtelung**: PatentsView hat **kein JSON nested** in BigQuery – alles auf Tabellen verteilt. Man nutzt JOINs statt UNNEST.

- **Hauptschlüssel**:

- Für Joins intern: `patent_id`, `inventor_id`, `assignee_id`, etc.

- Für Abgleich mit extern (Google): `patent.number` kann mit Google `publication_number` gematcht werden, aber *ACHTUNG*: PatentsView `number` hat **kein Länderkürzel** gespeichert (da immer US), und oft ohne Kind z.B. "5123456" oder mit kind? Die DB-Doku sagt *patent number is the numeric portion* – also **ohne** Kind. Für einen Grant muss man Kind ableiten (B1/B2). Besser man nutzt PatentView nur *innerhalb US-Auswertungen*, oder man bedient sich der `publication_number` in Google und filtert country_code='US' und substring auf numeric, um zu joinen. Evtl. besser: via Appl. Nr. joinen – PatentView hat aber keine Appl in Patent table, nur Pregrant.

- **Beispiel Info-Schema**: `SELECT column_name FROM ...patentsview.INFORMATION_SCHEMA.COLUMNS WHERE table_name='patent';` würde o.g. Felder zeigen. So kann man unbekannte Spalten (z.B. `num_claims`) entdecken.

USPTO PatEx (Patent Examination Dataset) – Tabellen `application_data`, `all_inventors`, ggf. weitere

- Gemäß OCE Tech-Doku besteht PatEx aus **mehreren Dateien** entsprechend den Reitern in Public PAIR [70]:
- `application_data`: wichtigste Tabelle, **eine Zeile pro Anmeldung**.
  - Felder (Auszug, siehe TechDoc und SSRN [71]):
  - `application_number` (STRING): Anmeldenummer (Seriencode+Nummer, z.B. "15/123456").
  - `filing_date` (DATE), `application_type` (INT: 1=Utility, 2=Reissue, 4=Design, 5=Prov., 6=PCT US, 7=Plant).
  - `examiner_id` (INT) & `examiner_art_unit` (INT): zuständiger Prüfer und ArtUnit.
  - `uspc_class` / `uspc_subclass`: Hauptklasse nach USPC bei Filing.
  - `status_code` (INT) & `status_description` (Text): z.B. 1 = Patented, 3 = Abandoned, etc.
  - `patent_number` (INT): Falls erteilt, die Patentnummer.
  - `patent_issue_date` (DATE): Erteilungsdatum.
  - `decision_date` (DATE): Abschlussdatum (Grant oder Abandon).
  - `foreign_priority` (INT: Anzahl ausländischer Prioritäten).
  - `small_entity_indicator` (BOOL).
  - `num_RCE` (INT, wie viele RCEs eingereicht).
  - `num_claims_allowed` (INT).
  - `earliest_pgpub_number` (Pre-grant pub Nr, falls vorhanden).
  - ... etc. (In neuem Release evtl. Patent Term Adjustment).
  - *Datentypen:* Meist INTEGER für Codes, DATE für Daten, STRING für Nummern.
- `all_inventors`: Liste aller Erfinder zu allen Anmeldungen.
  - Felder: `application_number`, `inventor_name_first`, `inventor_name_last`, `inventor_rank` (1=Haupterfinder), `city`, `state`, `country`.
  - Hier muss man evtl. nach Name oder Appl joinen, da keine disambiguierten IDs.
- `transaction_history` (falls verfügbar): Jede Änderung (Transaktion) in PAIR als Zeile, z.B. Mailings, Status changes.
  - Felder: `application_number`, `transaction_code`, `transaction_date`, `transaction_description`.
- `continuity_data`: Beziehungen zu Parent/Child apps.
  - Felder: `application_number`, `parent_application_number`, `parent_status` etc.
- `foreign_priority_data`: Prioritätsdetails (Land, Nummer, date).

- `patent_term_adjustment_data`: PTA und PTE Tage falls vorhanden.
- `attorney_agent_data`: Vertreter.

- In BigQuery sind diese evtl. als separate Tables unter dem Dataset `uspto_oce_pair` abgelegt. Mit `INFORMATION_SCHEMA.TABLES` würde man die Namen sehen (z.B. `application_data`, `all_inventors`, `continuity_data`, etc. wenn vorhanden).

- **Nested?**: PatEx-Tabellen sind flache CSV-Dumps, keine JSON Strukturen – alles relational. Keine Arrays, daher `UNNEST` nicht nötig. Eher JOINS zwischen den Tabellchen:

- Bsp: `SELECT a.application_number, a.filing_date, inv.inventor_name_last FROM ...application_data a LEFT JOIN ...all_inventors inv ON inv.application_number=a.application_number AND inv.inventor_rank=1;` – um den Haupterfinder zu bekommen.

- **Wichtige Felder**:

- `application_number` (STRING): Primärschlüssel, für alle Joins zentral.
- `patent_number` (INT/string): Patent erteilt (wenn status = Patented).
- `status_code` / `status_description`: Endstatus (z.B. Patent erteilt, abgelehnt, abandoned after RCE etc.).
- `examiner_id` & `examiner_art_unit`: Für Analysen nach Prüfer oder TechUnit.
- `filing_date`, `patent_issue_date`, `decision_date`: Erlauben Pendency-Berechnung (Decision - Filing).
- `num_RCE`: Schlüssel für Qualitätsmaß (viele RCEs = schwieriger Prozess).
- `parent_continuity_count` (mglw. im continuity Data, anzahl Eltern).
- `foreign_priority` count.

- *Mit Info-Schema*: Nachschauen, z.B. `SELECT column_name FROM ...COLUMNS WHERE table_name='application_data';` liefert genaue Feldliste und Typen.

USPTO Patent Claims Research – Tabelle(n) `patent_claims` (Grant & Publ getrennt)

- Laut Beschreibung sind es drei Files, aber eigentlich nur *zwei Typen*: Ansprüche zu Granted Patents und zu Published Applications (denn Zeiträume überlappen nicht 100%):

- Mögliche Tabellen: `granted_claims` und `pgpub_claims` (Namen angenommen).

- **Wichtige Felder**:

- `patent_number` (INT oder STRING): Patent Nr (7-digit for older, 8-digit post-2014, in 1976–2014 Range).
- `pub_number` (STRING): Bei Applications 2001–2014, Publikationsnummer (could be numeric or with year).
- `claim_number` (INT): Nummer des Anspruchs.
- `claim_text` (STRING): Volltext des Anspruchs (kann mehrere Sätze, oft endet mit " .").
- `independent` (BOOL oder INT): Kennzeichnet unabhängigen Anspruch (z.B. 1 = independent).

- Möglicherweise: `dependent_on` (INT: falls dependent, welche Claim# referenziert).
- `application_number` (vielleicht, um bei pubs zu Patent zu verknüpfen).

- **Keine verschachtelten Felder** – reine tabellarische Form. Jedes Claim als eigene Zeile, Patent mehrfach vertreten (so viele Zeilen wie Ansprüche).

- **Nutzung**:

- Evtl. Index besser auf (patent_number, claim_number) vorhanden.
- Um z.B. alle Ansprüche eines Patents in einer Zeile zu haben, müsste man Strings aggregieren.
- Aber typischer use-case: *Durchschnittliche Claim-Anzahl* → `COUNT(*) GROUP BY patent_number`.
- Oder *Claim-Länge* → z.B. `AVG(LENGTH(claim_text)) GROUP BY patent_number`.

- Info-Schema: analog, falls Tables heißen `uspto_oce_patent_claims.granted_claims` – das Information Schema würde das hergeben.

USPTO Office Actions Research – Tabellen `office_actions`, `rejections`, `citations`

- `office_actions`:
- `application_number` (STRING)
- `mail_date` (DATE)
- `oa_type` (STRING oder INT – Non-final, Final, Advisory etc.)
- `disposal_type` (vielleicht ob nach final allowed, abandoned etc.)
- `examiner_id` (wenn drin)
- `art_unit`

- **Key**: (application_number, mail_date, oa_sequence).

- `rejections`:

- Jede Zeile = ein Zurückweisungspunkt in einem bestimmten Office Action:
- `application_number`, `mail_date` (um zum Office Action zu gehören)
- `ground` (z.B. 101, 102, 103, 112 etc.),
- `claim` (Nummer des betroffenen Anspruchs),
- `reference_count` (Anzahl Zitate in dieser Ablehnung),

- evtl. `final_flag` (ob Final-Rejection).

- `citations`:

- Jede Zeile = in einem Office Action genanntes Zitat (Patent oder Nichtpatent).
- `application_number`, `mail_date`
- `citation_type` (patent vs npl),
- Wenn Patent: `cited_patent_number` + `kind` + `country`,
- Wenn NPL: `npl_text` (Auszug).

- `claim_rejected` (mglw., an welchen Anspruch gekoppelt).

- **Joins**: Offensichtlich join-key ist `(application_number, mail_date)` um Rejection/Cite dem Office Action zuzuordnen.

- **Keine JSON Strukturen**, alles flach. Häufiger aber gleiche app_number mit mehreren actions (1:N).

- Info-Schema würde die Spalten zeigen (ggf. muss man Table-Namen exakt wissen, z.B. `office_actions` oder `officeaction`).

- Hier kann `INFORMATION_SCHEMA.TABLES` helfen: `SELECT table_name FROM uspto_oce_office_actions.INFORMATION_SCHEMA.TABLES;` – sollte `office_actions`, `rejections`, `citations` zurückgeben, so weiß man es genau.

USPTO PTAB – Tabellen `trials` und `match`

- `trials`:
- `trial_number` (STRING, "IPR20XX-XXXXX"),
- `petition_date`, `institution_date`, `final_decision_date` (Dates),
- `prosecution_status` (STRING, z.B. "Pending", "Denied", "Completed"),
- `patent_number` (INT, 7/8-stellig – **ohne US, ohne Kind** in Original) [15],
- `application_number` (STRING, zugehörige Anmeldung, falls relevant – scheint es zu geben, wird im join verwendet) [16],
- `petitioner_party_name` (STRING),
- `patent_owner_name` (STRING),

- `tech_center` (vielleicht, oder "case_type" IPR/PGR).

- `match`:

- Diese Tabelle dient als Brücke. Felder:
- `patent_number` (INT) und `publication_number` (STRING im Google-Format, z.B. "US-7499872-B1") – um Patent zu matchen [20],
- `application_number` (STRING, vermutlich in gleicher Form wie im trials).

- Damit kann man *direkt* `publication_number` *aus Patent Public Data bekommen*, oder so:

    - In IFI Blog Query haben sie: `ptab JOIN ptab_match ON ptab.application_number = ptab_match.application_number JOIN patents.publications ON ptab_match.application_number = patents.application_number` [16]. Hier wird `application_number` als Join genutzt, was elegant die Patentnummer-Problematik umgeht (basiert darauf, dass in ptab.match die application_number vorhanden ist und dieselbe wie im patents.publications). Das deutet darauf hin, dass PTAB Trials Datensatz das entsprechende Appl.-Nr-Feld hat (z.B. wenn Patent contested, die Appl. extrahiert).

- **Keine verschachtelten Strukturen** hier.

- Info-Schema: `SELECT column_name FROM uspto_ptab.INFORMATION_SCHEMA.COLUMNS WHERE table_name='trials';` würde offenbaren, ob `PatentNumber` num oder string etc. (Im IFI-Artikel wird PatentNumber in BQ wohl als INT gelesen, da sie erwähnen "number format" [15]).

CPC Schema – Tabelle `cpc_scheme` (Name angenommen)

- Felder:
- `code` (STRING): Voller CPC-Code (z.B. "H04L12/24").
- `title_part` (STRING): Titeltext der Klasse.
- `level` (INT): 1=Section (A/B/C...), 2=Class (A61,...), 3=Subclass (A61K,...), 4=Group, 5=Subgroup.
- `parent_code` (STRING): Übergeordneter Code (z.B. für H04L12/24 parent ist H04L12/00).
- `ipc_concordant` (STRING): Entsprechender IPC Code (falls definiert).
- `date_revised` etc.: Wann eingeführt/aktualisiert.
- Keine verschachtelten Felder, 1 Zeile = 1 Klassifikationsnode.
- Info-Schema: Standard.

MAREC – Tabelle `documents`

- Hier basiert auf Mutmaßung:
- `document_id` (STRING): Evtl. MAREC definierte ID (z.B. "EP-123456-A1").
- `country`, `doc_number`, `kind`, `date` (int).
- `title`, `abstract`, `claims`, `description` (alle TEXT). Eventuell aufgespaltet oder ein Feld `full_text`.
- **Da es 19 Mio Dokumente sind**, vermutlich keine extreme Spaltenanzahl, eher kompakt.
- Wenn Volltexte drin sind, sind das sehr große Textfelder. BigQuery-Spaltentyp wäre STRING (max. 10MB pro Feld). MAREC-Dokumente teils mehrere 100KB (patent text), aber in Summe handlebar.
- Info-Schema: Falls MAREC da ist, könnte man so das Schema bestätigen.

UCB Fung – unbekannt, wohl ähnliches Schema wie PatentsView (Patent, Inventor etc.) aber eigen.

Patent PDF Samples – verm. eine Tabelle:

- `patent_id`,
- `pdf` (wenn als URI),
- extrahierte Entities: evtl. JSON field or separate table like `pdf_claims` or so.

SureChEMBL – Tabellen:

- `patent_compound` mapping:
  - `patent_id` (STRING, e.g. "WO2010123456"),
  - `compound_id` (INT or InChIKey)
  - `example_number` (the example in patent),
  - `context` (maybe claim or text snippet)
- `compound` details:
  - `compound_id`, `smiles`, `inchikey`, etc.

- So Abfragen oft: join patent_compound to compound to get structure of compounds in patent.

Zusammenfassend: Bei **allen Datasets** empfiehlt sich, erst einen *Blick ins Schema* zu werfen, um Feldnamen genau zu kennen, besonders bei den USPTO/OCE-Daten, wo z.B. `app_number` vs `application_number` differieren können. BigQuerys INFORMATION_SCHEMA ist hier eine große Hilfe, um unbekannte Tabellen/Felder schnell zu verifizieren.

Teil D – Join-Logik: Verknüpfung der Datensätze

Eines der Kernziele ist, **Daten aus verschiedenen Quellen zusammenzuführen** – z.B. Patentbibliografie mit Prüfungsdaten oder Klassifikationen. Hier skizzieren wir die **Join-Map** zwischen den Datasets und geben erprobte Join-Muster in SQL.

Wichtige Schlüssel und ihre Kompatibilität

- `publication_number` **(Land-Dokument-Kombination)**: Dies ist der *universelle Identifier* in vielen Tabellen – insbesondere zwischen Google Public Data und Research Data identisch [16]. Auch CPC- oder Citation-Analysen nutzen diese Form. **Format**: `"CC-NNNNNNN-KK"`, Bsp. "EP-123456-A1".

- *Kompatibilität*:

    - Google Public & Research: **ja** (exakt gleich).
    - PatentsView: **teilweise** – dort muss man Land/Kind ergänzen. Z.B. US grant in PatentsView hat `number="5123456"` und `kind="B2"`, also könnte man `"US-5123456-B2"` konstruieren.
    - OCE Datasets: meist **nein**, da diese sich auf Application- oder Patentnummern konzentrieren.
    - CPC Schema: hat keine `publication_number`, dort matched man CPC Codes, nicht Dokumente.

- `application_number` **(Anmeldenummer)**: Wichtig v.a. in **USPTO-Kontext**. Format uneinheitlich:

- Google Public: Enthält Land und formatierte Nummer (Bsp US: "US-20111345678", EP: "EP-2008-70123456").
- PatentsView: Hat `patent_id` aber Application separat nur in Application table.
- PatEx/PEDS: US-Format "NN/NNN,NNN" oder numeric string.
- **USPTO-only Joins**: PatEx <-> OfficeActions <-> PatentsView (Pre-grant) nutzen `application_number`.
- **Public vs OCE**: In IFI Public Data sind US application_number idR im Format "US########" (without slash). Möglicherweise bedarf es anpassung: z.B. PatEx "15/123456" vs Google "US15123456" oder ähnlich. Hier kann man z.B. alle Nicht-Ziffern entfernen.

- **Fazit**: application_number ist **Schlüssel für USPTO interne Verknüpfungen** (PatEx, Office Actions, PTAB match) [16], aber für **international** weniger relevant.

- **USPTO** `patent_id`: In PatentsView *interne ID*. **Nicht** in anderen nutzbar außer innerhalb PatentsView (dort join patent->inventor etc.).

- Nicht verwechseln mit Patentnummer.

- **Patentnummer (Grant)**: Meint die *Erteilungsnummer* (US 7,123,456).

- Google Public Data: In `publication_number` mit Land/KZ oder als Teil (z.B. `publication_number` von Grants hat das).
- PatentsView: im Patent-Table als `number` (ohne Land).
- PatEx: als `patent_number` (ohne Land, int).
- Office Actions: hat keine Patentnummer, da es sich auf die Anmeldung bezieht (die Patentnummer entsteht erst am Ende).
- PTAB: hat `PatentNumber` (ohne Land).
- **Mapping**: Um Patentnummern (ohne Land) mit Google `publication_number` zu verbinden, muss man Land + Kind anfügen. Bei US: Kind kann B1 oder B2 sein. *Trick*: Google Public Data hat Feld `kind_code`. Wenn uns egal ist, ob B1/B2, kann man joinen mit entweder B1 oder B2 per OR. Oder man nutzt *match*-Tabellen (siehe PTAB), die Patentnummern gemappt bereithalten [20] .

- Alternative: Join über Application als Proxy (s. unten muster PTAB).

- **Familien-IDs / Prioritätsbezüge**:

- `family_id` (IFI): nur in Google Public Data. Kann man intern nutzen (z.B. eine Repräsentant-Publikation pro Familie holen). *Nicht mit externen Quellen kompatibel*, da proprietär.

- Continuation/Parent: PatEx hat Continuity data, man kann darüber z.B. alle Divisions etc. finden. Evtl. join mit Patent grant via application_number um Familienteile zu verknüpfen.

- **CPC/IPC Codes**:

- Join zwischen Patent-Publikation und CPC-Schema geht über Code match (kein numeric join, aber Feldgleichheit).
- Google Data hat CPC in Array, man muss erst `UNNEST` CPC, dann join auf `cpc_scheme.code`.

- PatentsView hat separate CPC tables, dort direkter join CPC->scheme möglich.

- **Assignee/Inventor**:

- PatentsView vs Google: Google hat *harmonisierte Namen*, aber nicht IDs. PatentsView hat IDs. Man kann jedoch einen indirekten Join versuchen: z.B. join nach normalized name + country. Das ist unsauber und nicht empfohlen – besser man nutzt nur eine Quelle für solche Analysen. Falls gewünscht, könnte man z.B. alle Patente einer Google-Query in PatentsView matchen, indem man nach Patentnummer filtert.

- **Beispiel-Join-Map (ASCII)**:

```
Google Public (patents.publications)
    |-- join on publication_number --> Google Research
(research.publications)
    |-- join on country_code/code --> CPC Scheme (cpc_scheme)
    |-- join on publication_number (US only) --> PatentsView.patent (via US
number+kind)
    |                          \-> or join on application_number (US) ->
PatentsView.application
```

```
    |-- join on application_number (US only) --> PatEx.application_data
         |-- join on application_number --> OfficeActions.office_actions
         |                                  \-> OfficeActions.rejections/
citations (via application + mail_date)
         |-- join on application_number --> PTAB.match (to get PTAB trials)
              \--> join PTAB.match -> PTAB.trials (via application_number)
```

- **Erläuterung**:
- Für globale Analysen: oft reicht Google Public+Research+CPC.
- Für US-spezifische Pipeline:
  - Google Public mit `application_number` joinen nach PatEx (so bekommt man z.B. von einem Patent die Prüfungsdaten).
  - Von PatEx weiter nach OfficeActions via application.
  - PTAB: hat Patentnummer und Appl, wie im ASCII, meist geht man *über application*, weil Patentnummer Format tricky ist. Das *PTAB.match* erlaubt Patent -> Appl mapping. In dem IFI-Beispiel wird genau dieses 3er-Join Konstrukt genutzt [16].

Jetzt konkrete **Join-Patterns** mit SQL-Snippets:

Pattern 1: Public Data ↔ Research Data (Bibliografie + ML-Features)

Ziel: **Übersetze und Embeddings zu den Patentstammdaten holen**.
Schlüssel: `publication_number`.

SQL-Template: Alle deutschen Patentanmeldungen der letzten 5 Jahre mit ihren englischen Abstracts und Top-Embeddings holen:

```sql
SELECT pub.publication_number,
       pub.title_localized[OFFSET(0)].text AS original_title,
       gpr.abstract_en AS translated_abstract,
       gpr.top_terms,
       gpr.embedding_v1
FROM `patents-public-data.patents.publications` AS pub
JOIN `patents-public-data.google_patents_research.publications` AS gpr
  ON pub.publication_number = gpr.publication_number
WHERE pub.country_code = 'DE'
  AND pub.filing_date >= 20150101;
```

Hier nutzen wir `publication_number` 1:1 für den Join [37]. Ergebnis enthält pro Patent die Sprachen und Vektoren aus Research. *(Tipp:* `title_localized[OFFSET(0)]` *gibt den ersten Titel-Eintrag. Für DE-Patente ist das typischerweise der deutsche Titel.)*

Pattern 2: Public Data ↔ CPC Scheme (Publikationen mit Klassentext)

Ziel: **Klassentitel zu Patentdaten hinzufügen**, z.B. bei Aggregationen.

Schlüssel: CPC Code.

SQL-Template: Top 5 CPC-Unterklassen nach Anzahl Patente im Bereich "A61K" (Medizinische Präparate):

```sql
SELECT c.code AS cpc_code,
       scheme.title AS cpc_title,
       COUNT(DISTINCT pub.publication_number) AS patent_count
FROM `patents-public-data.patents.publications` AS pub,
     UNNEST(pub.cpc) AS c
JOIN `patents-public-data.cpc_scheme.codes` AS scheme
     ON c.code = scheme.code
WHERE c.code LIKE 'A61K%'
GROUP BY c.code, scheme.title
ORDER BY patent_count DESC
LIMIT 5;
```

Hier werden die CPC-Codes jeder Publikation entpackt und per `code` an die Scheme-Tabelle angebunden. So erhält man lesbare Klassentitel (z.B. "A61K - Preparations for medical, dental purposes" etc.) [72] . *(Hinweis:* `%` *Wildcard bei LIKE filtert z.B. alle Unterklassen von A61K.)*

Pattern 3: Public Data ↔ PatentsView (Anmelder/Erfinder Disambiguation)

Ziel: **Zusammenführen globaler Daten mit disambiguierten US-Entitäten**. Oft will man eine **US-Portfolioanalyse** machen: z.B. eine Firmenliste von Google Public Data mit eindeutigen IDs aus PatentsView.

Schlüssel: US Patentnummer oder Publikationsnummer. Da Format abweicht, nutzen wir am besten *Patentnummer + Jahr* oder die Appl.-Nummer.

Variante A: Join über Granted Patentnummer:

```sql
SELECT pv.assignee.organization, COUNT(*) as patent_count
FROM `patents-public-data.patents.publications` AS pub
JOIN `patents-public-data.uspto_patentsview.patent` AS pv
  ON pub.country_code = 'US'
     AND pub.grant_date IS NOT NULL        -- nur erteilte
     AND CAST(REGEXP_EXTRACT(pub.publication_number, r'-(\d+)-') AS INT64)
         = CAST(pv.number AS INT64)
     AND pv.kind = SUBSTR(pub.publication_number, -2)  -- match Kindcode
JOIN `patents-public-data.uspto_patentsview.patent_assignee` AS pa
  ON pa.patent_id = pv.id
JOIN `patents-public-data.uspto_patentsview.assignee` AS pv.assignee
  ON pv.assignee.id = pa.assignee_id
WHERE pv.assignee.organization LIKE '%Google%'
GROUP BY pv.assignee.organization;
```

Das ist komplex: Wir extrahieren die numerische Portion der Publikationsnummer (nach erstem '-') [42] und vergleichen mit `pv.number` (PatentsView Patentnummer). Zudem vergleichen wir den Kindcode (letzte 2 Stellen) mit `pv.kind`. Dies soll sicherstellen, dass wir "US-12345-B2" dem richtigen B2-Eintrag matchen. Mit dem so gematchten Patent holen wir uns via patent_assignee -> assignee den normalisierten Firmennamen. Obiger Query würde z.B. die Varianten von "Google" zählen.

Variante B: (Einfacher für Pre-grant oder inkl. Anwendungen) – **Join über Application**: PatentsView `application.number` hat Format like "US20020123A1". Diese entspricht `publication_number` in Google (bis auf Land/KZ Position). Evtl. kann man:

```
... ON pub.publication_number = pv_app.number
```

nutzen (wo pv_app = PatentsView application table). Hier muss Land und Format genau passen. Oft ist es identisch (PatentsView application.number = "USYYYYNNNNN...").

In der Praxis wird meist *nicht beides gemischt*, sondern man wählt eine Datenquelle. Aber es ist möglich, wie gezeigt.

Pattern 4: Public Data ↔ PatEx / Office Actions (Prüfungsverlauf)

Ziel: **Einen vollständigen Prüfungsverlauf anreichern** – z.B. vom Filing bis zum Grant mit allen Office Actions.

Schlüssel: `application_number` (US).

Vorgehen: 1. Public Data liefert Patent *Meta* (z.B. Titel, Technologiefeld). 2. PatEx liefert *Prüfungs Metriken* (Dauer, Outcome). 3. Office Actions liefert *detaillierte Events*.

SQL-Template: Liste aller Patente einer Firma X inklusive wie viele Office Actions durchlaufen:

```
WITH company_patents AS (
  SELECT application_number, publication_number,
ANY_VALUE(title_localized.text) AS title
  FROM `patents-public-data.patents.publications`,
       UNNEST(assignee_harmonized) AS assignee
  WHERE assignee.name = 'CROSSBAR INC'  -- Beispiel-Firma [73]
    AND country_code = 'US'
    AND grant_date IS NOT NULL
)
SELECT cp.publication_number, cp.title,
       pe.status_description, pe.patent_issue_date,
       COUNT(distinct oa.mail_date) AS office_action_count,
       MAX(oa.final_flag) AS had_final_rejection
FROM company_patents cp
LEFT JOIN `patents-public-data.uspto_oce_pair.application_data` pe
  ON REPLACE(cp.application_number, '-', '') = pe.application_number
       -- evtl. Bindestriche entfernen falls Formatunterschied
LEFT JOIN `patents-public-data.uspto_oce_office_actions.office_actions` oa
  ON oa.application_number = pe.application_number
GROUP BY cp.publication_number, cp.title, pe.status_description,
pe.patent_issue_date;
```

Erläuterungen: - Wir wählen in `company_patents` alle Patent-Publikationen einer Firma (hier Crossbar Inc, analog dem IFI Blog Beispiel) [74] . ANY_VALUE(title) weil Title mehrfach (mehrsprachig)

sein könnte. - Join mit PatEx: Wir matchen `application_number`. Je nach Format muss man evtl. 'US' entfernen oder so. Hier `REPLACE(cp.application_number, '-', '')` entfernt eventuelle Trennzeichen, damit es z.B. "15123456" vs "15/123456" passt. - Dann joinen wir Office Actions: Auf gleicher application_number. Jetzt können wir z.B. zählen, wie viele Bescheide es gab, und prüfen ob ein Final Rejection dabei war (`final_flag` in rejections Table – hier angenom. als Feld in `office_actions` zusammengeführt). - Ergebnis: Patent, Titel, Status (z.B. Patented or Abandoned), Erteilungsdatum, Anzahl Office Actions, Indikator ob Final Rejection. Das gibt ein Bild, wie aufwendig die Prüfung war. *(Bei großem Datenvolumen besser die Joins einschränken, hier z.B. nur Firma X macht es schon)*.

Pattern 5: Public Data ↔ PTAB (Challenges/Trials)

Ziel: **Patente mit PTAB-Verfahren verknüpfen** – z.B. welche erteilten Patente einer Firma wurden vor der PTAB angefochten und von wem.

Schlüssel: Via *application_number* über *ptab.match* (empfohlen).

SQL-Template: Liste aller PTAB Trials, die gegen Patente von Firma Y laufen, mit Ausgang:

```
SELECT patents.publication_number,
       patents.family_id,
       trials.trial_number,
       trials.petitionerpartyname,
       trials.prosecutionstatus
FROM `patents-public-data.uspto_ptab.trials` AS trials
JOIN `patents-public-data.uspto_ptab.match` AS pm
  ON trials.applicationnumber = pm.application_number
JOIN `patents-public-data.patents.publications` AS patents
  ON pm.application_number = patents.application_number
WHERE patents.country_code = 'US'
  AND EXISTS (
       SELECT 1 FROM UNNEST(patents.assignee_harmonized) a
       WHERE a.name = 'XYZ Corp'
      );
```

Hier nutzt man: - `trials.applicationnumber` → `ptab.match.application_number` [16]. - Und `pm.application_number` → `patents.application_number` [16]. - Danach filtert man auf Assignee = "XYZ Corp" in den Patentdaten (hier mit `EXISTS (SELECT 1 FROM UNNEST(...))` um innerhalb der Array nach Name zu suchen). - Ergebnis: Zeigt Trial-Nummer, Petitioner und Status je Patent. Family_id ggf., um zusammengehörige zu gruppieren.

(In der IFI Quelle sieht man genau dieses Muster, dort filtern sie Petitioner=Mylan und zeigen PatentOwner + Status an) [56] [16].

Zusammenfassung Join-Strategie:

- **Immer formatgleichen Schlüssel finden**: Z.B. US-Patente – ApplNr oder PatentNr – und nötigenfalls Konvertierung (z.B. Padding, Entfernen von "US").

- **match-Tabellen von Google nutzen**: z.B. `uspto_ptab.match` war extrem hilfreich um PatentNr zu ApplNr zu mappen [20]. Eventuell gibt es auch für andere (PatentsView vs Google).
- **Wenn Family/Group joinen**: Mit `family_id` können Sie gruppieren aber nicht zwischen Datasets joinen (proprietär). Besser `priority_claim` Felder nutzen oder via PatentsView continuity.
- **Country beachten**: Bei Multi-Länder-Datensätzen immer filtern, sonst könnten z.B. `application_number` aus DE vs US kollidieren (die Formate sind aber oft unterschiedlich genug).
- **JOIN + UNNEST**: Falls joinende Felder in Arrays liegen (z.B. Patent-> Assignee via Name, was in Array vorliegt), muss man unnesten oder `EXISTS` in der WHERE wie oben nutzen.

Durch diese Patterns lassen sich die vielfältigen Tabellen in kohärente Analysen überführen.

Teil E – Analyse-Module: Mögliche Auswertungsbausteine

Nun bauen wir aus den Datensätzen **wiederverwendbare Analyse-Bausteine** mit Beispiel-SQL. Die Module sind so gestaltet, dass sie für verschiedene Fragestellungen angepasst werden können. Wir betrachten jeweils: **Zweck**, **genutzte Datasets/Felder**, **SQL-Template** und **Performance-Tipps**.

Modul 1: Basic Landscape – Grundlegende Patentlandschaft

Wozu gut? – Um einen schnellen Überblick über ein Technologiegebiet oder Portfolio zu erhalten. Typische Fragen: *Wie viele Patente pro Jahr? Wer sind die Top-Anmelder? In welchen IPC/CPC Klassen wird viel angemeldet?*

Welche Datasets? – Hauptsächlich **Google Patents Public Data** (für globale Zählungen). CPC-Schema für Klassentexte. Für Assignee/Inventor kann man harmonisierte Namen nutzen (aus Google) oder PatentsView (falls nur US und man saubere IDs braucht).

SQL-Templates:

- **Patent Count pro Jahr (Timeline):** Anzahl Veröffentlichungen oder Erteilungen pro Jahr, z.B. weltweit oder pro Land. Beispiel:

```sql
SELECT FLOOR(publication_date/10000) AS year,
       COUNT(DISTINCT publication_number) AS pub_count
FROM `patents-public-data.patents.publications`
WHERE publication_date IS NOT NULL
  AND publication_date >= 20000101
GROUP BY year
ORDER BY year;
```

Erläuterung: `publication_date/10000` schneidet Monat/Tag ab [19]. DISTINCT ist hier optional, da jede pub unique ist. Für Erteilte nur `WHERE grant_date IS NOT NULL`. Filter ab 2000 verhindert sehr alte Daten, reduziert Scan. **Performance-Tipp:** Tabelle ist groß – unbedingt `WHERE` nach Datum oder Land begrenzen, sonst scannt ~90M Zeilen.

- **Top Assignees:** z.B. Top-10 Anmelder in einem Technologiefeld (CPC Filter) oder Land.

```
SELECT assignee.name AS assignee_name,
       COUNT(*) AS patent_count
FROM `patents-public-data.patents.publications`,
     UNNEST(assignee_harmonized) AS assignee,
     UNNEST(cpc) AS c
WHERE c.code LIKE 'H04L%'"  -- z.B. alle Patente in CPC H04L
(Kommunikationstech)
  AND publication_date BETWEEN 20100101 AND 20201231
GROUP BY assignee_name
ORDER BY patent_count DESC
LIMIT 10;
```

Hier wird *jede Publikation pro Assignee* gezählt (falls mehrere Assignees pro Patent, zählt Patent mehrfach; will man das vermeiden, kann man `COUNT(DISTINCT publication_number)` nehmen, aber dann werden Co-Assignee-Patente nur einfach gezählt und einer Firma zugeordnet – je nach Ziel). **Tipp:** Oft möchte man nur *ersten* Assignee (sequence=0). Dann `WHERE assignee.sequence = 0`. Die Google-Daten haben sequence glaube ich nicht, aber PatentsView hat es. Mit Google harmonized sind Co-Assignees selten, kann vernachlässigt werden oder DISTINCT.

- **Top Erfinder:** Ähnlich, aber bei Erfindern sind Namen nicht disambiguiert in Google. Besser PatentsView nutzen:

```
SELECT inventor.name_last || ', ' || inventor.name_first AS
inventor_name,
       COUNT(DISTINCT pi.patent_id) AS patent_count
FROM `patents-public-data.uspto_patentsview.inventor` AS inventor
JOIN `patents-public-data.uspto_patentsview.patent_inventor` AS pi
  ON inventor.id = pi.inventor_id
JOIN `patents-public-data.uspto_patentsview.patent` AS pat
  ON pi.patent_id = pat.id
WHERE pat.patent_year >= 2010
GROUP BY inventor_name
ORDER BY patent_count DESC
LIMIT 10;
```

Hier nutzen wir disambiguierte Erfinder (selbe Person -> gleiche ID). Performance: Der Join inventor->patent_inventor->patent ist heavy (Millionen Zeilen). Evtl. filtern auf Jahr vor dem Join, z.B. patent_year im Subselect. **Tipp:** PatentsView Indizes nutzen: z.B. erst Patent filtern auf Jahr, dann join zu Patent_inventor, dann inventor. BigQuery kann aber auch so optimieren.

- **Top CPC/IPC Klassen:**

```
SELECT SUBSTR(c.code, 1, 4) AS CPC_section, COUNT(*) AS pub_count
FROM `patents-public-data.patents.publications`, UNNEST(cpc) AS c
WHERE publication_year = 2020
```

```
GROUP BY CPC_section
ORDER BY pub_count DESC;
```

Das nimmt die ersten 4 Stellen des CPC (z.B. "H04L") – meist Sektion+Hauptklasse – und zählt Dokumente 2020. Für feinere Level kann man ganzen Code nehmen oder bis Schrägstrich.

Cost/Performance-Tipps Basic:

- Begrenze *zeitlich* (Jahrzehnteweise) oder *nach Ländern*, um unnötige Datenmengen auszuschließen.
- Verwende aggregierte Felder: z.B. Google hat `publication_year` möglicherweise als computed field (oder man berechnet wie oben). Partition auf Jahr ermöglicht Partition Pruning.
- Bei Top-N Abfragen ohne weiteren Filter: Evtl. zuerst nach Land partitionieren und dann aggregieren – BigQuery kann Partition parallel abarbeiten.
- **Sampling**: Für grobe Schätzungen kann man `TABLESAMPLE SYSTEM (1 PERCENT)` nutzen, aber bei Rankings kann das ungenau sein. Besser, wo passend, die *approx_* Funktionen (approx_count_distinct etc.) nutzen, falls Distinct counts in sehr großen Mengen nötig sind.

Modul 2: Trend Detection – Trends und Emerging Topics

Wozu? – **Wachstumsraten und neue Themen** erkennen. Etwa: *Welche Technologiefelder wachsen am schnellsten?* oder *Wann taucht ein neues Schlagwort erstmals gehäuft in Patenten auf?*

Datasets: - Für Technologiefelder: CPC/IPC + Public Data (Anmeldekurven je Klasse). - Für Schlagwörter: **Research Data** (Top Terms) oder Abstract-Text durchsuchen. - Man kann auch **PatentsView** WIPO-Technologiefelder pro Jahr nutzen.

Beispiele:

- **Wachstum neuer Veröffentlichungen pro CPC:** *Idee:* Berechne für jede CPC-Subclass die jährliche Zunahme.

```
SELECT year,
       COUNTIF(subsection = 'G06N') AS AI_related_patents,
       COUNT(*) AS total_patents
FROM (
  SELECT FLOOR(publication_date/10000) AS year,
         ANY_VALUE(
           EXISTS(SELECT 1 FROM UNNEST(cpc) c WHERE c.code LIKE 'G06N%')
         ) AS subsection
  FROM `patents-public-data.patents.publications`
  WHERE publication_date >= 20000101
  GROUP BY application_number, year
)
GROUP BY year
ORDER BY year;
```

In diesem Beispiel ermitteln wir, ob pro Anmeldung (Familie) eine CPC aus G06N (AI) vorkommt, dann aggregieren pro Jahr Anteil. Man sieht dann z.B. *Anstieg des Anteils AI-Patente über Zeit*.

Emerging Patterns: Ein starker Anstieg nach 2010 könnte auf Trend hinweisen. (Man könnte auch Wachstumsrate berechnen mit window functions über die Year-Liste).

- **Emerging Topics über Abstract/Claims:** Z.B. *neue Begriffe*: Idee: Finde Wörter, die im Zeitraum X selten waren und jetzt häufig. Eine vereinfachte Query: Zähle in Abstracts 2010 vs 2020 und sortiere nach Differenz.

```
WITH term_counts AS (
  SELECT LOWER(word) AS term,
         COUNTIF(pub_year = 2010) AS freq_2010,
         COUNTIF(pub_year = 2020) AS freq_2020
  FROM (
    SELECT FLOOR(publication_date/10000) AS pub_year,
           REGEXP_EXTRACT_ALL(abstract_text, r'\w+') AS words
    FROM `patents-public-data.patents.publications`
    WHERE country_code='US' AND publication_date BETWEEN 20100101 AND
20201231
  ), UNNEST(words) AS word
  GROUP BY term
)
SELECT term, freq_2010, freq_2020,
       SAFE_DIVIDE(freq_2020 - freq_2010, GREATEST(freq_2010,1)) AS
growth_factor
FROM term_counts
WHERE freq_2020 > 50 AND freq_2010 < 5
ORDER BY growth_factor DESC
LIMIT 20;
```

Das extrahiert alle Wörter (Achtung: *sehr heavy* – nutzt keine Fulltext Indexe) aus Abstracts. Daher nur USA und begrenzte Jahre im Beispiel. Dann identifiziert es Begriffe, die 2010 <5 mal vorkamen aber 2020 >50 mal. Das könnten **Buzzwords** sein, die neu aufkamen (z.B. "deep learning", "blockchain" etc. könnte man erwarten). **Performance:** Diese Query scannt viel Text – besser, Top Terms aus Research Data nutzen: Statt REGEXP auf Abstract, könnte man `UNNEST(top_terms)` pro Patent (embedding table) und dann Jahresvergleich machen. Das ist deutlich schneller.

- **Wachstum bestimmter Technologien**: Bsp.: *Anteil elektrischer Fahrzeuge in Autobereich.* Hier bräuchte man Kombination: E.g. CPC B60L (electric vehicles) vs B60 (all vehicles). -> Patentzahlen pro Jahr B60L vs alle B60.

- **Cost Tip:** Trends über Zeit lassen sich gut mit **Materialized Views** abbilden. Z.B. eine materialized view `patent_counts_by_year_and_field` die partitioniert nach Jahr schon gespeichert ist. So muss man Trend-Queries (die sehr häufig sind) nicht jedes Mal auf Raw Data rechnen.

- Für Terms: Evtl. **NLP Preprocessing** außerhalb BigQuery machen (z.B. mit Dataflow), da Regex in SQL auf Millionen Abstracts teuer ist. Alternative: BigQuery Text Analytics Funktionen (falls verfügbar).

Modul 3: Citation & Knowledge Flow – Zitationsanalysen

Wozu? – **Zitationsnetzwerke** analysieren: Wer zitiert wen? Wie "jung" sind Zitate? Fließt Wissen von Unis zu Firmen (oder umgekehrt)? etc. Zitate gelten als Proxy für **Wissensfluss** und Patentqualität/Relevanz.

Datasets: – **Google Public Data** hat *backward citations* im Feld `citation` (Patent -> seine Referenzen) [42]. Für *forward citations* muss man umdrehen (d.h. schauen, wer zitiert X). Oder **PatentsView** Zitations-Tabellen (z.B. uspatentcitation) nutzen – dort sind separate Zeilen pro Zitierung.

Analysen:

- **Forward/Backward Citation Count:**
- *Backward count* = wie viele Referenzen macht Patent X? (Indikator für Patentumfang an Vorwissen)
- *Forward count* = wie oft wurde Patent X von späteren Patenten zitiert? (Indikator für Impact).

SQL Beispiel (Forward Citations via Google Data):

```
SELECT cited_pub AS patent, COUNT(DISTINCT citing_pub) AS forward_citations
FROM (
  SELECT c.publication_number AS cited_pub, p.publication_number AS
citing_pub
  FROM `patents-public-data.patents.publications` p
       LEFT JOIN UNNEST(p.citation) c
       ON c.publication_number IS NOT NULL
  WHERE p.publication_date >= 20000000
)
GROUP BY cited_pub;
```

Hier erzeugt man ein Pair citing -> cited und aggregiert dann auf cited. Das ergibt pro Patent die Zahl, wie oft es zitiert wurde [26]. **Tipp:** Filter p.publication_date damit nur relevante Zeit (z.B. ab 2000) einbezogen – sonst sehr alte CITEs evtl. rauslassen. DISTINCT citing_pub, falls ein Patent mehrfach denselben zitiert (passiert normal nicht).

Mit PatentsView ginge es direkter: `SELECT citation_id, COUNT(*) FROM uspatentcitation GROUP BY citation_id` gibt Zahl forward cites je Patent.

- **Citation network Export:** Um ein Netzwerk (Kantenliste) zu erhalten, kann man ähnlich die `SELECT citing, cited` Query nutzen und z.B. auf bestimmte Menge filtern (z.B. nur Patente von Unternehmen Y als citing -> cited, um Netz zwischen Y und anderen zu sehen). Das Result kann man in ein Tool wie Gephi exportieren. Beispiel:

  ```
  SELECT p.publication_number AS citing, c.publication_number AS cited
  FROM `patents-public-data.patents.publications` p, UNNEST(p.citation) c
  WHERE p.country_code='US' AND p.filing_date BETWEEN 20100000 AND
  20150000;
  ```

Das erzeugt Tausende Kanten – lieber auf einen CPC Filter begrenzen oder spezifische Patente.

- **Zitationshalbwertszeit / Cycle-Time:** *Definition:* Halbwertszeit = Zeit bis die Hälfte der maximalen Zitate eines Jahrgangs erhalten sind. Cycle Time = Durchschn. Altersunterschied zwischen Patent und seinen Zitaten.

- Das kann man berechnen:
  - Erst extrahiere (citing_pub_year - cited_pub_year) pro Zitierung.
  - Dann median je cited_pub_year = CycleTime.

Example:

```
SELECT cited_year, APPROX_QUANTILES(citing_year - cited_year, 2)[OFFSET(1)]
AS half_life
FROM (
  SELECT FLOOR(c.publication_date/10000) AS cited_year,
         FLOOR(p.publication_date/10000) AS citing_year
  FROM `patents-public-data.patents.publications` p, UNNEST(p.citation) AS c
  WHERE p.publication_date IS NOT NULL AND c.publication_date IS NOT NULL
)
WHERE citing_year >= cited_year
GROUP BY cited_year;
```

Hier wird per cited Patent (via dessen Publikationsdatum) die Differenz zum citing Patent berechnet. Dann median (approx) aggregiert pro Jahr. *Performance:* Sehr heavy (viele Zitationspaare). Besser auf Technologiefeld beschränken oder Approximationsfunktionen nutzen (wie gezeigt).

- **Cross-Citation (wer zitiert wen nach Assignee):** Interessant: *Firmenüberschneidungen* – z.B. wie oft zitiert Firma A Patente von Firma B? Das erfordert:
  1. Für alle citations, finde Assignee von citing und cited Patent.
  2. Group by (citing_assignee, cited_assignee).

Mit Google:

```
SELECT cite_assignee, cited_assignee, COUNT(*) as citations
FROM (
  SELECT
    (SELECT ANY_VALUE(a.name) FROM UNNEST(p.assignee_harmonized) a) AS
cite_assignee,
    (SELECT ANY_VALUE(a2.name) FROM UNNEST(cited.assignee_harmonized) a2) AS
cited_assignee
  FROM `patents-public-data.patents.publications` p, UNNEST(p.citation) AS
cit
  JOIN `patents-public-data.patents.publications` AS cited
    ON cit.publication_number = cited.publication_number
  WHERE p.country_code='US' AND cited.country_code='US'
)
GROUP BY cite_assignee, cited_assignee
```

```
ORDER BY citations DESC
LIMIT 20;
```

Wir nehmen `ANY_VALUE` eines Assignee-Namens (wenn mehrere, erster). Für grobe Stats okay. Das joinen von jeder Citation an das cited Patent kostet Performance (Zitationsmenge * join overhead). Besser: PatentsView uspatentcitation mit patent_id->assignee join, dort ist es reiner SQL join, evtl. schneller.

Cost/Performance: - Zitationsanalysen können *sehr datenintensiv* sein (zig Millionen edges). Bei reinen Counts besser die voraggregierten Datensätze zu nutzen (PatentsView hat 1976-2019 alles, aber auch riesig). - Nutzen Sie *WHERE-Einschränkungen* (z.B. nur bestimmte Jahrgänge, oder bestimmte Firmen) um Graph klein zu halten. - Um Netzwerkgrafiken zu erstellen, lieber offline Tools (Spark o.Ä.) falls Netz >100k edges, BigQuery kann das Ergebnis aber in CSV schreiben. - For large forward cite counts, BigQuery now supports *approx_count_distinct* oder *HLL* – kann nützlich sein, falls deduplikate needed.

Modul 4: Claims Analytics – Anspruchsbezogene Analysen

Wozu? – Ansprüche sind *das rechtliche Herzstück* eines Patents. Analysen hier geben Hinweise auf **Patentbreite, Komplexität** etc. Metriken: *Anzahl Ansprüche*, *Durchschnittliche Anspruchslänge (Wörter)*, *Anzahl unabhängiger Ansprüche*, *Claim Tree Depth*, etc. Auch **semantische Suche über Claims** (z.B. mit Embeddings) gehört hierher.

Datasets: – Für aktuelle: Google Public Data (US claims sind enthalten als Text). Für historisch: USPTO Claims Research (1976–2014, strukturierter). Kombinierbar mit Embeddings (Google hat bislang keine claim-spezifischen embeddings öffentlich, aber man könnte Patent-Embedding approximativ dafür nutzen).

Beispiele:

- **Claim Count / Indep. Count pro Patent:**

```
SELECT publication_number,
       ARRAY_LENGTH(claims_localized[OFFSET(0)].claims) as claim_count
FROM `patents-public-data.patents.publications`
WHERE country_code='US' AND grant_date IS NOT NULL
ORDER BY claim_count DESC
LIMIT 10;
```

Hypothetisch: Falls claims_localized ist nested (möglicherweise hat es inside fields). Ich nehme an Google hat vlt. `claims_localized` mit inner structure. Im Zweifel besser OCE Claims DB:

```
SELECT patent_number, COUNT(*) as claim_count,
       COUNTIF(independent = 1) as indep_count
FROM `patents-public-data.uspto_oce_patent_claims.granted_claims`
GROUP BY patent_number;
```

Dann join mit Patent info für inhaltliche Sortierung.

- **Claim Length (Textlänge) Analyse:**

- z.B. *Durchschnittliche Wortzahl in unabhängigen Ansprüchen pro Jahr*:

```
SELECT patent_issue_year,
       AVG(word_count) AS avg_words_indep_claim1
FROM (
  SELECT FLOOR(patent_issue_date/10000) AS patent_issue_year,
         (SELECT COUNT(SPLIT(claim_text, ' '))
          FROM `patents-public-
data.uspto_oce_patent_claims.granted_claims` cl
          WHERE cl.patent_number = p.patent_number
            AND cl.independent = 1
          ORDER BY claim_number ASC LIMIT 1
         ) as word_count
  FROM `patents-public-data.uspto_oce_pair.application_data` p
  WHERE patent_issue_date IS NOT NULL
)
GROUP BY patent_issue_year;
```

Das Sub-SELECT holt die Wortanzahl des 1. unabhängigen Anspruchs. Dann mitteln pro Jahr. (Complex, aber geht. Performance moderate dank filter by year possibly.)

- **Claim Breadth Proxy:**

- Eines ist *Je kürzer ein unabh. Anspruch, desto breiter tendenziell* (weil weniger einschränkend).
- Also man könnte Sortieren, Patent mit kürzesten main claims finden.

- Oder *#Words Independent / #Words Dependent* Ratio als Proxy. Solche Kennzahlen kann man mit dem Claims DB errechnen.

- **Semantische Suche (Claims):**

- Falls man z.B. nach bestimmten Features in Ansprüchen sucht (z.B. KI-Modelle oder spezifische technische Schlagworte):

- Mit OCE DB:

```
SELECT patent_number
FROM `patents-public-data.uspto_oce_patent_claims.granted_claims`
WHERE LOWER(claim_text) LIKE '%machine learning%'
GROUP BY patent_number;
```

Dann kann man diese Patentliste weiter analysieren (Trends etc.). *Performance:* `LIKE '%...%'` auf claim_text (viel Text) ist langsam. Besser, man indexiert die Begriffe extern oder nutzt FullText search via Elastic. Aber als Prototyp geht es.

- **Embeddings:** Falls Patent-Embeddings vorhanden (die aus Title/Abstract), könnte man sie als Proxy für inhaltliche Suche nutzen. Use-case: Hat man ein Patent, will man ähnliche Ansprüche in

anderen Patenten finden. Dann würde man das Embedding aus `google_patents_research` nehmen und einen Cosine Similarity mit allen anderen berechnen (BigQuery kann seit 2022 ANN-Joins mit `VECTOR_DISTANCE` auf indexed vectors). Bsp.:

```
SELECT p2.publication_number,
       VECTOR_DISTANCE(p1.embedding_v1, p2.embedding_v1) AS similarity
FROM `patents-public-data.google_patents_research.publications` p1,
     `patents-public-data.google_patents_research.publications` p2
WHERE p1.publication_number = 'US-1234567-B2'
ORDER BY similarity
LIMIT 10;
```

Das würde die 10 nächsten Nachbarn (embedding-wise) zu Patent X finden. *Hinweis:* Embedding basiert auf Titel/Abstract, aber meist korreliert es mit Claim-Inhalten.

Performance Tipps: - Claim-Daten sind textlastig. Nutzen Sie nach Möglichkeit **strukturierte Felder** (Anzahl, Länge) statt Volltextsuche. - Für repetitive komplexe Auswertungen, erwäge **Zwischenspeicherung**: z.B. einmalig alle Patent claim counts berechnen und in Tabelle schreiben, statt jedes mal OCE CSV neu zu aggregieren. - Bei Textsuche, auf neuere DB ausweichen oder BigQuery Text Features (es gibt jetzt z.B. ML.NLPI_FEATURES, aber wohl eher Entities extrahieren). - Partitionieren nach Patent_issue_year bei OCE claims, falls man oft jahrweise auswertet.

Modul 5: Examination Analytics – Prüfungsverlauf-Auswertungen

Wozu? – Kennzahlen zur **Patentprüfung** liefern Einblicke in **Effizienz des Patentamts** und Strategien der Anmelder. Beispiele: *Durchschnittliche Zeit bis First Office Action*, *Anzahl RCE pro Fall*, *Allowance Rate*, *Abandonment Rate*, *Effekt von Appeals*.

Datasets: – **USPTO PatEx** für die Gesamtsicht (enthält Filing, Issue, Status, RCE Count), **Office Actions** für tiefergehende (First action date, count of actions), **PatentsView** oder Public Data für zusätzliche Kontext (Tech classes, etc.).

Beispielanalysen:

- **Time-to-First-Office-Action (TFFA):**
- *Definition:* Zeit zwischen Filing und erstem Bescheid.

- *Berechnung:* OfficeActions Dataset hat alle OAs – filtern auf erste pro app.

```
SELECT AVG(DATE_DIFF(first_action_date, filing_date, DAY)) as avg_days
FROM (
  SELECT oa.application_number,
         MIN(oa.mail_date) AS first_action_date
  FROM `patents-public-data.uspto_oce_office_actions.office_actions` oa
  GROUP BY oa.application_number
) f
JOIN `patents-public-data.uspto_oce_pair.application_data` app
```

```
  ON app.application_number = f.application_number
WHERE app.filing_date >= '2015-01-01';
```

Das gibt die durchschn. Tage bis FA seit 2015. Man könnte nach Art Unit gruppieren, indem man `app.examiner_art_unit` mitselektiert. **Tipp:** Um Partition Pruning zu nutzen, sollte `office_actions` ggf. nach Jahr partitioniert sein – man könnte nach mail_date >= 2015 filtern.

- **RCE-Schleifen / Abandonment / Allowance Rate:**

- *RCE-Quote:* z.B. Anteil der Patente mit >=1 RCE.
  - In PatEx: `COUNTIF(num_RCE > 0) / COUNT(*)`.
- *Allowance Rate:* z.B. % der erledigten Anmeldungen, die erteilt wurden.
  - `COUNTIF(status_code = 'P') / COUNTIF(status_code IN ('P','A'))` – P=Patented, A=Abandoned (Annahme der Codes).
  - Oder aus OCE working paper entnehmen: i.d.R ~55-60% in vielen Feldern.
- *Abandonment nach Final:* Mit Office Action dataset:
  - Z.B. *wie viele Abandonments nach 1st final rejection vs nach RCE?*
  - Man würde die Office Actions Historien anschauen und Outcome:
  - e.g. wenn status = Abandoned und es gab >=1 Final (und keine RCE?), das ist "Abandon after final".
- *Allowance nach RCE:*
  - `COUNTIF(num_RCE>0 AND status='Patented') / COUNTIF(num_RCE>0)`.

Beispiel: Allowance Rate pro Tech Center:

```
SELECT examiner_art_unit DIV 100 AS tech_center,
       COUNTIF(status_code = 1)/COUNT(*) AS allowance_rate
FROM `patents-public-data.uspto_oce_pair.application_data`
WHERE filing_date BETWEEN '2010-01-01' AND '2014-12-31'
GROUP BY tech_center;
```

(Hier angenommen status_code=1 heißt patented, und wir gruppieren ArtUnit grob in Tech Center by integer division.)

- **Durchlaufzeit (Pendency):**
- `DATE_DIFF(decision_date, filing_date, MONTH)` durchschnittlich.
- Option nach status differenzieren – erteilte vs aband.

Example:

```
SELECT status_description,
       APPROX_QUANTILES(DATE_DIFF(decision_date, filing_date, DAY), 3)
[OFFSET(1)] AS median_pendency_days
FROM `patents-public-data.uspto_oce_pair.application_data`
WHERE filing_date >= '2005-01-01'
GROUP BY status_description;
```

Das gibt median pendency für erteilt vs abgelehnt.

Performance-Tipps: - PatEx queries sind relativ flott, da <12M Zeilen. Aber detailliertere (Office actions) sind ~4.4M actions + ~1.1GB citations -> dort immer filtern, z.B. nach Jahr oder bestimmte ArtUnit. - Oft will man stratifizieren (z.B. nach Tech Center, Entity size). Dafür Indexe mental nutzen: `examiner_art_unit` ist gut, `small_entity_indicator` etc. - Falls oft ähnliche Auswertungen: lieber einmal eine Tabelle `application_metrics` erstellen: Spalten (application_number, pendency_days, rce_count>0 flag, allowed_flag, etc.). Dann kann man damit adhoc pivotten ohne immer neu zu joinen.

Modul 6: Competitive Intelligence – Wettbewerbsbeobachtung

Wozu? – Herausfinden, **wie Unternehmen interagieren und wo Lücken sind**. Beispiele: *Wer zitiert die Patente von wem?* (Cross-citations), *Welche Technologiefelder bearbeiten Wettbewerber nicht?* (White-space).

Datasets: – *Cross-citation:* braucht Zitation + Assignee Info (Public Data beides drin). *White-space:* braucht Patentklassifikationen und evtl. text cluster.

Beispiele:

- **"Wer zitiert wen" nach Assignee:** (Siehe oben Pattern bei Citation network). Das kann man noch einschränken: z.B. *nur Top-5 Mitbewerber* – man kann WHERE assignee in (list). Oder *eine Firma vs alle* – z.B. fix cite_assignee = 'Company A', gruppieren nach cited_assignee. Das zeigt, wessen Patente Company A häufig referenziert (vielleicht Konkurrenz oder Unis).

Auswertung: Man kann daraus z.B. *Technologiefelder ableiten, in denen Company A auf Vorarbeiten von B aufbaut.* (Wenn sehr einseitige Zitationsrichtung, könnte A in B's Domäne vorgedrungen sein.)

- **White Space Analyse:** *Definition:* Findet *Kombinationen von CPC Klassen und z.B. Schlagworten*, wo kaum Patente existieren, obwohl angrenzende Felder reich bestückt sind.

Vorgehen: 1. Definiere relevante CPC-Kombinationen (z.B. CPC1 + CPC2, oder CPC + Keyword). 2. Zähle Patente je Kombination. 3. Suche Kombination mit Count = 0 oder sehr niedrig, wobei Einzelbereiche hoch sind.

Beisp.: *Unterbesetzte CPC-Kombinationen bei einem Top-Thema* – Angenommen CPC "A61K" (Pharma) und "AI" als Keyword. Es gibt viel AI (G06N) und viel A61K Patente, aber wie viele mit beidem?

```
SELECT
  COUNTIF(ai_patent AND pharma_patent) as both_count,
  COUNTIF(ai_patent AND NOT pharma_patent) as only_AI_count,
  COUNTIF(pharma_patent AND NOT ai_patent) as only_pharma_count
FROM (
  SELECT pub.publication_number,
         MAX(IF(c.code LIKE 'G06N%', TRUE, FALSE)) AS ai_patent,
         MAX(IF(c.code LIKE 'A61K%', TRUE, FALSE)) AS pharma_patent
  FROM `patents-public-data.patents.publications` pub, UNNEST(pub.cpc) c
  WHERE pub.publication_date >= 20150000
  GROUP BY pub.publication_number
);
```

Wenn `both_count` deutlich kleiner als Einzel, dann Lücke. Will man wissen *wer könnte die Lücke füllen*, kann man schauen ob Firmen in AI und Firmen in Pharma disjunkt sind.

Oder White space nach *Themen-Kombi*: - Nehme Top 20 Schlagworte in Bereich X vs Bereich Y (z.B. "battery" vs "blockchain"). - Build grid und find none overlaps.

Solche Analysen brauchen aber Domänen-Knowhow um sinnvolle Kombis zu wählen.

Cost/Performance: - Cross-citation Aggregation hat O(n) ~ 100M edges -> big. Filtern auf Top N Firmen mindert deutlich Kanten (z.B. CITEs nur wo assignee in that set). - White-space loops (two-phase analysis) in SQL kann ineffizient sein. Evtl. manuell in Python nach Aggregation auswerten (z.B. get sets and do combination). - For white-space, consider using **Matrix factorization or co-occurrence** matrices externally, as BigQuery isn't optimized for large sparse matrix analysis beyond a certain complexity.

Jedes dieser Module kann je nach Bedarf angepasst werden (andere Filter, Felder). Wichtig ist, dass man dabei immer auf Effizienz achtet: Data prunen, unnötige Joins vermeiden, ggf. Zwischenschritte materialisieren.

Teil F – Kosten & Performance in BigQuery (Best Practices)

Damit BigQuery-Analysen bezahlbar und schnell bleiben, hier eine praxisnahe **Checkliste**:

1. **Minimiere gescanntes Datenvolumen**: BigQuery berechnet Kosten nach Bytes gelesen. Also:
2. **Filter so früh wie möglich**: Setze `WHERE` Bedingungen auf Partitionen/Clustering, damit unnötige Daten gar nicht erst eingelesen werden. Z.B. Analyse auf Patenten ab 2010 -> `WHERE publication_date >= 20100101`.
3. **Nur benötigte Spalten auswählen**: Vermeide `SELECT *` – das lädt alle Spalten jedes Treffers. Stattdessen liste gezielt die Felder. Beispiel: Willst du nur zählen, brauchst du z.B. keine Abstract-Texte (die können MB groß sein!). Auch Arrays: Wenn nur CPC gezählt wird, muss man nicht die riesigen Claims-Felder mitschleppen.

4. **LIMIT allein spart keine Kosten**: BigQuery scannt trotzdem die Datenmenge, auch wenn es nur 10 Ergebnisse ausgibt. Also immer Filtering statt rely on LIMIT.

5. **Verstehe Partitionierung & Clustering**:

6. **Partitionierung** teilt Tabelle nach Datum/Ranges. Queries, die Partition-Felder filtern, scannen nur entsprechende Partitionen. **Check**: in Web-UI sieht man, ob "X of Y partitions scanned".
7. **Cluster** ordnet Daten intern. Wenn nach cluster-Feld gefiltert oder sortiert wird, liest BQ weniger Blocks.
8. Patent Public Data ist wahrscheinlich partitioniert nach `publication_date` und geclustered z.B. nach `country_code` oder so. Wenn du nach Jahr UND Land filterst, profitierst du doppelt (weniger Partitionen, weniger Blocks).

9. *Praxis:* `WHERE country_code='US' AND publication_date BETWEEN 20100101 AND 20191231` – hier Partition prune über Datum, cluster prune über Land. Kostenersparnis erheblich (statt global alle Daten).

10. **Warum** `SELECT *` **teuer ist, und Alternativen**:

11. `SELECT *` liest *jede Spalte*, auch riesige Textfelder oder Arrays, obwohl du sie evtl. gar nicht nutzt. Z.B. SELECT * auf Patent-Tabelle zieht Title, Abstract, Claims (teils 100KB Text) pro Zeile. Summiert über Mio. Zeilen = viele GB.
12. **Alternative:** *Projektions-Pushdown* – BigQuery liest nur angeforderte Spalten. Also immer nur die Spalten angeben, die man wirklich braucht.

13. Faustregel: Wenn du mehr als ~5 Spalten brauchst, überleg ob du wirklich alle benötigst oder den heavy Kram auslassen kannst. Z.B. CITATION count – nimm nicht gesamte CITATION Struct, sondern `SELECT SIZE(citation)` (falls BigQuery das als on-the-fly computed field zulässt) oder unnest es, aber lade nicht die Abstracts parallel.

14. **Partition Pruning / Cluster Keys**:

15. Schon beim Tabellendesign überlegen: Wenn du eine eigene result table erstellst, partitioniere sie nach häufig genutztem Filter. Bsp: Deine "filtered_publications" mit nur Elektroautos – partitioniere nach Jahr, cluster nach Land oder CPC. Dann sind Folgereports wie "pro Jahr" quasi kostenfrei (jeder Jahr-Partition max ~1/ N).
16. Nutze das im Query: setze Filter auf Partition keys immer als direktes `WHERE`, nicht innerhalb einer subquery, sonst erkennt BQ evtl. das Pruning nicht.

17. **Überprüfe Bytes vor Ausführung**: BigQuery UI zeigt "will process 1.7 GB". Wenn das viel höher als erwartet, schau, ob Partition filter nicht greift. Evtl. Datentyp-Mismatch? (z.B. if date stored as INT, filtering with string could full scan).

18. **Sampling-Strategien**:

19. Zum Explorieren kann man `TABLESAMPLE SYSTEM (10 PERCENT)` an FROM anhängen (nicht immer verfügbar, aber neuere BQ StandardSQL hat it). Das zieht zufällig 10% der Daten – *Vorsicht*, kann ungleich verteilt sein.
20. Oder man nimmt `WHERE MOD(FARM_FINGERPRINT(key), 10) = 0` um pseudozufällig ~10% (über Hash mod 10).
21. Sampling eignet sich für Entwicklung/Debugging von Queries, sodass du nicht immer volle Kosten fährst. Für finalen Report dann auf vollen Daten laufen lassen.

22. Ebenfalls gut: **Preview-Funktion** in UI – zeigt 100 Zeilen kostenlos, um Schema und Inhalte zu checken.

23. **Materialisierte Zwischenstufen**:

24. **Warum?** Komplexe mehrstufige Analysen (z.B. erst relevante Patent-IDs filtern, dann tiefer joinen) immer wieder neu berechnen = ineffizient.
25. **Lösung:** Schreibe Zwischenergebnis in eine Tabelle (oder TEMP table). Bsp: du ermittelst ein "seed set" von 5000 Patenten in Query1, und nutzt es in Query2 via `IN (...)` – das kann ineffizient sein (IN-Liste groß). Besser: speicher die IDs in `my_dataset.seed_patents`. Dann in Query2 `JOIN seed_patents` – das skaliert.
26. **Materialized View**: BigQuery erlaubt MV, z.B. eine MV auf `SELECT assignee, COUNT(*) FROM patents GROUP BY assignee` – BigQuery pflegt diese bei Updates. In mostly static data (like these public sets updated quarterly), MV kann Summaries bereithalten. Prüfe aber, ob Access ist (manchmal nur in Enterprise etc.)

27. Im Patentfall: Could create MV for counts by year or by assignee to serve dashboards quickly.

28. **Caching/wiederverwendbare Views**:

29. BigQuery cached das Ergebnis einer Query für ~24h, **wenn exakt identische Query nochmals**. D.h. bei iterative Entwicklung kann es sein, dass neu laufen nahezu 0 B scanned (if nothing changed).
30. *Aber:* Schon ein Leerzeichen anders bricht den Cache. Man kann UI-Cache ausschalten für final run if needed.
31. **Reusable Views**: Wenn Teammitglieder immer wieder ähnliche Abfragen tippen, erstelle eine **View** mit komplexer Logik. Beispiel: eine View `us_patents_last_5yrs` als `SELECT * FROM patents.publications WHERE country_code='US' AND publication_date >= 20150101`. Die nutzt Partition und cluster. Kollegen können darauf aufsetzen wie auf einer Tabelle, ohne jedes mal Filter neu zu schreiben. Das minimiert Fehler (Filter vergessen) und garantiert, dass auch ihre Abfragen effizient sind (weil die View den Partition-Filter enthält).

32. *Authorized views:* du kannst Views bauen, die z.B. nur aggregierte oder bestimmte Spalten zeigen und extern freigeben, ohne Rohdaten preiszugeben.

33. **Denormalisierung vs Joins**:

34. In BQ darf man großzügig denormalisieren, denn Joins über riesige Tabellen kosten viel (Daten müssen shuffle).
35. Daher hat Google Patent Public Data ja z.B. alles in one table (statt separate inventor table).
36. Überlege bei eigenen Datasets: Evtl. lieber etwas Redundanz in Kauf nehmen, dafür weniger Joins. Z.B. anstatt pro Patent ID separate table mit CPCs, kann man CPCs als Array im Patent table behalten – in BQ ist das gut (nested rep.).

37. Wenn join nötig: Versuche an einem Key partitioniert/clustern. Oder nutze **Broadcast join**: BQ macht automatisch small table broadcast (bis 8MB iirc). D.h. kleine Dimensionstabelle (CPC definitions) wird verteilt – kost fast nix in Query. Also erstelle Lookup-Tabellen klein (z.B. Assignee Name -> Type), dann join => BQ cached es oft.

38. **Benutze Batch-Mode für große Jobs**:

39. BigQuery bietet *Batch Priority* an (billiger, aber kann Wartzeit haben). Wenn du Auswertung laufen lassen kannst über Nacht, stell Query auf Batch (in UI oder CLI) – es ist bis zu 5x günstiger teilweise.
40. Streaming eher meiden (teuer).

41. Wenn interaktiv, Standard (on-demand) is fine.

42. **Exporte für Downstream**:

    - Wenn die Datenanalyse in BigQuery fertig ist, oft will man Ergebnisse in anderen Tools (Excel, Python, ML).
    - **Export nach GCS**:
    - Kann direkt mit SQL: `EXPORT DATA OPTIONS(uri='gs://bucket/file.csv', format='CSV') AS SELECT ...;` – damit große Resultsets bequem als CSV/JSON oder Parquet raus.
    - Parquet bevorzugen, wenn DataFrame/Pandas oder Spark genutzt wird – behält Schema, komprimiert, schneller in Python zu laden als CSV für Millionen Zeilen.

- **Anbindung an Pandas**: Über z.B. `pandas_gbq` kann man Query direkt in DataFrame laden – aber Vorsicht, nicht zu große Mengen, sonst lieber den GCS Weg, um Memory zu schonen.
- **Feature Store**: Wenn Embeddings oder patent feature vectors gebraucht, man kann sie in BigQuery belassen und direkt aus ML Tools via BigQuery Connector nutzen (z.B. TensorFlow Data Validation mit BQ), oder export als TFRecord/CSV.

Ziel ist stets: **maximale Daten nutzen, minimale Kosten**. Mit obigen Methoden haben wir in der Praxis z.B. komplette Landscape-Reports (zig Abfragen) auf 10+ Millionen Patente für unter wenigen Dollar generiert, was mit heruntergeladenen Daten unmöglich in der Zeit ginge.

Teil G – "Distribution": Ergebnisse als Data Product bereitstellen

Nach der Analyse will man Resultate oft **weitergeben** – an Kollegen, Vorgesetzte oder in andere Systeme (BI-Tools, ML-Pipelines). BigQuery erleichtert dies mit verschiedenen Möglichkeiten. Hier sind **3 saubere Wege**, um Ergebnisse als *Data Product* verfügbar zu machen:

1. **Analystenfreundliche Views**:
2. Erstelle **dokumentierte Views**, die komplexe SQL verstecken und den Nutzern ein einfaches "virtuelles Tabelle" Interface bieten.
3. Beispiel: Du hast eine komplizierte Query, die mehrere Joins und Berechnungen enthält (z.B. *"Top 100 Assignees in AI (CPC G06N) seit 2015 mit Anteil erteilter Patente"*). Anstatt diese jedem Analysten neu zu erklären, mach eine View:

```
CREATE VIEW mydataset.v_ai_top_assignees AS
SELECT assignee_name, patent_count, grant_rate
FROM (... komplizierte Query ...);
```

4. Füge **Beschreibung/Labels** hinzu (im BQ Web UI oder CLI `bq update --description "..."`), sodass klar ist, was die View liefert.
5. Vergib aussagekräftige Spaltennamen, Einheiten etc.
6. Access: Du kannst diese View einem ganzen Team via BQ IAM freigeben, *ohne* dass sie auf die zugrundeliegenden Tabellen Zugriff brauchen (Authorized View). So sehen sie nur aggregierte Infos.
7. Vorteil: Aktualität – Views sind live (bei jedem Aufruf neu berechnet oder aus Cache), keine veralteten Excel-Exporte.
8. Anwendungsfall: z.B. das Management braucht monatlich die Zahl neuer AI-Patente pro Land. Statt jedes Mal Query, stell eine View `v_ai_patents_by_country` bereit; dann kann man die direkt in Data Studio als Quelle nehmen.

9. **Tipp:** Wenn Performance kritisch, lieber materialized view oder Table (siehe nächster Punkt). Bei moderaten Daten aber View okay.

10. **Curated Tables (persisted)**:

11. **Persistierte Tabellen** mit **Versionierung/Datum** bieten Snapshots von Analyseergebnissen.
12. Beispiel: Nach Abschluss deiner Landscape-Analyse erstellst du Tabelle `analysis.landscape_2023Q1` mit allen Metriken. Diese frierst du ein – so kann später nachgeschaut werden, was Stand Q1 war (auch wenn die Live-Daten sich ändern).

13. Bei Bedarf könntest du quartalsweise neue Versionen anlegen: `landscape_2023Q2` etc., oder ein Partitionsfeld `report_date` nutzen.
14. Diese Tables können dann z.B. ins Data Warehouse des Unternehmens integriert oder in BI-Tools eingespeist werden.
15. Vorteil: **Performance** – Abfragen auf diese kleineren, voraggregierten Tables sind sehr schnell und belasten nicht die riesigen Raw-Datasets.
16. Auch extern weitergeben: z.B. als CSV-Export – aber wenn möglich, intern als Table lassen, damit Nachnutzung (z.B. Drill-down) einfacher ist.
17. **Beispiel**: Ergebnis-Tabelle `companyX_patent_landscape` mit Spalten: assignee, filing_year, patent_count, top_CPC_section, ... Das Team kann damit eigene Pivot-Analysen bauen, ohne sich je um BQ-Kosten großer Joins zu sorgen.

18. Achte auf **Dokumentation**: Lege am besten Schema + Beschreibung ab (im Table Metadata), was die Felder bedeuten und Quelle der Berechnung.

19. **Exports für Apps/ML**:

20. Wenn Ergebnisse in **Downstream-Applikationen oder ML-Workflows** genutzt werden sollen, bietet sich *Export* oder *Direktanbindung* an.
21. **Variante A: Export als Files** – schon in Teil F erwähnt. Für ML-Modelle: Embeddings von Patenten (z.B. 64-d float) könnte man als TFRecord exportieren und dann in TensorFlow laden, um Clustering oder einen Klassifikator zu bauen.
22. Chemie-Patente: Export als Parquet und dann in Python RDKit verarbeiten.
23. **Variante B: BigQuery als Feature Store** – Manche ML-Pipelines lassen Daten direkt aus BigQuery einfließen, dank Konnektoren (z.B. TensorFlow BigQuery reader). In dem Fall könntest du ein *View oder Table mit Features pro Patent* bereitstellen. Z.B. Tabelle `ml_features.patent_vectors` mit patent_id, embedding_v1, citation_count, claim_count,... ML-Ingenieure könnten daraus Trainingsdaten zusammenstellen (mit Label z.B. "successful = granted vs not").
24. **Variante C: Integration in Apps** – Mit BigQuery REST API oder Cloud Functions lässt sich eine Applikation bauen, die Abfragen in Echtzeit stellt. Eher selten, aber denkbar: z.B. ein interaktives Dashboard wo Nutzer CPC Klassen wählen und BQ liefert Daten zurück (dann sind Ergebnisse das "Data Product" im UI).
25. Wichtig hierbei: **Zugriffskontrolle** – wenn externe App oder ein Service zugreift, sollte er nur vordefinierte Views/Tables sehen, nicht sämtliche Rohdaten. Hierfür wieder Authorized Views oder separate Dataset mit nur relevanten Ergebnistabellen nutzen.

Zusammengefasst: Man verwandelt die rohen Analyse-Resultate in **nutzbare Artefakte** – sei es ein ein-Klick-Chart in DataStudio, ein freigegebenes CSV oder eine BQ-View für andere Analysten. Dadurch lebt die Arbeit weiter und muss nicht jedes Mal neu gerechnet werden (außer man will es, dann View statt Table).

Teil H – Output-Format / Deliverables

Abschließend fassen wir zusammen, **welche konkreten Deliverables** aus so einem Projekt hervorgehen sollten:

1. **Dataset-Katalog (Tabelle)** – *erledigt oben*: Wir haben eine ausführliche Markdown-Tabelle aller relevanten Datensätze, mit Pfaden und Eigenschaften, die als Nachschlagewerk dient.

2. **Join-Map** – Wir haben die wichtigsten Join-Verbindungen beschrieben (siehe Teil D), inklusive einer ASCII-Darstellung der Beziehungen. Falls gewünscht, könnte man noch ein Diagramm zeichnen, aber textuell ist es klar. Wichtig ist, dass ein zukünftiger Nutzer versteht: *"Wie komme ich von Data A zu B?"*.

3. **SQL Snippets (10–15)** – Im Text verteilt (Teil D & E) haben wir mehrere Snippets in Markdown gesetzt. Zusammengezählt sind es ~15 kleine Query-Beispiele (z.B. Trend-Query, Citation-Query, Join-Query). Diese sind copy-paste-fähig (müssten evtl. an eigene ProjektIDs angepasst werden, aber Pfade sind so wie im Google Public Projekt angegeben).

4. **Minimal Starter Kit**:

5. **Empfehlung: 2–3 Datasets für MVP-Landscape**: Für einen schnellen MVP Patent Landscape würde ich vorschlagen:
    - **Google Patents Public Data** (`patents.publications`) – als Kern für alle Basiskennzahlen (Anmeldetrends, Assignees, Länder, CPC).
    - **CPC Schema** (`cpc_scheme`) – damit man nicht nur Codekürzel hat, sondern die Bedeutung der Technologiefelder.
    - **Google Patents Research Data** (`google_patents_research`) – optional, aber sehr hilfreich für Textanalysen (z.B. Trends von Begriffen, semantische Ähnlichkeit).
    - *Mit diesen kann man 80% einer Landscape bauen*: Timeline, Top Player, Tech Breakdown, geographische Verteilung, evtl. simple forward citations.

6. **Erweiterungen danach**:

    - **Anspruchs-Analysen**: USPTO Claims Dataset, sobald man ins Detail Patentqualität gehen will.
    - **Prüfungsdaten**: PatEx & Office Actions, wenn Fragen zur Prüfungsdauer oder -härte aufkommen (z.B. *"In diesem Techfeld dauert Erteilung länger als im Durchschnitt?"*).
    - **PTAB**: Falls relevant (z.B. im Feld sind viele Streitigkeiten), um Challenges zu tracken.
    - **Embeddings/ML**: Wenn man tiefer in semantische Ähnlichkeit, automatisches Clustering oder Prior Art Suche gehen will – dann Research Data Embeddings oder externe Modelle (Google hat Patent-BERT, siehe Kaggle).
    - **Externe Datasets**: z.B. **Litigation** (District Court Klagen – nicht in obiger Liste, aber es gibt z.B. Stanford Lex Machina dataset, wenn zugänglich) oder **Standard-Essential** (SEP) falls Standardpatente im Feld wichtig sind.
    - **Wenn international wichtig**: Evtl. PATSTAT (EPO Datensatz) zusätzlich, wobei Google Public Data das meiste davon abdeckt.

7. **Assumption Register**:

8. **Sichere Annahmen**:
    - Die BigQuery-Public-Datasets enthalten die Daten wie beschrieben (Quellen untermauert, z.B. IFI Data [28]).
    - Pfade wie `patents-public-data.patents.publications` stimmen [75] .
    - Datumsformat INT YYYYMMDD mehrfach bestätigt [19] .
    - Zitationsstruktur als Array belegt [42] .
    - OCE Datensätze Inhalt laut USPTO-Dokumentation [52] .
    - Update-Frequenzen (Google Datasets quartalsweise) [24] .

9. **Zu verifizieren**:
    - Genaue Namen mancher Datasets in BigQuery: z.B. ob das **Office Actions** Dataset tatsächlich `uspto_oce_office_actions` heißt und die Tabellen `office_actions`, `rejections`, `citations`. (Wir haben es aus dem Kaggle geschlossen, sollte aber im Cloud Marketplace geprüft werden). *Wie prüfen:* `INFORMATION_SCHEMA.TABLES` Abfrage wie gezeigt.
    - **PatentsView** in BigQuery: Es ist gelistet, aber soll man sicherstellen, ob im Projekt `patents-public-data` die Dataset `uspto_patentsview` heißt. (Wahrscheinlich ja, referenziert in Kaggle).
    - **UCB Fung Dataset Schema** – darüber gibt es kaum Info öffentlich; falls nutzen, müsste man erst via `INFORMATION_SCHEMA.COLUMNS` schauen was drin steckt.
    - **Exact content von SEP Dataset**: Wir haben Beschreibung, aber wenn wirklich nutzen, Schema inspizieren (z.B. ob es Patentnummern oder ApplIDs listet).
    - **PatEx 2020 Felder** – wir haben einiges von 2019 Doc entnommen, aber neue Release hat vlt. mehr Felder (z.B. PTAB involvement?).
10. Diese offenen Punkte würde man vor Projektfinale klären, indem man in BigQuery selbst die Schemata durchgeht oder kleine Samples zieht.

Mit diesem Dossier hat man einen **fundierten Leitfaden**, um großskalige Patentanalysen mit BigQuery erfolgreich umzusetzen – vom Verständnis der Datenquellen über das Verknüpfen bis zur fertigen Ergebnisverteilung im Team.

---

1 3 4 5 6 7 8 9 11 12 17 18 28 29 32 58 62 63 64 65 66 72 Google Patents Public Datasets: connecting public, paid, and private patent data | Google Cloud Blog
https://cloud.google.com/blog/topics/public-datasets/google-patents-public-datasets-connecting-public-paid-and-private-patent-data

2 10 22 23 30 67 68 75 Programmatic Patent Searches Using Google's BigQuery & Public Patent Data
https://www.aipla.org/list/innovate-articles/programmatic-patent-searches-using-google-s-bigquery-public-patent-data

13 14 15 16 19 20 21 26 27 42 46 55 56 57 73 74 Public patent data now available on Google BigQuery - IFI Claims
https://www.ificlaims.com/news/public-patent-data-now-available-on-google-bigquery/

24 33 38 39 40 41 43 50 59 60 61 iiindex -> datasets
https://iiindex.org/datasets/

25 34 35 36 37 Expanding your patent set with ML and BigQuery | Google Cloud Blog
https://cloud.google.com/blog/products/data-analytics/expanding-your-patent-set-with-ml-and-bigquery

31 [PDF] D3.2 Worker resilience to technology shocks - Gini-Research
https://gini-research.org/wp-content/uploads/2024/03/GI-NI_WP_3_D3.2_Final_20240209.pdf

44 45 47 48 49 70 Technical Documentation for the 2019 Patent Examination Research Dataset (PatEx) Release
https://www.uspto.gov/sites/default/files/documents/PatEx-2019-Technical-Doc.pdf

51 52 53 54 Office Action Research Dataset for Patents | USPTO
https://www.uspto.gov/ip-policy/economic-research/research-datasets/office-action-research-dataset-patents

69 Search embeddings with vector search | BigQuery
https://docs.cloud.google.com/bigquery/docs/vector-search

71 [PDF] Gendered Words and Grant Rates: A Textual Analysis of Disparate ...

https://law.depaul.edu/academics/centers-institutes-initiatives/center-for-intellectual-property-law-and-information-technology/programs/ip-scholars-conference/Documents/IPSC%202025/Papers/Marcowitz-Bitton%20-%20Paper.pdf